NTNU
Norwegian University of
Science and Technology

Refining Network Intrusion Alerts with Multi-Sensor Fusion

Emil Henry Flakk

Master of Science in Communication Technology
Submission date: June 2018
Supervisor: Yuming Jiang, IIK
Co-supervisor: Arne Øslebø, Uninett

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Refining Network Intrusion Alerts with Multi-Sensor Fusion
Student:	Emil Henry Flakk

Problem description:

Modern network security increasingly relies on good situational awareness and on the ability to quickly react to threats. A major issue with existing network intrusion detection systems (NIDS) is the high number of false positives, affecting their effectiveness in practice. In addition, they rarely jointly make use of information from different kinds of sensors, which could help provide an enhanced view of the current threat situation.

This project will look into using machine learning to analyze data collected from NIDS and other sensors (e.g., DNS, Netflow and host-based intrusion detection) to decrease the number of false positives. A realistic, monitored network environment will be created to generate sensor data. This data will be combined ("fused") and used to create a labeled training/testing dataset for anomaly detection purposes. Feature extraction will then be performed to reduce the dimensionality of the dataset, making learning more feasible. In turn, the training dataset will then be used to train classifiers using supervised classification algorithms. Finally, the resulting classifiers will be applied to the testing dataset and compared according to their ability in decreasing the false positive rate.

Responsible professor:	Yuming Jiang, IIK, NTNU
Supervisor:	Arne Øslebø, Uninett AS

Abstract

Modern Computer Emergency Response Teams (CERTs) are heavily reliant on Network Security Monitoring (NSM) in order to defend their networks from intrusions. As attacks increase in frequency and complexity, the human resources to deal with them become constrained. A particular issue is that Network IDS (NIDS) tend to produce a huge number of false positive alerts. This is in part due to the very low base rate of intrusions compared to normal traffic, leading to a base rate fallacy when classifying traffic. Experienced incident handlers use their human intuition to filter out such alerts, often looking at other sensor data to inform their situational assessment. This thesis tries to capture this intuition by applying the conceptual model of Multi-Sensor Data Fusion (MSDF), allowing for the automatic refinement of alert lists and the removal of false positive alerts, as well as potentially the detection of more sophisticated attacks. Its contribution is two-fold: First, a simple test-bed using virtual machines and NSM sensors is constructed to acquire NSM sensor data from simulated users and an attacker. Then, a graph-based feature extraction approach (RolX) and binary classifiers are applied to perform anomaly detection using data from NSM sensors. We show that, given data generated by our test-bed, commonly available binary classifiers like Artifical Neural Networks (ANN), RandomForest and State Vector Machines (SVM) perform well and are able to filter out respectively 93 %, 97 % and 94 % of false positives. Future work is also suggested to investigate and improve the applicability of these methods to more complex scenarios.

Samandrag

Hendingshandteringsteam (CERT-ar) er avhengige av såkalla nettsikkerheitsmonitorering (NSM) for å verne nettverka sine mot inntrengjing. Etter kvart som åtaka blir vanlegare og støtt meir kompliserte, så vinn ikkje dei menneskelege kreftene til lenger. Eit særskilt problem er at programvare som oppdagar innbrot i nettverket (NIDS) ofte lagar mange falske positive varslar. Dette er til dels fordi grunnraten til intrusjonstrafikk er særs låg samanlikna med harmlaus eller normal trafikk, ein såkalla bayesisk Base Rate Fallacy. Erfarne medarbeidarar tek derfor i bruk eiga røynsle og vit for å filtrere bort slike varslar, ofte ved å nytte annan NSM-sensordata for å dana seg eit oversyn over situasjonen. Denne masteroppgåva freistar fange denne intuisjonen ved å nytte den konseptuelle modellen kjend som Multi-Sensor Data Fusion (MSDF). Det er då von dette opnar for å automatisk slipe ned varsellister og kan hende oppdaga meir samansette åtak. Arbeidet bringar to bidrag til bords: Fyrst blir eit testmiljø laga for generering av NSM-sensordata frå simulerte brukarar og ein åtakar. Deretter blir ei grafbasert feature extraction-tilnærming (RolX) og binære klasseinndelarar nytta for å gjennomføre anomalideteksjon på denne sensordataen. Vi syner at klasseinndelarane Artifical Neural Networks (ANN), RandomForest og State Vector Machines (SVM) fungerer godt med såleis data, og fjernar høvesvis 93, 97 og 94 prosent av falske positive varslar. Vi føreslår også vidare arbeid for å utforske og betre nytte desse metodane i meir samansette høve.

Preface

In the 1990s, I was excited about the future, and I dreamed of a world where everyone would install GPG. Now I'm still excited about the future, but I dream of a world where I can uninstall it.
– MOXIE MARLINSPIKE

This thesis marks the end of life as a student at NTNU. The process of writing it, much like student life itself, has been both enjoyable yet at times unpredictable, however always steadfastly moving towards a set goal. Its culmination is in no small part the result of many people's assistance along the way. I am deeply indebted to my professor Yuming Jiang (IIK, NTNU) and supervisor Arne Øslebø (Uninett AS) for their patience and encouragement in bringing this thesis to its completion. Moreover, I'd like to thank my colleagues at Uninett AS in general, David Palma (IIK, NTNU), Christoffer V. Hallstensen (NTNU Digital Security), Gurvinder Singh (Uninett IoU), Boye Borg Nygård and many others for their input and friendly advice. I would also be at fault for not acknowledging my girlfriend for her immeasurable contributions to my well-being. Lastly, my life as a student would not have been half as fulfilling hadn't it been for the years spent as a volunteer at the Student Society in Trondheim (Studentersamfundet), a source of many late night discussions and friendships.

Contents

List of Figures **ix**

List of Tables **xi**

List of Algorithms **xiii**

List of Acronyms **xv**

1 Introduction **1**

1.1 Research questions 2

2 Background **5**

2.1 Network Security Monitoring and Intrusion Detection 5

2.1.1 Sensor data 8

2.1.2 Intrusion detection systems 9

2.2 NSM and Multi-Sensor Data Fusion 11

2.3 Machine Learning 12

2.3.1 Evaluating binary classifiers 14

2.3.2 Pre-processing and features 16

2.3.3 Supervised classifiers 18

2.4 Machine learning and NSM 19

2.4.1 Data acquirement 20

2.4.2 Anomaly detection using role dynamics 22

3 Methodology **25**

3.1 Fusion approach 25

3.2 Data acquirement 27

3.3 Pre-processing and feature extraction 30

3.3.1 Feature extraction 30

3.3.2 Pre-processing 31
3.4 Classification 34
3.5 Evaluation 35
3.6 Discussion 35

4 Findings and Discussion **39**
4.1 Data generation 39
4.2 Feature extraction 42
4.2.1 Pre-processing 43
4.3 Classifier results 43
4.4 Discussion 44

5 Conclusion and future work **51**
5.1 Future work 51

References **53**

A Code **57**
A.1 attack.py 57
A.2 simulate_user_traffic.py 58

List of Figures

2.1 To intercept network traffic, it might be either extracted inline the transmission cable ("tapped") or copied onto a separate switch port (commonly known as a *span* or *mirror* port).[Bej13] 8

2.2 ML process as explained by [NFP12]: First, data with known labels is pre-processed, turned into features and used to train classifiers. For evaluation, previously unseen data is pre-processed and turned into features in a similar manner. The trained classifiers then predict the labels, which is used as a basis for scoring. 13

2.3 A confusion matrix for a well-performing binary classifier (RandomForest) using randomly generated, linearly separable data. The cells are colored using a gradient according to their normalized recall. Note the brightly colored diagonal, suggesting the classifier discriminates well between different classes. 16

2.4 A Receiver Operator Characteristics (ROC) plot for a binary classifier (RandomForest) of uniformly generated, linearly separable data. Note how the False Positive Rate (FPR) asymptotically increases as the True Positive Rate (TPR) approaches $TPR = 1$, showing the trade-off between recall and the amount of false positives. 17

3.1 A high-level overview of how this research was performed. First, Network Security Monitoring (NSM) data is generated using a test-bed. Then a feature extraction and fusion process is applied to alert data in order to generate a labeled dataset, where the labels correspond to the original alert being a false or true positive. 66 % of the labeled data is used to train classifiers, and the remaining 33 % is used to evaluate the trained classifiers according to their ability to predict the labels of unseen data. 26

3.2 The topology of the test-bed used for generating NSM data. The attacker sits on an external network and interacts with the web server. A Network IDS (NIDS) is installed on the network gateway, and the web server runs a Host IDS (HIDS) agent. Both export NSM data to an ElasticSearch server (not shown) for aggregation. 28
3.3 Kibana provides a searchable view into the NSM sensor data continuously aggregated and stored in ElasticSearch. Here we can see Netflow sensor data originating from Suricata, generated for the purpose of data acquirement, the details and results of which are presented in Section 4.1. 29
3.4 A simplified situation graph. Blue vertices represent alerts triggered by Wazuh, whereas red alerts stem from Suricata. Both alerts have the IP pairs 192.168.88.90 and 172.168.88.20 in common, and thus form a network. 32

4.1 ROC Curve for SVM (with SMOTE rebalanced data) 45
4.2 ROC Curve for ann (with SMOTE rebalanced data) 45
4.3 ROC Curve for RandomForest (with SMOTE rebalanced data) 47

List of Tables

4.1 Result table for *data_smote.training.csv* 46
4.2 Result table for *data_raw.training.csv* 46
4.3 Result table for *data_over_undersampled.training.csv* 46

List of Algorithms

3.1 Algorithm for mining alert data A_{alerts} for candidate hosts and their respective NSM alert data A_{nsm} within the last N minutes. 32
3.2 Algorithm for constructing the situation graph 33

List of Acronyms

ALAC Adaptive Learner for Alert Classification.

ANN Artifical Neural Networks.

ASIDS Application-specific IDS.

AUC Area Under Curve.

CERT Computer Emergency Response Team.

DNS Domain Name System.

DPI Deep Packet Inspection.

FN False Negative.

FNR False Negative Rate.

FP False Positive.

FPR False Positive Rate.

HIDS Host IDS.

HTTP HyperText Transfer Protocol.

IDS Intrusion Detection System.

IPS Intrusion Prevent System.

JDL Joint Directors of Laboratories.

kNN *k*-Nearest Neighbor.

MLP	Multi-Level Perceptron.
MSDF	Multi-Sensor Data Fusion.
NFL	No Free Lunch.
NIDS	Network IDS.
NMF	Non-Negative Matrix Factorization.
NSM	Network Security Monitoring.
OTS	Off-The-Shelf.
PCA	Principal Component Analysis.
ROC	Receiver Operator Characteristics.
SMOTE	Synthetic Minority Oversampling Technique.
SVM	State Vector Machines.
TF-IDF	Term Frequency-Inverse Document Frequency.
TN	True Negative.
TNR	True Negative Rate.
ToS	Type of Service.
TP	True Positive.
TPR	True Positive Rate.

Chapter 1 Introduction

Information security is increasingly becoming an issue for modern organizations. As noted by one study[Pon17], whereas the average cost of per breached record has fallen from to 158 to 141 USD from 2016 to 2018, the mean number of records breached has increased 1.8 % to more than 24 000. Governments, on the other hand, need to be mindful of the consequences to national security. Besides economic costs and other factors counterproductive to organizational goals, breaches erode trust from consumers and citizens, and perhaps more pressingly, stockholders and politicians.

Modern attackers are increasingly apply sophisticated attacks[Man18], often establishing a permanent presence in the network and hiding from the owners of the affected systems. In this case, it's interesting to consider the *mean dwell time*, the amount before an attacker is evicted from the network. In 2018, one study reported this to be 100 days globally[Man18]. To make matters worse, there is an increasing skill gap between the needed knowledge to counter-act such intrusions[Man18]. Hence, the human capacity is already too limited.

Enter Computer Emergency Response Teams (CERTs), which every day try to detect, handle and limit the impact of intrusions into their networks. To do so, they often deploy Network IDSs (NIDSs), which analyze traffic and create alerts when suspicious or misuse traffic is detected. However, this often leads to a high degree of false positives. Intuitively, this is unavoidable: Intrusive traffics constitutes only a small amount of the total traffic volume, meaning even a small degree of misclassification leads to a large amount of false positives. Moreover, attackers will often try to masquerade their traffic as normal traffic, making the problem even worse.

The issue of false positives remain one of the major obstacles in effectively deploying NIDSs[JD02], and arguably most types of Intrusion Detection Systems (IDSs). These unnecessary alerts bog down incident handlers: Their limited capacities are wasted filtering away harmless alerts, instead of dealing with other pressing security concerns. Moreover, the large amount of false alerts defeats the purpose of monitoring: If most alerts are meaningless or discractive, one is less likely to care about them, which might lead to intrusions going undetected.

This work is motivated by the fact that incident handlers will use their human intuition and experience to discriminate between harmless and actual intrusion alerts. To this end, they often apply sensor data besides NIDS: For example, they might investigate logs or DNS queries performed by a particular host. This begs the question: Is there some way to capture this intuition? By doing so, we achieve two noble goals: First, we automate tedious work already performed by incident handlers. Secondly, if the approach is *data oriented*, it might grow organically as new sensors are installed into the network, saving both development time and perhaps even allowing us to detect increasingly complex attacks.

In the general sense, we therefore seek to combine different types of sensor data in order to gain more situational awareness and make the right decision: Should we pursue this alert further or not? The problem is not new, and the research field known as *Multi-Sensor Data Fusion (MSDF)* provides a set of techniques and conceptual models for precisely such problems.

Due to the short timeframe of this thesis, the scope must be tighetened somewhat to be manageable, as even intrusion detection by itself is a vast and complicated field. Therefore, it is proposed to start by filtering away false positive alerts. As such, the goal of this thesis can be considered to be the complement of typical IDS research: Instead of developing more sophisticated detection mechanism, we attack the problem from the other side and try to detect the normal traffic, allowing us to deliver more pin-pointed alert lists.

1.1 Research questions

Based on the previous discussion, the following is proposed as the main goals of this research:

R1 What is a suitable feature representation of NIDS alert data with auxiliary sensors and their contribution to operational awareness in computer networks?

R2 Is it possible to construct classifiers (based on such a representation) that are able to detect primary false positives alert data and ideally retain true positives given auxiliary sensor data?

Chapter 2
Background

This chapter will present the theoretical background of the work performed during this research. It will also present other works relevant to understanding the techniques used in the Methodology section.

First, the field of Network Security Monitoring (NSM) will be introduced and an explaination of different monitoring mechanisms (or *sensors*) that might be applied. We will also introduce the concept of Intrusion Detection System (IDS).

Then, we will provide a brief introduction to the concepts and terminology of Multi-Sensor Data Fusion (MSDF), which provides a framework for understanding how sensor data might be combined.

Then, machine learning will be presented in general. Binary classification learning problems will then be presented, with a brief introduction to research methodology and common terminology.

Finally, we will consider applications of machine learning techniques to NSM. We will particularly consider one approach ([ea18]) that has been a major inspiration to this research.

2.1 Network Security Monitoring and Intrusion Detection

When dealing with the protection of networks, we refer to the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions as NSM. The goal of NSM is not to *prevent* intrusion. Rather, it recognises that intrusion is inevitable and tries to build tools for detecting and measuring the impact of intrusions. [Bej13]

In order to better grasp the motivation for NSM, it is useful to understand what one is trying to detect. The word *kill chain* is used to describe the steps taken during intrusion into a network. The term comes from military doctrine, and its usage within information security was pioneered by Lockheed Martin[HCA11], who identifies the following phases during intrusion:

- **Reconnaissance**, a pioneering phase focused on discovering targets and enumerating their properties. Examples of actions taken in this phase include scanning network ranges for open services ("port knocking") and analysis of open sources like mailing lists.
- **Weaponization**, the selection and preparation of a digital artefact, including exploit code and some mechanism to remotely control the target after successful exploitation. This step might include applying binary obfuscation techniques to make the the payload look less inconspicuous, in order to avoid detection.
- **Delivery**. The transmission of the weaponized artefact to the target.
- **Exploitation**. The artefact's code is triggered and bypasses security boundaries in order to escalate privilegies to those of a system operator or greater (e.g. operating system routines), colloquially known as "owning" the system.
- **Installation**. The remote control mechanism is persistently put on the system in order to bootstrap the next phase. This might include installing operating system hooks to hide from antivirus software or human operators.
- **Command and Control (C2)**. The remote control mechanism establishes contact with one or more attacker-controlled server(s) in order to receive further instructions. During or before this stage, the compromised node might register itself at the attacker. Multiple control nodes and C2 methods might be employed for resilience to counter-intrusion measures and to avoid detection, usually requiring some sort of discovery mechanism.
- **Actions on Objectives**. From here on, intruders perform the necessary actions to reach the goal of the intrusion.

Within any phase of this model, NSM *sensors* that react to events (actions) in the network might be installed. As the kill chain operates at multiple layers throughout the network, e.g. actions visible on the network or only on the target host itself it is usually necessary to install several different kinds of sensors. [Bej13] identifies the following types of data generated:

– **Full content**, that is, exact copies of the traffic that passes through the sensor. The format used is commonly PCAP, compatible with popular analysis tools like Wireshark.

– **Extracted content**. These are generally the application or higher-level payloads of the traffic, such as images or text transferred across the network.

– **Session data** is a record of a conversation between two hosts. This includes information about the duration, source and destination IPs and ports, protocol, and bytes transmitted by either party.

– **Transaction data** is information about the nature of an observed transaction. This is usually specific to the application at hand: For example, transaction data about an HTTP request will tell us what page was requested, what web browser (user-agent) was used and so forth.

– **Statistical data** are aggregates or metrics about events observed on the network. By observing the session over time, one is able to calculate, say, the mean number of bytes transferred and the total duration of it.

– **Metadata** is data about the data observed on network traffic. For example, if a certain domain looks interesting, we might augment it with WHOIS data from the TLD registry.

– **Alert data** are entries created when tools in the NSM pipeline detects anomalous or suspicious behavior. Example sources include custom heuristics (e.g. "this host contacted $> n = 50$ other hosts in 5 minutes"), or, perhaps more importantly, IDSs.

The practical realisation of installing sensors can be achieved in many ways: Sensors can be installed inline on network equipment, such as firewalls or edge routers. Another approach is to intercept traffic and redirect it to dedicated equipment with sensors installed.[Bej13] With fiber optic cables, this is often achieved by installing a *fiber optic tap*, which sends some of the transmitted light through a separate path (right part of Figure 2.1. It is also possible to have a switch send copies of traffic through dedicated *span ports* (left part of Figure 2.1). What approach is

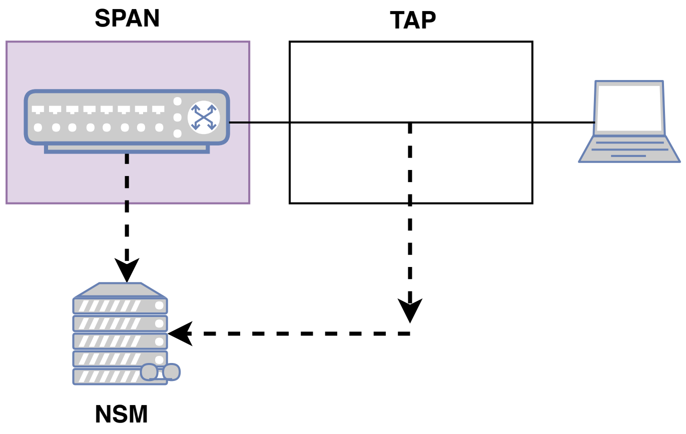


Figure 2.1: To intercept network traffic, it might be either extracted inline the transmission cable (“tapped”) or copied onto a separate switch port (commonly known as a *span* or *mirror* port).[Bej13]

used depends on the data and resource requirements of the different sensors: For example, a sensor for port knocking might be installed on a firewall, whereas a more taxing might be installed onto a dedicated, powerful server. Bear in mind that one should still *aggregate* NSM data in a central place, in order to faciliate analysis of the data[Bej13].

2.1.1 Sensor data

We will now present some concrete data representations from NSM sensors and some of their applications or characteristics.

DNS Domain Name System (DNS) is a client-server protocol used to associate records of multiple types with any domain (often called a *zone*). A zone might have additional sub-domains or *labels* with their own records. For example, an *A record* or *AAAA* record associates a domain with an IPv4 or IPv4 address respectively. This indirection achieves two main goals: It allows users to consume services without knowing IP addresses and providers to change IP addresses without notifying users. RFC2136[VT$^+$97] extends DNS to allow clients to dynamically update records. These two features make DNS particularly well-suited for implementing C2 infrastructures[DRF$^+$11][XBSY13], providing both registration of compromised hosts and a discovery mechanism for control nodes. This in turn makes DNS an important source for indicators of compromise.

Netflow Sometimes, we wish to keep metrics of and audit conversations throughout our network. Netflow is a telemetry protocol that provides us with useful information like what hosts communicates, when, for how long and so forth. It usually does this by tracking traffic with the same 7-tuple of 1) the ingress interface, 2+3) the source and destination IP addresses, 4) the IP protocol, 5+6) source and destination ports, and finally the 7) Type of Service (ToS). There also exists a 5-tuple variety, omitting the ToS and interface fields.[HČT+14] It then keeps track of when flows were initiated, and uses this information to calculate metrics like the number of bytes transferred and the number of packets in total. Hence, it is a attractive source of session data in NSM. Usually, routers in the network will *export* flow data, and a *collector* will aggregate and store it centrally.

Syslog Modern UNIX-like operating systems and others offer logging facilities that store the a consistently structured log output of applications and the operating system itself. The format expected is known as *syslog* and is currently standardized by RFC 5424 [Ger09]. By collecting syslog output, one is able to gain insight into the current situation as seen by applications and operating systems. Depending on the logging capabilities and semantics of the application, syslog might offer any kind of data, from transactional data to extracted content or statistical data. Syslog can be remotely aggregated using software like `rsyslog`.

2.1.2 Intrusion detection systems

IDSs are systems that try to detect intrusions in computer networks. If they also offer the ability to block the intrusion attempt, say by dropping the traffic, they are known as Intrusion Prevent System (IPS).

It is common to distinguish between two main IDS approaches[Pet17a]:

- **Misuse detection**, which detects unwanted behavior using sets of rules (often called signatures), for example the content of HTTP requests based on previously observed malicious traffic.
- **Anomaly detection**, which performs pattern recognition on network activity and tries detecting unexpected or deviant behavior. Such patterns might be

any number of different features of the sensor data collected: Using parametric modelling, it is possible to induce the expected distributions of such features and hence calculate the deviation, as seen in [Den87]. Non-parametric methods include clustering techniques like k-means and binary classifiers.

In an NSM context, an IDS might be considered an additional sensor that provides alert data to the system. Such alerts is generated from analysing NSM data from other sensors or its own. This is entirely dependent on the type of intrusion detection being performed. There are chiefly three different "flavors" of IDS[Pet17a]:

– **Network IDS (NIDS)**, which performs analysis in network traffic. This might entail payload inspection, statistical analysis or other heuristics.

– **Host IDS (HIDS)**, which tries to detect intrusion on a particular host using local resources like application and system logs. For example, the system might detect users running suspicious binaries.

– **Application-specific IDS (ASIDS)**, which is mainly concerned with detecting intrusion within a particular application, like a database. In the single-host case, this is similar to HIDS (albeit a less general approach), but ASIDS might also be applied to distributed applications.

We will now consider two examples of such IDSs:

Suricata is a NIDS capable of providing several sorts of NSM data. First and foremost, it performs Deep Packet Inspection (DPI) on received traffic, tracking sessions (e.g. defragmentation of packets) and matching the payload with *rules*. Such rules might consider the value of certain protocol-specific fields (like the *User-Agent* of HTTP traffic) or more low-level properties, such as the TCP flags, or any combination thereof. As such, it performs misuse detection. Upon a match, Suricata creates an *alert*, i.e. alert data. Suricata is also capable of exporting Netflow and perform several types of content-extraction, notably of DNS, HyperText Transfer Protocol (HTTP) and TLS certificates.

Suricata is a multi-platform HIDS based on OSSec. On the host, Wazuh runs as an agent communicating with a server, i.e. it uses a distributed architecture with a central control node for controlling agents and storing their outputs. Wazuh comes with rulesets for matching suspicious behavior in many common applications, such as the Apache HTTP web server and OpenSSH. In Windows, it can also track the state of the registry. Additionally, Wazuh supports tracking the integrity of processes and binaries throughout the system.

2.2 NSM and Multi-Sensor Data Fusion

MSDF is a set of techniques, processes and models that combine data from different sources in order to provide a holistic view of an entity,[HL97] a process known as *fusion*. It has seeen applications within a variety of fields, from the defense industry[HL97] to intrusion detection and [Bas00] and sharing of NSM data[And16].

By itself, MSDF contributes conceptual models for understanding how such fusion might be achieved. The techniques themselves might be borrowed from other fields, such as machine learning or statistical estimation[HL97]. As such, multiple conceptual models exists for the fusion process. We will now focus on a commonly accepted and readily applicable model for this research, the **Joint Directors of Laboratories (JDL) Fusion Loop**.

First, it must be noted that fusion is not a singular process. Different kinds of fusion might be performed: We might have data directly related to the entities themselves, but sometimes the goal is rather to consider the contribution of an entity to the current *situation* of a system. Doing so might require different types of data sources (sensors) or techniques. This is captured by distinguishing between different *levels* of processing in the JDL Fusion Loop, as explained by [HL97]:

– **Object Refinement (Level 1 Processing)**. Here, we discover the objects and provide a refined representation of them, the main goal being to arrive at a common frame of reference to aid object comparison. Moreover, auxiliary data or metadata is attached to the object in order to faciliate classification or satistical estimation of its properties.

– **Situation Refinement (Level 2 Processing)**. This describes how an object related to the surrounding environment, or compared to other objects. By doing so, we are able to contextualize the features of the object and also get a better grasp of the overall situation of the system, i.e. it provides situational awareness.

- **Threat Refinement (Level 3 Processing)**. The main goal of this process is to predict the future situation: For example, is it likely or not that these objects will constitute a threat to us in the future?
- **Process Refinement (Level 4 Processing)**. This is, perhaps somewhat counter-intuitively, a *meta-process* concerned with tuning the fusion pipeline itself. Its main responsibility is to adjust the other processes based on what new information is required, fine-tune performance for real-time tracking, and directing any sources to the appropriate processes as required by the desired inferences to be made. In a general sense, this level might be applied

It is important to note that it is not the case that the processing of data always goes from lower to higher levels. Rather, these levels allow us to conceptually differentiate between the different goals of fusion processes.

There exists work applying such conceptual models to NSM data. [Hal17] provides an overall introduction to the topic and a general fusion model for using such NSM data to perform intrusion detection and increase the situational awareness of incident handlers. [And16] presents a fusion model geared towards incident handling in the financial sector, applying machine learning methods and fusion to enhance situational awareness and aid decision making.

2.3 Machine Learning

Machine learning (also known as pattern recognition) is a discipline dealing with how computers might discover and infer structures in data[WFHP16]. In the general sense, it is concerned with how to teach computers to deal with *concepts* in the underlying dataset[WFHP16], such as whether an image is offensive or not, or if two computer users behave similarly. It is related to the discipline called *data mining*, which is concerned with extracting these concepts[WFHP16]. In practice, the terms are often used interchangeably.

When applying machine learning to a problem, one usually deals with a three step process[JDM00]: First, the data is transformed into a structure appropriate for the algorithm at hand ("feature selection/extraction" and "pre-processing"), and patterns are learned ("training"). The system is then ready to be used, and unknown data is pre-processed in a similar manner and fed into the system to measure its performance ("evaluation"). This is true for both the development of machine learning systems and

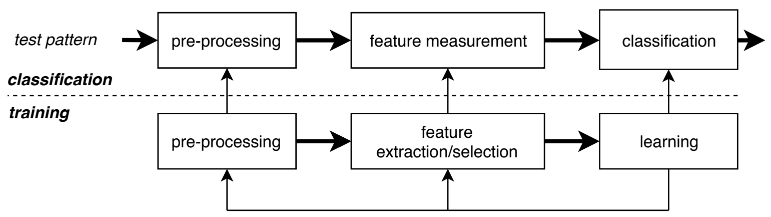


Figure 2.2: ML process as explained by [NFP12]: First, data with known labels is pre-processed, turned into features and used to train classifiers. For evaluation, previously unseen data is pre-processed and turned into features in a similar manner. The trained classifiers then predict the labels, which is used as a basis for scoring.

their application[WFHP16]. [NFP12] presents a similar model to machine learning research, which can be seen in Figure 2.2.

The output of such algorithms is highly dependent on the concept to be learned. [WFHP16] identifies four styles of learning:

– *Classification learning* tries to classify unknown examples into discrete classes based on known inputs. *Binary classification learning* methods in particular aim to determine whether the input belongs to either of two classes, e.g. “this is (not) a *hot dog*”. A subset of this is *binary logistical regression learning*, which outputs the probabilities of an input belonging to either class.

– *Clustering*, which tries to group similar examples.

– *Numeric prediction*, which predicts a numeric value, similar to interpolation in linear regression.

– *Association learning* discovers the relationships between the properties of the input, for example when learning consumer preference based upon what items the customer bought.

Likewise, the way such algorithms learn vary. *Supervised learning* deals with data where the correct answer (“label”) is known. Conversely, when the learning is *unsupervised*, the labels are not known during training.[WFHP16] *Hybrid learning* combines supervised and unsupervised techniques.

2.3.1 Evaluating binary classifiers

Machine learning is subject to the **No Free Lunch (NFL) theorem**, which states that we cannot *a priori* determine the performance of learning algorithms on any previously unseen dataset[Wol96]. Conversely, it suggests that we cannot assume one algorithm to be the best option for *all* problems, or even *any* problem. Hence, it is common to compare the performance of multiple algorithms in order to arrive at the best approach. Likewise, machine learning research relies on careful selection of methods and interpretation of results. A common concern is therefore the issue of **overfitting**, or reaching a local minima that reduces the applicability to new data.

In order to reduce the risk of overfitting, machine learning researchers generally use two distinct datasets, one for training ("training set") and one for evaluating performance ("test set"). Similarly, when dealing with algorithms whose performance is highly dependent on its parameters (such as RandomForest), researchers might introduce an additional *evaluation set* to measure the performance of different parameters. A common technique for the former is to randomly partition the data, using say 70 % of the data for training and the remainder for evaluation. For evaluation sets, a common technique is *k-folds cross-validation*, which creates k random subsamples of the training data, using one for evaluating the model and the remainder for training it[PVG+11].

How do we then measure this performance? In binary classification problems, we try to arrive at some inference function $C(k|X_0 \dots X_n)$ that outputs any label $L \in 0, 1$ for the example k with the features $[X_0 \dots X_n]$. The *performance* of any classifier then refers to statistics that measures its ability to predict the class given new data. During evaluation, L is already known for all k, and we score according to the ability to match either 0 (the negative) or 1 (the positive). Let $L*$ be the known label and L be the classifier's prediction. We then have the following outcomes:

- Classifying as $L = 1$ when $L*$ is either 0 or 1 is known as a False Positive (FP) or True Positive (TP) respectively.
- Classifying as $L = 0$ when $L*$ is either 0 or 1 is known as a True Negative (TN) or False Negative (FN) respectively.

Now let P be the number of positive examples, and N be the number of negative examples. We then have the following statistics:

– The **False Positive Rate (FPR)** (fall-out) $FPR = FP/P$

– The **True Positive Rate (TPR)** (recall) $TPR = TP/P$

– The **False Negative Rate (FNR)** (miss rate) $FNR = FN/N$

– The **True Negative Rate (TNR)** (specificity) $TNR = TN/N$

– The **accuracy** $ACC = (TP + FN)/(P + N)$

– The **Positive Predictive Value** (precision) $PPV = TP/(TP + FP)$.

– The **support** is the number of members of a particular class, i.e. the sum of TP and FN, or TN and FP for true and false positives respectively.

These statistics form the basis of most scores used within binary classification. The choice of statistics depends greatly on the goal. For example, this research seeks to increase the recall rate for certain classes (false positives), but does not particularly care for the accuracy. Another example of a common score is the **F-measure** (f1), the harmonic mean of the precision and recall, i.e. $f1 = 2(PPV \times TPR)/(PPV + TPR)$. Note that this score never accounts for the classifier's performance with TN, and therefore might not work for certain kinds of research goals.

It is often useful to visualize how these statistics fare when compared with eachother. Notably, the **Bayesian Base Rate Fallacy**[Axe99] applies when we are dealing with predicting a positive that is very rare compared to the negative, say classifying malicious traffic in a computer network. In this case, it is expected that increased recall is associated with an asymptotic increase in the FPR. Given a set of probabilities $P = (p_0, p_1, \ldots, p_n)$ for belonging to either class (according to the classifier), we can then plot the TPR against the FPR at different probability tresholds $\phi_n \in [0, 1]$. This is known as a **Receiver Operator Characteristics (ROC)** plot and is demonstrated in Figure 2.4. A particular advantage of this visualization is that is easily allows us to find the "sweet-spot" between between acceptable recall and the amount of false positives[ZOM07]. A related measure is the Area Under Curve (AUC) of the ROC, which has seen a lot of use in binary classification, but also received criticism for being a noisy statistic.[HHS$^+$10].

One might also be interested in seeing how well the classifier discriminates between different classes, both in the binary and the multi-class case. To this end, a **confusion matrix**[PVG$^+$11] might be applied. This is a two-dimensional table

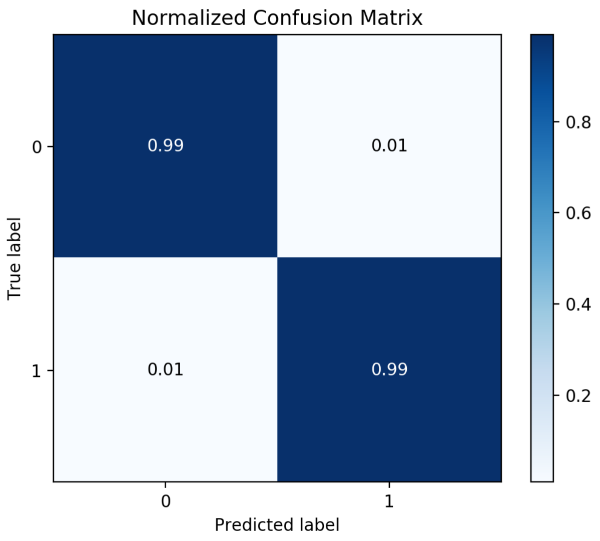


Figure 2.3: A confusion matrix for a well-performing binary classifier (RandomForest) using randomly generated, linearly separable data. The cells are colored using a gradient according to their normalized recall. Note the brightly colored diagonal, suggesting the classifier discriminates well between different classes.

of the correspondence between expected and predicted labels, often normalized to give relative probabilities. Gradients are then applied to highlight well-performing intersections. An example can be seen in Figure 2.3. Intuitively, a well-performing general classifier should have a strongly-colored diagonal, as it recalls the corrects label (independent of what label it is) most of the time.

2.3.2 Pre-processing and features

Regardless of the type of learning used, the major obstacle in machine learning is usually adapting the input data[WFHP16]. This entails cleaning up the data as-is, and also finding or extracting the *features* or attributes of the input data. Such features might be numerical or any other datatype, depending on the algorithm.

Beyond the initial concerns of removing or amending invalid or low-quality data (outliers and similar), the main goal of preprocessing is to make the data fit the input requirements: The algorithm might only support certain datatypes, often numerical values, and other datatypes (like strings) must then be converted to an appropriate

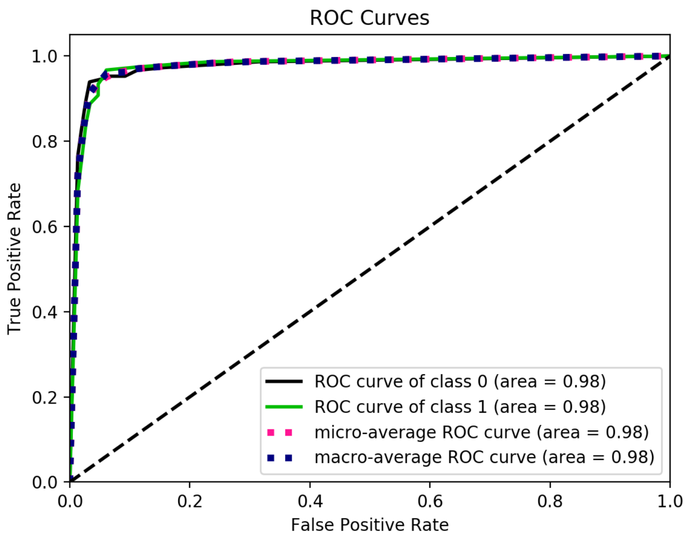


Figure 2.4: A ROC plot for a binary classifier (RandomForest) of uniformly generated, linearly separable data. Note how the FPR asymptotically increases as the TPR approaches $TPR = 1$, showing the trade-off between recall and the amount of false positives.

representation. Likewise, the algorithm might assume values lie in a discrete domain, requiring discretization of continuous values (and vice versa).

Pre-processing is not limited to the values themselves, but might also include ensuring certain statistical properties. Certain classifier algorithms in particular require *standardization*, which entails transforming the data so that it follows a Gaussian distribution with mean and variance. Similarly, *normalization* is done to ensure each sample have unit norm. [PVG$^+$11] This might be necessary to e.g. prevent certain numerically large features dominating other smaller features.

In binary classification, a particular concern might be unbalanced datasetsSynthetic Minority Oversampling Technique (SMOTE)[CBHK02], with very few examples of the minority class compared to the majority. While this is often the case with certain kinds of real data, such as when performing intrusion detection[Axe99] or fraud analysisSMOTE[CBHK02], it is detrimental if the goal is to predict the minority class, as it risks introducing overfitting. Two techniques are common to counteract this: We might *undersample* the majority class, or we might *oversample* the minority class. Sampling might then be performed uniformly with or without replacement. A technique called SMOTE[CBHK02] achieves oversampling by synthesizing examples of the minority class.

After the data has been prepared, we can find the appropriate features for conveying the concept to be learned. These might be already part of the dataset, or might need to be calculated. We might also have a highly dimensional or complicated dataset, making it hard to discover features manually. More importantly, highly dimensional datasets tend to lead to both slow learning and overfitting, a phenomenon known as the *curse of dimensionality*[KM17]. Therefore, *dimensionality reduction* is often a goal during this step of the process.

An initial approach is **feature selection**, in which case we select the best-performing existing features according to some metric. By selecting, say, any features with at least a variance[PVG+11] of 0.1, we remove features that possibly do not contain useful information to distinguish two classes of examples (as in binary classification). One might even use machine learning, like decision trees[PVG+11].

Alternatively, we might perform **feature extraction**, which computes features based on existing data. The motivation for doing so are many: Sometimes, another feature space is more fitting for the task at hand. Examples of such algorithms include Term Frequency-Inverse Document Frequency (TF-IDF) and the Fourier Transform. Other times, it also allows us to deal with huge datasets that are too complex to deal with manually. For example, by applying eigenvector decomposition, Principal Component Analysis (PCA) and similar techniques, we project the data into linear space of a lesser dimension, thus avoiding the curse of dimensionality[WFHP16].

2.3.3 Supervised classifiers

There exists a multitude of algorithms for classification learning. Common examples include:

– ***k*-Nearest Neighbor (kNN)** is a technique that classifies the example according to the class of its k nearest neighbors in feature space.[PVG+11] Hence, neighbors "vote" for their class, and the vote might be weighted and normalized in certain varieties of the algorithm. Similarly, ***k*-means** classifies the example such that the distance between its attributes and the mean values of already known clusters of classes is minimized[PVG+11]. The distance measure might vary[PVG+11], but the Euclidian and Hamming distance are common for numerical and string features respectively.

– **Decision trees**. This approach builds a decision tree according to some *splitting criteria*[PVG+11]: Often, one splits the dataset with the attribute that contributes to the lowest amount of aggregate information information theoretic entropy. Intuitively, as a higher entropy suggests a more disordered set, the split with the lowest entropy leads to more ordered data. A similar approach might be taken with Gini impurity as the metric[PVG+11].

– Similarly, **RandomForest** is an ensemble of N decision trees with a randomized splitting criteria, i.e. the attribute for splitting is selected at random. [PVG+11] Each sub-tree is then allowed to classify the example, and the final classification then decided by having each tree vote[PVG+11] for its preference.

– **Multi-Level Perceptron (MLP)** is a form of Artifical Neural Networks (ANN). It starts with a network of *neurons* organized in layers[PVG+11], where each neuron in layer L is connected to each neuron in layer $L+1$ using weighted, directed edges. A neuron applies a *activation function* (commonly sigmoids) to its weighted inputs, forwarding the result to the next layer. In the case of binary classification, the first layer receives any features $X = [x_0, x_1, ..., x_n]$ and the final layer is a single perceptron outputting a result in the domain $[0, 1]$. After the initial *forward-feeding* and resulting prediction, the error of the prediction is calculated, and a technique called *backpropagation*[PVG+11] is applied to adjust the weights of each neuron's incoming edges.

– **State Vector Machines (SVM)**. This approach seeks a hyperplane, which is a line through a p dimensional space such that it is divided into $p-1$ partitions, and which maximized the distance between the points closest to the hyperplane (the *margin of separation*). Formally, a hyperplane is defined by any points x such that $w \times x - b = 0$. As the input space might not be linearly separable, a kernel function $\phi(x)$ is used to nonlinearly map any inputs $X = X[x_0, x_1, ..., x_n]$ into a higher-dimensional feature space F that facilitates the computation of the optimal hyperplane using a linear approach.[HDO+98]

2.4 Machine learning and NSM

Machine learning has been applied to detect intrusions in computer networks. In fact, anomaly-based IDSs can be considered a special case of binary classification. Early anomaly-based intrusion detection systems pursued parametric models[Den87], but modern machine learning techniques are increasingly being applied. Some examples are offered to illuminate the wide range of applications:

– [ZXXW15] applies decision trees (J48) to novel DNS features, in combination with misuse- and anomaly-based IDSs, in order to detect malware traffic based on e.g. C2 traffic.

– [And16] applies feature selection to fused NSM data in order to improve classifier performance for intrusion detection and threat intelligence, as well as enabling data-oriented, privacy-aware feature sharing between incident handlers.

– A system known as SENATUS[ASØ12] uses a vote-based anomaly detection scheme based on a set of representative flows (from Netflow) and PCA to detect anomalous flows.

– Adaptive Learner for Alert Classification (ALAC) is a so-called human-computer interaction model, where human analysts use their implicit expert knowledge to classify alerts as either true or false positives to train a classifier. By doing so, the system is increasingly able topredict human expectations and aid the analyst in reducing the number of false positives.[Pie04]

– [JNVDÅ17] offers a novel anomaly-based machine learning approach to detecting attacks on telecom infrastructure by discovering anomalies in SS7 traffic.

– [ea18] constructs a system for detecting complex kill chains via a Bayesian logistical regression approach. An unsupervised feature extraction approach is applied to a graph of NSM data. This approach is explained in more detail in Section 2.4.2.

2.4.1 Data acquirement

There exists multiple well-known NSM datasets for evaluating and training machine learning-based IDSs. They are usually the easier option compared to performing research on data from real-world networks, which comes with a set of ethical and privacy concerns. Moreover, research data might often be more readily shared between researchers, making it easier to perform and reproduce research. However, there are usually disagreements about their applicability[Pet17b]: A particularly well-known labeled dataset is KDDcup99[TBLG09]. In one meta-evaluation, KDDcup99 is found to be be non-representative of real networks, suffering from parameter issues like low TTL values, and to have too different datasets for learning and testing to achieve meaningful learning outcomes[Eng10]. Similar critiques apply to other common datasets, particularly in IDS research.

As such, there exists no clear-cut criteria for the quality of a dataset in general. However,[SSTG12] suggests there are five such criteria that should generally apply:

- **Realistic network and traffic**. In order to be applicable in the real world, the network should look familiar to “ordinary” networks, with users, servers and so forth. This might include a realistic relative distribution of certain protocols. For example, one would expect HTTP to be very common in a user network, SSH less so. Similary, it is common for intrusions in particular to have a low base rate[Axe99] compared to other types of traffic. Hence, the amount of intrusive traffic should be much smaller than the amount of normal traffic.
- **Labeled dataset**. Manual labeling of traffic as e.g. either normal or anomalous is laborious and thus impractical for researchers.
- **Total interaction capture**. All interactions between hosts and services on the network should be provided, in order to fully understand and interpret the results.
- **Complete capture**. Research data often provides only the resulting network traces or heavily anonymized payloads. This makes it difficult to recreate, expand upon and discover new results.
- **Diverse intrusion scenario**. Modern attacks are becoming increasingly creative, bespoke and complex[SSTG12]. Hence, it is essential to develop IDS that are effective against such attacks in order to keep up with emerging threats.

The authors then evaluate five common datasets in IDS research according to these criteria. Similarly, [GSLG16] extends these criteria and evaluate eleven datasets using the extended criteria.

Among the criteria mentioned, the requirement of *realism* is perhaps the largest challenge. In response, [SSTG12] proposes solution based on *profiles*. The idea behind this is to generate recipes and formal parameters that allow researchers to recreate and adapt the setup behind the dataset to their needs, and also verify that the result is reproducible. In this sense, it can be seen as an empirical induction over discrete simulation techniques. Two different classes of profiles are proposed:

- β-profiles extract behaviors of actors within the network, such as the relative distributions in time, type and volume of traffic. Using these parameters, individual agents might recreate the desired traffic patterns and even payloads. Such profiles might be mined from real or test-bed networks in order with the desired structure.
- α-profiles are unambigious "recipes" for attacks. Such profiles might model kill chains (multi-stage attacks) and be carried out by either machines or the researchers themselves.

2.4.2 Anomaly detection using role dynamics

[ea18] performs anomaly detection using NSM data from OSSec (a common HIDS) and Snort (a common NIDS). The data was acquired using a virtual, heterogenous network similar to a corporate network.

To understand their approach, first consider that an entry in the NSM system might be considered to be a collection of *attributes* and their corresponding *values*, or simply *attribute-value pairs.* We can then express the entry as a graph of vertices of such pairs. Then, let for example the attribute-value pair *IP = 192.168.88.5* appear in multiple entries. By linking corresponding attribute-value pairs, the sub-graphs of these entries will then be linked and we correlate seemingly related logs.

Given such a graph, it is possible to mine for patterns. [ea18] uses RolX, an unsupervised algorithm for extracting *structural roles* from graphs[HGER+12]. The goal of the algorithm might be understood as identifying different behaviors of vertices: Some nodes are outliers, whereas other nodes might be belong to the critical path of many of the graph's sub-graphs (i.e. they are central). The induced roles are complementary and distinctly different from *communities* in graphs: The latter deals with inter-connected nodes, whereas roles deal with structural properties across the whole graph. Hence it can be argued that unlike communities, roles transfer across when analysing separate (disconnected) graphs[HGER+12].

[HGER+12] describes RolX in roughly two steps:

– First, an empty matrix V is initialized. Features are calculated for each node based on a selection of graph metrics and appended to V. This is repeated until convergence (i.e. recursively), with interleaved trimming of columns to reduce the amount of redundant information after each round.

– Non-Negative Matrix Factorization (NMF) is performed on V to find a decomposition $V \approx G \times F$, where G is known as the *node-role matrix* and F is called the *sense-making matrix.* G_n in particular is a row of length r decomposing any vertex n into r different roles. F is used to relate the discovered roles to well-understood graph features like the PageRank score.

By applying this algorithm on a virtual network, [ea18] ends up with time series of role decompositions for any node in the graphs, i.e. attributes. In the case of the attribute being an IP address, this assigns some role to a host. Then, using an approach known as *role dynamics*[RGNH13][RGNH12], a time-oriented normalcy model of expected ("normal") roles are then constructed. Finally, sophisticated attacks are injected into the network, analysed using the same role-decomposition approach and detected under the condition that they deviate greatly from the roles predicted by the normalcy model.

Chapter 3 Methodology

Having presented the prerequisite theory in Chapter 2, we will now discuss how this research was performed. A high-level explaination of the overall methodology of this thesis is shown in Figure 3.1.

First, we will briefly frame our research goals in terms of MSDF to provide a conceptual overview of the steps taken to achieve these goals.

Then, the data acquirement process will be described, including the test bed used to create the NSM sensor data used for machine learning.

Moving on, the procedure for feature generation and pre-processing will be shown. After this, we will consider the classifiers and evaluation procedure. The resulting performance of these classifiers according to this procedure will be presented and discussed in the next chapter.

Finally, a brief discussion will consider the overall choice of methods and justify them with regards to the previous theoretical background in Chapter 2, as well as the stated research goals and other concerns.

3.1 Fusion approach

With regards to Research Question 1 and 2 (Section 4.4), the overall problem we are trying to solve is the following: Let A_{alert} be alert data created by a NIDS. Given the superset A_{nsm} containing data from multiple sensors (including the NIDS), we wish to classify any alert in A_{alert} as either credible or not credible, i.e. either a true or false positive respectively. We will now explore how fusion processes relate to the process of solving this problem.

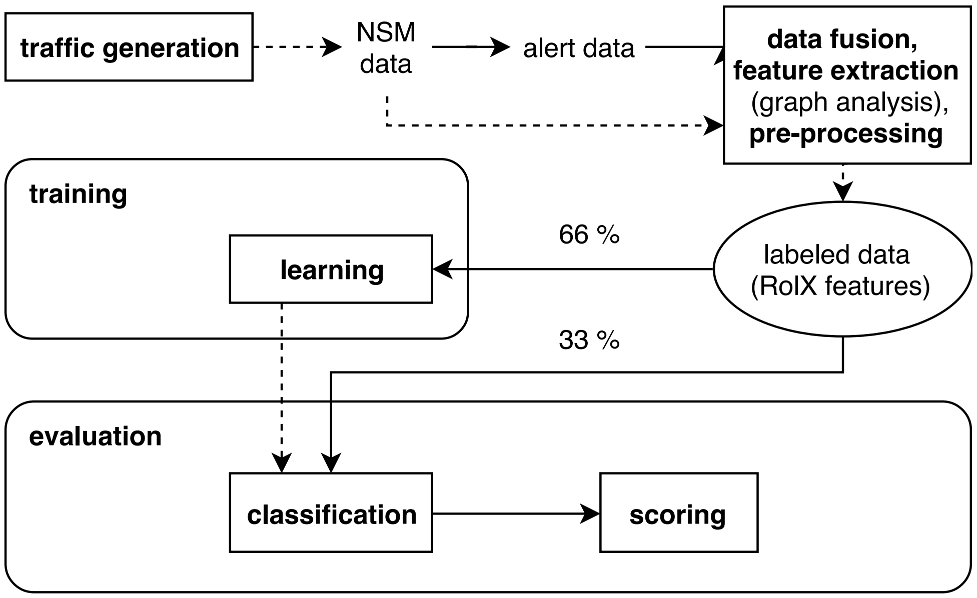

traffic generation
NSM data
alert data
data fusion, feature extraction (graph analysis), pre-processing
training
learning
66 %
labeled data (RolX features)
33 %
evaluation
classification
scoring

Figure 3.1: A high-level overview of how this research was performed. First, NSM data is generated using a test-bed. Then a feature extraction and fusion process is applied to alert data in order to generate a labeled dataset, where the labels correspond to the original alert being a false or true positive. 66 % of the labeled data is used to train classifiers, and the remaining 33 % is used to evaluate the trained classifiers according to their ability to predict the labels of unseen data.

Initially, a common frame of reference is required, or an Object Refinement process. NIDS alerts react to specific conversations initiated hosts, and it is the intention of the hosts that determines our classification: An attacker-controlled host creates *true* alerts, whereas an innocent user creates *false* alerts. Moreover, actions performed by a host is registered by the sensor at some given time. Therefore, it makes sense to considers hosts as our objects, with any sensor data correlated by time and the host. We will refer to the data correlated this way as an *event list*, as the data is made up of events creating an impulse in the sensor.

Given these objects and their associated event lists, we now wish to compare objects in context, i.e. by their contribution to the current environment. This is an Situation Refinement process, and the goal of doing so is in this case to enable a Threat Refinement process: Given these characteristics, we wish to predict the intent of any host, i.e. classify it as an attacker or not. Therefore, we need to train classifiers to realize this step.

Finally, having achieved our classification of the host, we are able to filter out alerts that, given their contextual evaluation, do not seem to be true positives or real intrusion traffic. That is, as intent determines the nature of these alerts, hosts that are *not* classified as attackers will have their alerts removed from the alert list. Thus, we achieve our goal of removing false positive alerts.

3.2 Data acquirement

An initial requirement of this research was to combine NIDS alert data with HIDS and any number of other sensors. This presents a couple of limitation: First of, there are few datasets containing the necessary data. Even when full packet captures are available, these are not immediately applicable: Stateful TCP traffic is not trivially replayable (unlike UDP traffic, which uses stateless connections), making it hard to induce some response in e.g. most HIDSs.

Given these concerns, two choices immediately arise: Either simulate the sensor impulse with existing research data or generate the traffic with Off-The-Shelf (OTS) sensors commonly used in real-world networks. In the interest of time and real-world applicability, the latter approach was selected, as it was considered too burdensome to correctly emulate realistic network sensors.

In order to satisfy the *realism* criteria suggested in Section 2.4.1, it was decided to use virtual machines running Linux, a common operating system for services. The structure of the test-bed is shown in Figure 3.2. It has the following main characteristics:

- Hosts belong to either an *a home net* (internal network) or an external network. The ranges 192.168.88.0/24 and 172.168.88.0/24 were respectively allocated towards these networks. The gateway connects these two networks together and routes traffic between them. The networks are switched separately and thus do not share an L2 collision domain, which would introduce the risk of hosts performing L2 communication and bypassing NSM sensors.

- The internal network is monitored internally by a HIDS running on exposed services, in this case a single web server running a blog. Moreover, any traffic

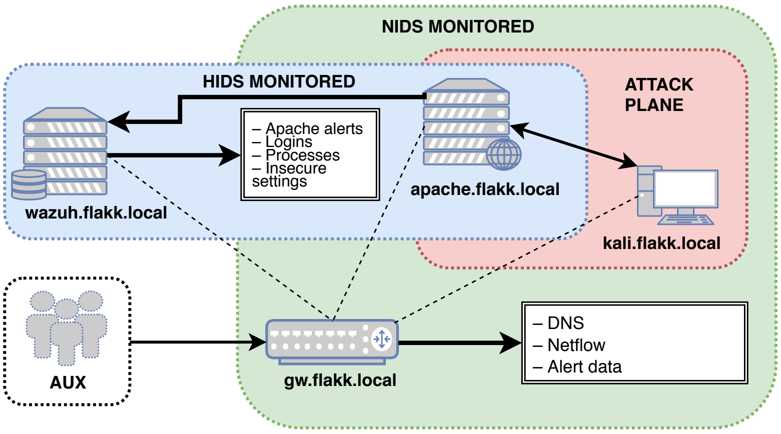


Figure 3.2: The topology of the test-bed used for generating NSM data. The attacker sits on an external network and interacts with the web server. A NIDS is installed on the network gateway, and the web server runs a HIDS agent. Both export NSM data to an ElasticSearch server (not shown) for aggregation.

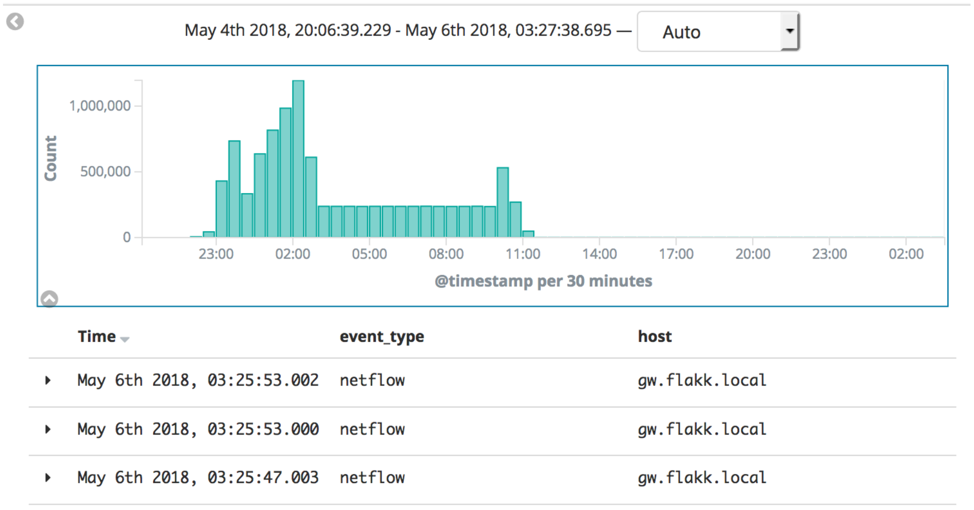
entering or leaving the internal network is monitored by an IDS installed on the gateway.

– An attack scenario is played out: The attackers sits on the external network, scanning for vulnerable services and launching bruteforce attacks on them.

Two sensors were selected and installed into the network to generate NSM data: Suricata and Wazuh. As noted in Section 2.1.2, the former provides not only provides NIDS alert data, but might also be configured to export DNS logs and Netflow. The latter works as a HIDS and performs misuse detection on the host it is installed. Both Suricata and Wazuh support central aggregation of sensor data, sending their data to e.g. ElasticSearch, a popular NoSQL datastore.

The following machines are part of the setup:

– **gw.flakk.local** (192.168.88.1/24, 172.168.88.10/24). This functions as the gateway of the internal network and the ingress from the external network. Suricata was installed on this host in order to provide NIDS alerts, DNS and Netflow data. This host is also connected to the external network, in order to provide transit for the Kali host.


May 4th 2018, 20:06:39.229 - May 6th 2018, 03:27:38.695 —
Auto
Count
1,000,000
500,000
0
23:00
02:00
05:00
08:00
11:00
14:00
17:00
20:00
23:00
02:00
@timestamp per 30 minutes

Time	event_type	host
May 6th 2018, 03:25:53.002	netflow	gw.flakk.local
May 6th 2018, 03:25:53.000	netflow	gw.flakk.local
May 6th 2018, 03:25:47.003	netflow	gw.flakk.local

Figure 3.3: Kibana provides a searchable view into the NSM sensor data continuously aggregated and stored in ElasticSearch. Here we can see Netflow sensor data originating from Suricata, generated for the purpose of data acquirement, the details and results of which are presented in Section 4.1.

- **apache.flakk.local** (192.168.88.90/24). This server runs a simple Wordpress blog via an Apache web server. It also runs a Wazuh agent as an HIDS sensor.
- **elasticsearch.flakk.local** (192.168.88.94). The central storage node for NSM data output by Wazuh and Suricata. It also runs Kibana, which provides a searchable view of the data stored, as seen in Figure 3.2.
- **red.flakk.local** (192.168.88.20). This hosts merely serves as a staging point for the traffic generators, which will be explain shortly.
- **kali.flakk.local** (172.168.88.20/24). This is the attacker in this scenario. Kali Linux comes built-in with tools suitable for doing recognisance, and launching bruteforce attacks and more advanced exploits. It communicates mainly with *gw.flakk.local* in order to reach the web server *apache.flakk.local*.
- **wazuh.flakk.local**. This host serves as the master node in Wazuh, controlling and storing the NSM data from the agents before forwarding it to ElasticSearch.

As the dataset needs to contain both false and true positive alerts, both attack traffic and normal traffic had to be generated. Moreover, the traffic should be somewhat varied and not just a single type of traffic, as noted in Section 2.4.1. Inspired by the concept of β- and α-profiles, this was achieved using two different scripts for generating traffic:

- `simulate_user_traffic.py` creates n workers with spoofed IP addresses, producing a stream of HTTP, SSH and DNS requests according to some distribution. The intensity of each worker is configurable and contributes to a cummulative request intensity of Erlang-n throughout the network. With a selectable probability, the request will contain “tags” (specially crafted user-agents for HTTP or domains for DNS) to trigger misuse rules in Suricata written for this purpose. Each worker will send traffic from an IP uniformly sampled from 192.168.88.128/25 and 172.168.54.0/25 for normal and false positive traffic generation respectively.
- The attacker runs a script called `attack.py`. This implements parts of a kill chain: First, `nmap` is used to scan the target network (192.168.88.0/24). Then, either a SSH or Wordpress login bruteforce attack is launched (by random choice) using `hydra`. It then sleeps for a whole minute. This script was accompanied by manually logging into the web server near the end of the generation era, simulating a successful intrusion.

3.3 Pre-processing and feature extraction

After NSM data has been generated by the test-bed, it is necessary to pre-process it to prepare it for machine learning. During this step, there are three major goals: Firstly, data is collected and transformed as deemed necessary. Then features are then extracted and compiled into a single dataset, and finally the data for classifier training and evaluation created from this.

3.3.1 Feature extraction

A RolX-inspired approach inspired by [ea18] (a discussion is provided in Section 2.4.2) was selected for data fusion and anomaly detection. This research deals with similar types of NSM data, and also provides an unsupervised mechanism for correlating time series of log data. A RolX implementation in Python was provided by Lab41.

Initially, we deal with a list of timestamped NSM data, a series of records with attribute-value pairs of sensor readings. We then wish to apply the fusion process described in Section 3.1 to align and fuse the data, finally arrive at our features. This was achieved using the following four steps steps:

– First, the NIDS alerts are split into time-divided buckets at regular intervals. Each bucket then constitutes a “snapshot” of the network situation at any point in time, i.e. a time series of network characteristics.

– Using Algorithm 3.1, each bucket was mined for hosts and any prior NSM data related to them within a configurable time window. Hence, the NSM data was effectively sampled using a sliding window technique. Note that during this step, attributes of NSM data related to IPs are also normalized: `dest_ip`, `dstip`, `src_ip` etc are all mapped into `ip`, as we only care about host identities.

– Once the situational context was compiled, a *situation graph* G was constructed from each event list using Algorithm 3.2. This graph provides a comprehensive view of all event data associated with a particular NSM alert. We ensure that there exists at most one vertex corresponding to any unique attribute-value pair. Therefore, alerts with overlapping attributes (e.g. the source IP is the same) are linked in the graph. A simplified graph is provided in Figure 3.4.

– Each situation graph was then fed into RolX. This resulted in an $r \times N$ matrix for r roles and N vertices, where r is a parameter of the RolX algorithm equal to the number of structural roles one is trying to discover. The role decomposition of each vertex corresponding to a particular host was then collected and labeled based on whether the host was an attacker or FP generator. The results of applying this step were then compiled into a single dataset.

3.3.2 Pre-processing

After data has been acquired, it was subjected to different kinds of pre-processing to prepare it for machine learning:

Standardization The issue of required statistical properties has previously been noted in Section 2.3.2. To make the data more friendly towards certain kinds of classifiers (e.g. SVM), standardization was applied. For any column, this removes the mean from each row and scales by the unit variance. This yields a data set that is approximately Gaussian with a zero mean and unit norm.

Algorithm 3.1 Algorithm for mining alert data A_{alerts} for candidate hosts and their respective NSM alert data A_{nsm} within the last N minutes.

```
function FIND_CANDIDATES(NSM alert data $A_{alerts}$, alert window $N$)
    $C \leftarrow \{\}$                    ▷ Initialize an empty set of (unique) candidates
    for $a \in A_{alerts}$ do                          ▷ Mine candidates from alert data
        $append(C, a_{srcIp})$
        $append(C, a_{dstIp})$
    end for
    $E \leftarrow \{\}$                    ▷ Initialize a map from candidates to event lists
    for $c \in C$ do          ▷ Get last $N$ minutes of NSM data matching candidate
        $E[c] \leftarrow \{a | a \in A_{nsm}, c_{timestamp} - a_{timestamp} \leq N, a_{srcIp} = c \vee a_{dstIp} = c\}$
    end for
    return $E$
end function
```

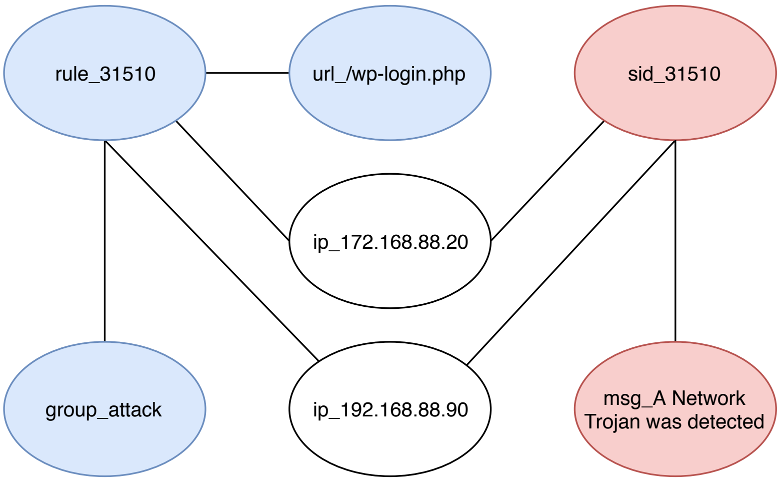


Figure 3.4: A simplified situation graph. Blue vertices represent alerts triggered by Wazuh, whereas red alerts stem from Suricata. Both alerts have the IP pairs 192.168.88.90 and 172.168.88.20 in common, and thus form a network.

Algorithm 3.2 Algorithm for constructing the situation graph

```
function MAKE_GRAPH(event list $E$)
    $S \leftarrow \{a_i | a_i \in E[i], i \in 0 \dots |E| - 1\}$   ▷ Set of unique attribute-value pairs in $E$
    $G \leftarrow (V, E)$, $|V| = |S|$   ▷ Initialize an undirected graph
    $H \leftarrow \{\}$
    $K \leftarrow \{\}$
    for $a_i \in S, i \in 0 \dots |S| - 1$ do
        $H[a_i] \leftarrow i$   ▷ Map attributes to vertices
        $K[i] \leftarrow a_i$   ▷ Map vertices to attributes
    end for
    for $e \in E$ do   ▷ Add events to graph
        for $a_i \in e, i \in 0 \dots |e| - 1$ do
            $h \leftarrow H(a_i)$
            for $a*_j \in e, j \in i + 1 \dots |e| - 1$ do   ▷ Link attributes from same event
                $h* \leftarrow H(a*_j)$
                Add an edge between $V_h$ and $V_{h*}$ in $G$.
            end for
        end for
    end for
    return $G, K$
end function
```

Rebalancing The bucket approach introduces an inherent low base rate of false alerts. The reason is simple: First, note that there are many IPs producing NIDS alerts, with only a few (or a single) of them generating true alerts. However, the buckets also aggregate alerts by IP. Hence, the number of false positive alerts considered in the dataset will be quite small. Moreover, the rate of true intrusion traffic is generally quite small compared to normal traffic, as suggested in Section 2.4.1 and 2.3.1.

Therefore, the learning set will invariably suffer from unbalanced classes. This harms classifier performance, as noted in Section 2.3.1: For example, if the number of true positives is much smaller than the number of false positives, a perfectly performant classifier might be constructed simply by classifying all alerts as false positives. Therefore, rebalancing might be applied to improve the performance. Two techniques (also introduced in 2.3.1) were considered:

- Over-sampling of true positives and undersampling of false positives: Another sampling was performed using only true positive alerts. Furthermore, a uniformly sampled subset of false positives of the same size were created, and the results appended together. This yields a balanced dataset.

- Undersampling and synthetic over-sampling: The implementation of SMOTE included in `imbalanced-learn` was applied to synthesize true positives examples from existing entries.

In the former case, the rebalancing operation changed the composition of the whole dataset. The latter approach was only to the learning set, in order to avoid introducing self-bias when evaluating the classifiers.

Splitting In order to avoid overfitting and evaluate the performance of the classifiers, the original dataset was uniformaly sampled (without replacement) into two partitions for learning and evaluation, composed respectively of 2/3 and 1/3 of the original data rows. Hence, evaluation is only performed using previously unseen data, in accordance with the methodological concerns discussed in Section 2.3.1.

3.4 Classification

It is now time to perform Threat Refinement, as presented in Section 3.1. Due to the NFL theorem (see Section 2.3.1), it is necessary to consider multiple classifiers when doing machine learning research. Three common classifiers readily available in `scikit` were therefore selected:

- RandomForest (`sklearn.ensemble.RandomForestClassifier`). Hyperparameter tuning was applied using a k-fold cross-validation approach as described in Section 2.3.1.
- SVM (`sklearn.svm.SVC`)
- MLP (`from sklearn.neural_network.MLPClassifier`), a type of ANN.

The classifier were trained on each dataset generated by the different rebalancing techniques discussed in Section 3.3.2. In accordance with methodological concerns outlined in Section 2.3.1, training was performed using data not part of set used for evaluating the data, in order to avoid overfitting.

3.5 Evaluation

It was decided to use the *recall* of the negative ("0", here false positive alerts) class as the main indicator of success, with the recall of the positive ("1") class as a secondary measure. Thus, we were able to satisfy the research goal in Research Question 2 (see Section 4.4) of mainly detecting false positive alerts and also retaining *some* true positives. However, it is necessary to account for what *some* true positives mean. We propose a metric as follows:

$$s(C) = (1 - \beta e^{-R_c(1)})R_c(0) \tag{3.1}$$

where $R_C(\phi)$ denotes the recall of the class ϕ using classifier C, and β denotes the importance of recalling true positive alerts. This mirrors the tradeoff between TPR and FPR discussed in Section 2.3.1: By reducing the value of β, one is able to decrease the number of false positives, at the expense of misclassifying some true positives. Conversely, a larger value of β is required if the risk of not detecting some intrusions is unacceptable. For the purpose of this research, it was assumed that the number of false positives was so great that it made NIDS alerts hard to use at all. Hence, β might be set to a low value.

Evaluation was performed by taking the trained classifiers and having them classify unseen data, i.e. the test set (see the discussion of splitting in Section 3.3.2). `sklearn.metrics.classification_report` was then used to calculate the resulting recall for each label (0 and 1), as well as its support (the number of members of each class), which provides an overview of the overall distribution of the classes in the dataset used for evaluation.

3.6 Discussion

We will now provide a short discussion of whether the methods now described will support us in reaching the goals of the research, as stated in Section 1.1. Allowing for paraphrasing, we seek (R1) a useful representation (features) for describing the situational (i.e. situational awareness) of a monitored network, and (R2) using such a representation to develop performant classifiers to remove false positive alerts in such networks, while still retaining some true positive alerts from such network. We then consider the following requirements:

– *a representation for describing the situation* - Our methodology suggests using structural role extraction on graphs of of NSM data, inspired by its application in [ea18].

– *a useful representation* - It is proposed that binary classifiers might use the features for a designed goal. In our case, being *useful* is inevitably linked with the performance of the classifiers utilizing such data, which will be considered in Chapter 4.

– *a monitored network* - We have proposed a simple test-bed that a) runs a virtualized, real-world network using OTS software, which is b) monitored by NSM sensors (HIDS, NIDS etc).

– *performant classifiers* - In Section 3.4, we have selected three classifiers for evaluation. Moreover, in Section 3.5, what *performance* entails in this scenario has been discussed, and a metric (Equation 3.1) proposed in on order to deal with varying preferences of *retaining some true positive alerts*. Moreover, we have also considered the methodological issues of training classifiers, and in particular suggested a separate dataset for evaluation purposes in order to avoid the common issue of overfitting.

As a virtual environment with traffic generation was used for this research, it is possible that the quality of the data is insufficient to conclude with applicability to other networks than the one used for data generation purposes. Considering the criteria put forth in Section 2.4.1, a particular concern is the lack of diversity in the traffic. More importantly, there is little diversity in the attacks performed. A real network would likely have more protocols represented, with different distributions and payloads than those offered by the generator. However, this concern is common and still remains even given a more thorough data generation routine: Due to NFL theorem (see Section 2.3.1), it is not given that any findings resulting from this data are necessarily applicable to real-world networks. Moreover, by generating traffic, the approach selected introduces the risk of implementation errors and issues, such as logical or periodic faults in the scripts responsible for generating traffic.

The feature extraction approach selected is also not without its issues. In “real” role dynamics (see Section 2.4.2), a Bayesian normalcy model for expected roles is constructed using role decomposition time series of the network in its “normal” state. Moreover, [ea18] applies this to all attribute-pairs belonging to the graph of NSM data, not only the vertices corresponding to host identities (IPs). Then, for a given treshold, role transitions that greatly different from the normalcy model indicate anomalies. In our case, it is assumes that the roles themselves encode important information that might be used for anomaly detection, i.e. that intrusions will generally have different structural properties in the graph than normal traffic. It is admitted that this might not necessarily be the case.

Chapter 4
Findings and Discussion

We will now present the results from the research performed in accordance with the methdology presented in Chapter 3 and summarized in Figure 3.1. First, we will present the data acquirement results. We will then consider the results of performing pre-processing and feature extraction. Then, the results of applying the selected classifiers will be shown. Finally, the overall results themselves will be discussed, as well as their implications.

4.1 Data generation

A virtual network environment was constructed in accordance with Section 3.2 using Linux guests, using KVM as the hypervisor. For the attacker, Kali Linux was selected as the operating system, otherwise Debian Linux was used. The following software was used to construct the environment:

– Debian 4.9.65 (Linux)

– Kali Linux 2018.1 (used for the attacker)

– ElasticSearch/Kibana/LogStash 6.2.1

– Suricata 3.2.1

– Wazuh 3.2.0

– WordPress 4.0.23

– nmap 7.00

– wpscan 2.6

– hydra 7.6

The different Linux hosts were connected together using native Linux bridges to form the topology shown Figure 3.2, as provided via the KVM hypervisor. This provides a virtualized network in which to perform generation. Moreover, each host was installed with software and configured as described by Section 3.2. For generation purposes, the scripts for generating true or false positive alerts were installed on `kali.flakk.local` and `gw.flakk.local` with the following parameters:

– **simulate_user_traffic.py**: Traffic distributions of HTTP, SSH and DNS were set to 20, 15 and 65 % respectively. $n = 40$ workers were started, generating false positive alerts with a probability of $p = 1/2$. Source IPs were uniformly sampled from 192.168.88.128/25 and 172.168.54.0/25 in order to simulate a busy network with both internal and external traffic. The source code this script is provided in Appendix A.2.

– **attack.py**: Sleep time between turns was set to $t = 60$ seconds. The probability for selecting either a SSH or WordPress bruteforce attack was set to $p = 1/2$. Although not a parameter of the script, the source IP of this traffic is 172.168.88.20, the IP of the attacker running Kali. The source code this script is provided in Appendix A.1.

Additionally, our sensors were configured as described in Section 3.2: Suricata (NIDS was installed on `gw.flakk.local`, providing misuse-based NIDS alert data, as well as sensor data for DNS and Netflow. In addition, Suricata was configured to create alerts based on the false positive "tags" used by `simulate_user_traffic`, i.e. a special user-agent and domain names for HTTP and DNS respectively. Wazuh was installed as an agent on `apache.flakk.local`, providing misuse-based HIDS data.

Data generation was then performed for 12 hours in total, leading to the sensors generating data. ElasticSearch was used throughout to collect and store the generated sensor data throughout. An example of sensor data generated by Suricata is provided in Listing 4.1. Throughout this period, the false positive generators were always running, whereas the attack script was primarily fired during the first four and last two hours of the generation time. This was due to an unintended crash in the software. However, it does not significantly affect the results. Moreover, it makes the base rate of intrusion traffic even lower, ensuring we are not generating artifically large amounts of real intrusion traffic.

Listing 4.1: Example of Suricata intrusion data, stored as JSON in ElasticSearch. Here we see a warning for suspicious DNS traffic.

```
{
    "src_ip": "192.168.88.1",
    "src_port": 53,
    "geoip": {},
    "application": "suricata",
    "tags": [
      "_geoip_lookup_failure"
    ],
    "host": "gw.flakk.local",
    "tx_id": 1,
    "event_type": "alert",
    "dest_port": 51641,
    "@timestamp": "2018-05-05T22:29:59.773Z",
    "in_iface": "ens2",
    "flow_id": 134566432126805,
    "proto": "UDP",
    "source": "/data/suricata/eve.json",
    "dest_ip": "192.168.88.90",
    "alert": {
      "rev": 1,
      "signature": "SURICATA DNS Unsolicited response",
      "category": "",
      "action": "allowed",
      "gid": 1,
      "severity": 3,
      "signature_id": 2240001
    }
}
```

It was noticed after generation that the DNS setup for `simulate_user_traffic.py` was faulty: Instead of creating labels from a distribution estimating the term frequency in English, a pseudorandom approach was used by mistake. Hence, DNS provides little or no real information about the behavior in the network, and was removed from the fusion pipeline as an NSM data source in the later steps.

4.2 Feature extraction

The following software were used during this step:

- Debian 4.9.65 (Linux) and macOS 10.13.4
- python-dateutil 2.6.0
- elasticsearch-py 6.2.0
- scikit-learn 0.19.1
- imbalanced-learn 0.3.3
- numpy 1.13.0
- pandas 0.21.0
- igraph 0.7.1 (Python/C++)

The process described in Section 3.3 was applied: First, all Suricata alert data were downloaded and aggregated from ElasticSearch as JSON files via its HTTP API. They were then sorted by their timestamp and split into buckets of 45 seconds duration each ($n = 850$). Then, for each bucket, alerts were aggregated into candidates using Algorithm 3.1. In turn, each candidate and its event list was turned into a situation graph using Algorithm 3.2. RolX was then applied to each bucket (with $r = 2$) and the resulting G_n role vector collected for each candidate. Finally, each node vector was labeled depending on the original alert being a false or true positive alert. This is in our case determined by the IP ranges set for true or false positive traffic generation in the scripts configured in Section 4.1), the networks 172.168.54.0/25 and 192.168.88.128/25 respectively.

In the end, $n = 37638$ rows of data were generated during this stage.

An example of the resulting data is provided below:

```
role1,role2,label
```

```
0.8088602254249131,0.11790503264412397,0
0.8662860800911023,0.29786666681631135,1
0.9314385843331936,0.00024310004476482954,0
0.91791797638251,0.09241440241407804,0
```

As can be seen, each row consists of two scalars implicating the degree of membership within one of the two ($r = 2$) roles, as well as a label stating whether or not the host was generating normal or intrusion traffic.

4.2.1 Pre-processing

After feature extraction, the generated data was subjected to further pre-processing, as described in Section 3.3.2. Standardization was applied using `scikit`'s StandardScaler to the final dataset in order to satisfy statistical properties required for e.g. SVM. Rebalancing was applied to avoid overfitting the classifiers during learning. Using the two approaches presented in Section 3.3.2, this resulted in three distinct datasets. Let L be the learning set and T be the test set. Then this resulted in these datasets:

- **data_raw**, where no rebalancing has been applied (i.e. original dataset), with $|L| = 25216$ (99 FPs, 25117 TPs) and $|T| = 12421$ (61 FPs, 12360 TPs).
- **data_smote** for rebalancing with SMOTE, with $|L| = 50234$ (25117 FPs, 25117 TPs) and $|T| = 12421$ (61 FPs, 12360 TPs).
- **data_over_undersampled** for rebalancing with under- and oversampling as described in Section 3.3.2, with $|L| = 7534$ (3767 FPs, 3767 TPs) and $|T| = 15600$ (1830 FPs, 13770 TPs).

4.3 Classifier results

As discussed in Section 3.5, the *recall* of false positives, or the ability to correctly classify FP examples as FPs, is a good measure for evaluating the success of the classifiers given our research goal. Moreover, we suggested a hybrid scoring technique in Equation 3.1 to account for the relative importance of retaining true positives. In the latter case, it was at the same time assumed that losing some true positives was highly tolerable, and the sensitivity was therefore set to $\beta = 0.75$. These considerations form the basis of our evaluation.

The classifiers ANN, RandomForest and SVM previously selected in Section 3.4 were evaluated separately using each the datasets generated by the previous step. First, they predicted the class for all examples in each test set. Then, the recall and support (the number of members of a class, e.g. number of false positives) were calculated using `sklearn.metrics.classification_report`, and then finally the $s(C)$ metric from Equation 3.1 from the recall of true and false positives. The results of applying these operations to the three datasets from the previous section can be seen in found in Table 4.1, Table 4.3, and Table 4.2 respectively.

We note that all classifiers overall perform poorly as general classifiers on unbalanced data, as seen in Table 4.2. However, SVM and RandomForest perform quite well if only the recall of false positive alerts is considered. Of course, this should not be done lightly: Given the low base rate of true positives, simply labeling all alerts as positive should yield a good recall of false positives.

Moving on to the rebalanced datasets, we notice that the dataset rebalanced with SMOTE (Table 4.1 leads to better performing classifiers overall in the case of recalling both false and true positive alerts. However, given the low degree of support (number of test cases), it should not be considered a decisive result: The SMOTE dataset has a test set with a lower amount of false positive alerts. This is realistic given the low base rate of intrusions in real traffic, but still leaves the possibility of just being lucky. Conversely, the over-undersampled dataset (Table 4.3 has a much smaller training set, which might explain the overfitting observed in the final trained classifiers: The classifiers merely recall about 62 % of all false positives.

Overall, the SMOTE based dataset leads to well-performing classifiers, with good recalls for both false and true positive alerts. Moreover, it does this with a training set that mimicks the low base rate of real intrusion traffic. We therefore consider it a "winner" and discuss it further: The best performing classifier overall given our metric $s(C)$ is RandomForest, with excellent recall of false positives (97 %), at some expense at the recall of true positive alerts (85 % vs 93 % and 93 % for ANN and SVM respectively). ROC curves for SVM, RandomForest and ANN are provided in Figure 4.3, Figure 4.3 and Figure 4.3 respectively. As expected, the FPR increases asymptotically with an increasing TPR.

4.4 Discussion

We first consider our original research goals from Section 1.1:

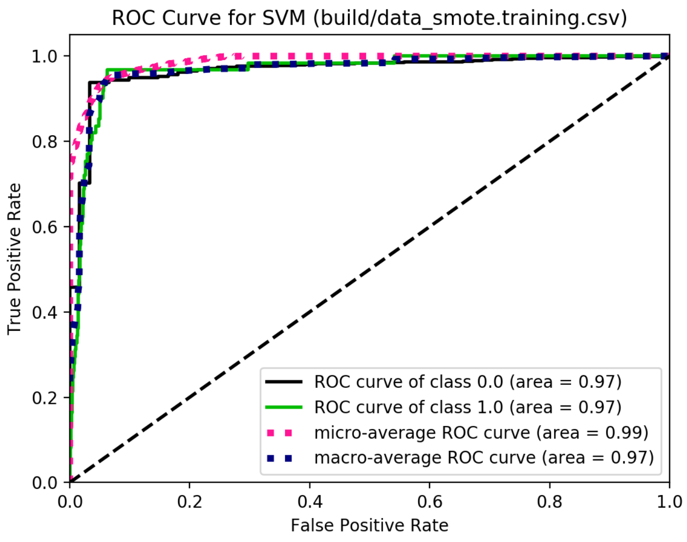


Figure 4.1: ROC Curve for SVM (with SMOTE rebalanced data)

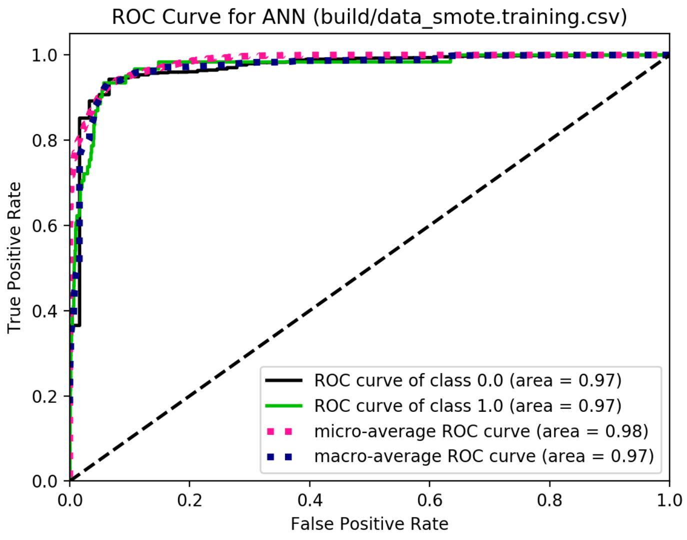


Figure 4.2: ROC Curve for ann (with SMOTE rebalanced data)

Table 4.1: Result table for *data_smote.training.csv*

	ANN		RandomForest		SVM	
	0	1	0	1	0	1
recall	0.93	0.93	0.97	0.85	0.94	0.93
support	12360.0	61.0	12360.0	61.0	12360.0	61.0
$s(C)$ ($\beta = 0.75$)	0.65		0.65		0.66	

Table 4.2: Result table for *data_raw.training.csv*

	ANN		RandomForest		SVM	
	0	1	0	1	0	1
recall	1.0	0.0	1.0	0.16	1.0	0.05
support	12360.0	61.0	12360.0	61.0	12360.0	61.0
$s(C)$ ($\beta = 0.75$)	0.25		0.36		0.28	

Table 4.3: Result table for *data_over_undersampled.training.csv*

	ANN		RandomForest		SVM	
	0	1	0	1	0	1
recall	0.62	0.96	0.62	0.91	0.63	0.96
support	13770.0	1830.0	13770.0	1830.0	13770.0	1830.0
$s(C)$ ($\beta = 0.75$)	0.44		0.43		0.45	

R1 What is a suitable feature representation of NIDS alert data with auxiliary sensors and their contribution to operational awareness in computer networks?

R2 Is it possible to construct classifiers (based on such a representation) that are able to detect primary false positives alert data and ideally retain true positives given auxiliary sensor data?

Having already suggested the selection of methods is appropriate for reaching these goals in Section 3.6, we now consider whether or not this is the case:

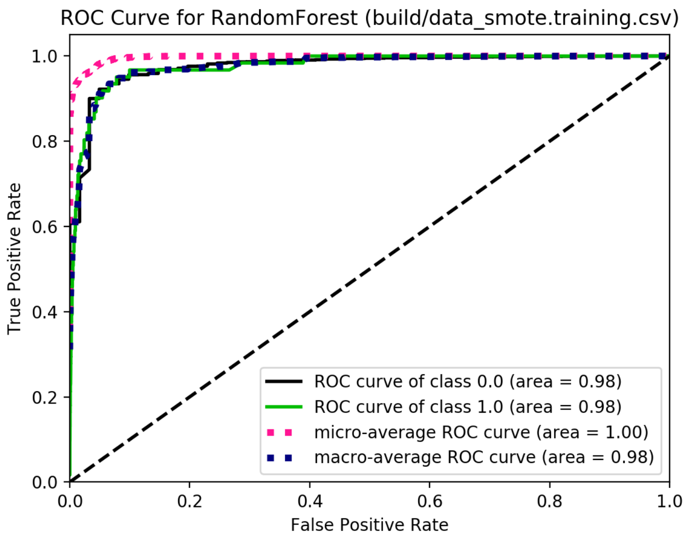


Figure 4.3: ROC Curve for RandomForest (with SMOTE rebalanced data)

Somewhat counter-intuitively, we consider the second goal (R2) first. Assuming the sake of argument that R1 is already true, we consider the results of constructing classifiers using this representation (i.e. after fusion). As seen in the classifier results presented in Section 4.3, we are in fact able to filter out a great deal of false positive alerts while retaining some false positives. Our hybrid metric (Equation 3.1) captures this trade-off in filtering, and RandomForest delivers the best result given our slight preference for many fewer false positives in exchange for a few lost true positives. For a visual representation of this trade-off, one might consider the ROC curves of each classifier (using the SMOTE rebalanced dataset from Section 4.1), as is demonstrated in Figure 4.3, Figure 4.3 and Figure 4.3: As the recall of true positives increase, so does the amount of false positives ("noise"). Moreover, the metric selected is adaptable to other preferences, such as the contrary of ours, adapting to the highly subjective consideration of *some* alerts. Hence, we have answered R2 and demonstrated the viability of constructing such classifiers.

With these observations in mind, we might consider the first goal (R1): The first goal can be said to be achieved primarily through our fusion step. By joining data into a graph connected by overlapping attributes, we perform correlation and give an overview of the current situation of the network, hence the term *situation graph*.

Moreover, it is suggested that the structural role extraction performed using RolX is *suitable*, given the good results presented in Section 4.3. Hence, for all practical motivations underlying this research question (as presented in Chapter 1), we have demonstrated a sufficient answer to R1.

However, it would of course be an error to not considered factors limiting the implications of these results. One such factor that must be considered is the nature of the dataset. As noted in Section 3.6, the network and methods used for generating the datasets leads to traffic and therefore NSM sensor data that is not very diverse, neither in types of traffic, the variety of attacks or even the network topology. For example, there is no Network Address Translation (NAT) mechanism in our network, although this would be common in a corporate IPv4 network. Hence, these results might not immediately be applicable to other types of traffic with different properties in the distribution of protocols, volumes, payloads and so forth. This fact alone suggests more research is needed in applying the techniques proposed by this thesis to more diverse datasets.

Another factor that suggests some moderation in interpreting these results is the feature extraction approach itself, as suggested by the earlier concerns voiced in Section 3.6. Beneath the approach considered in this thesis, there is an implicit belief that the structural roles extracted from the situation graph (i.e. the feature extraction used in this thesis) by themselves encode enough information to discriminate between normal and intrusive traffic. While it might be suggested that such traffic might exhibit *different* structural properties, it by no means follows that these properties themselves can be generalized. For example, networks might periodically drastically change their topological and traffic characteristics during certain seasons, like when new hires arrive, large projects are initiated, or new network services or configurations are introduced into the network. In this case, a Bayesian model similar to [ea18] might better capture the dynamics of the network, in that discovers the *transitions* in roles over time, in a sense mirroring the concept of the phases constitute a kill-chain, as presented in Section 2.1. Hence, more research is needed to compare the efficacy of these techniques. Moreover, it is suggested that more diverse traffic (as previously considered) might help provide a good ground for performing such comparisons.

These reservations notwithstanding, both research questions have been answered. Namely, we have demonstrated a (R1) feature representation that combines a superset of IDS data with NIDS alerts to convey the situational characteristics of the network at different points in time, and successfully applied it to (R2) construct binary classifiers that perform well when the goal is to remove false positive alerts and retain some number of true positive alerts, depending on the preference of the user. As such, these findings show that we have reached what the research set out to do, and provide motivation for further investigating the efficacy of the techniques herein to more diverse datasets.

Chapter 5 Conclusion and future work

This thesis has investigated whether combining data from multiple NSM sensors, including NIDSs and HIDSs, can be combined to reduce the amount of false positive alerts from NIDSs. We have demonstrated a way to combine ("fuse") NSM alerts and data from other sensors as a graph in order to express the situational context of alerts. Moreover, a virtual test-bed for generating NSM data was constructed in order to acquire data. We then demonstrated how a combination of a graph-based feature extraction algorithm (RolX) and binary classifiers applied to this data succeeds in recalling false positive alerts, while still retaining some portion of the true positive alerts. Particularly, the best approach (using a RandomForest classifier) shows that we recall 97 % of all false positive and 85 % of all true positive alerts.

This work is a contribution towards providing more accurate and relevante alerts in NSM. The results of the aforementioned techniques are promising, and warrant further investigation into the applicability of machine learning for the purpose of reducing false positives. In time, it is hoped that this will allow Computer Emergency Response Teams (CERTs) to allocate their resources more efficiently and also discover more attacks on their networks.

5.1 Future work

As this work has been performed in a virtualized test-bed, it remains to be seen how the approach will perform when applied to real networks. Future work should look into applying a similar approach to a NSM data from production networks. Alternatively, an approach using the β- and α-profiles presented in Secton 2.4.1 would be a great boon to constructing future test-beds, allowing for reuse, flexibility and easier verification of results.

The use of RolX in this research was heavily inspired by prior art by [ea18]. There, a more sophisticated time-series oriented approach known as role dynamics was applied. For this research, a simplified approach was selected instead. Future work should therefore investigate whether or not role dynamics will yield a better result.

Similarly, results in Chapter 4 suggest that graph-based features might prove useful for reducing false positives or even detect intrusions in network based on NSM data. There might exist other novel graph-based feature extraction techniques that work as well or even better for these purposes, or that might be developed. It is therefore suggested to further study the applicability of graph features to NSM data.

Additionally, the list of sensors and statistics used for data generation used for this research are by no means exhaustive. Without going too far in-depth, CERTs will often use measures such as the entropy of domain names and their popularity in global rankings to detect C2 traffic in particular. Therefore, more work is needed in generating datasets that include such and other types of sensor data.

Finally, the goals of the techniques are complementary to developing new hybrid IDS solutions: Instead of filtering away false positives, such techniques might instead be applied to detect intrusions. There is prior art in [ea18] for this application, but results might be different or better by additional sensors, graph-based anomaly detection techniques and any improvements provided by the above research suggestions.

References

[And16] Lars Christian Andersen. Data-driven approach to information sharing using data fusion and machine learning. Master's thesis, 2016.

[ASØ12] Christian Emil Askeland, Anders Emil Salvesen, and Arne Fjæren Østvold. Senatus - implementation and performance evaluation. Master's thesis, Department of Telematics, NTNU, 2012.

[Axe99] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 1–7. ACM, 1999.

[Bas00] Tim Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM*, 43(4):99–105, 2000.

[Bej13] Richard Bejtlich. *The practice of network security monitoring: understanding incident detection and response.* No Starch Press, 2013.

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[Den87] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987.

[DRF$^+$11] Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann. On botnets that use DNS for command and control. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 9–16. IEEE, 2011.

[ea18] Palladino et al. Anomaly detection in cyber networks using graph-node role-dynamics and netflow bayesian normalcy modeling. FloCon, 2018.

[Eng10] Vegard Engen. *Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data.* PhD thesis, Bournemouth University, 2010.

[Ger09] R. Gerhards. The syslog protocol. RFC 5424, RFC Editor, March 2009. http://www.rfc-editor.org/rfc/rfc5424.txt.

[GSLG16] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. An evaluation framework for intrusion detection dataset. In *Information Science and Security (ICISS), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[Hal17] Christoffer V. Hallstensen. Multisensor fusion for intrusion detection and situational awareness. Master's thesis, Department of Information Security and Communication Technology, NTNU, 2017.

[HCA11] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[HČT$^+$14] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.

[HDO$^+$98] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[HGER$^+$12] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2012.

[HHS$^+$10] Blaise Hanczar, Jianping Hua, Chao Sima, John Weinstein, Michael Bittner, and Edward R Dougherty. Small-sample precision of roc-related estimates. *Bioinformatics*, 26(6):822–830, 2010.

[HL97] David L. Hall and James Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[JD02] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375. ACM, 2002.

[JDM00] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[JNVDÅ17] Kristoffer Jensen, Hai Thanh Nguyen, Thanh Van Do, and André Årnes. A big data analytics approach to combat telecommunication vulnerabilities. *Cluster Computing*, 20(3):2363–2374, 2017.

[KM17] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017.

[Man18] Mandiant. M-trends 2018. https://www.fireeye.com/current-threats/annual-threat-report/mtrends.html, 2018. Last accessed June 17th 2018 12:25.

[NFP12] Hai Thanh Nguyen, Katrin Franke, and Slobodan Petrovic. Feature extraction methods for intrusion detection systems. *Threats, Countermeasures, and Advances in Applied Information Security*, 3:23–52, 2012.

[Pet17a] Slobodan Petrovic. IMT4204 — IDS/IPS Definition and Classification (university lecture). 2017.

[Pet17b] Slobodan Petrovic. IMT4204 — Testing IDS (university lecture). 2017.

[Pie04] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 102–124. Springer, 2004.

[Pon17] L Ponemon. Cost of data breach study: Global analysis. *Poneomon Institute sponsored by IBM*, 2017.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[RGNH12] Ryan Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Role-dynamics: fast mining of large dynamic networks. In *Proceedings of the 21st International Conference on World Wide Web*, pages 997–1006. ACM, 2012.

[RGNH13] Ryan A. Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM, 2013.

[SSTG12] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[TBLG09] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[VT$^+$97] Paul Vixie, , Susan Thomson, Yakov Rekhter, and Jim Bound. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, RFC Editor, April 1997. http://www.rfc-editor.org/rfc/rfc2136.txt.

[WFHP16] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[Wol96] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[XBSY13] Kui Xu, Patrick Butler, Sudip Saha, and Danfeng Yao. DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing*, 10(3):143–153, 2013.

[ZOM07] Kelly H Zou, A. James O'Malley, and Laura Mauri. Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models. *Circulation*, 115(5):654–657, 2007.

[ZXXW15] Guodong Zhao, Ke Xu, Lei Xu, and Bo Wu. Detecting apt malware infections based on malicious DNS and traffic analysis. *IEEE Access*, 3:1132–1142, 2015.

Chapter A
Code

A.1 attack.py

```
#!/usr/bin/env python2.7

import time
import random
import logging
import os
from subprocess import call, Popen

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.DEBUG)

DEVNULL = open(os.devnull, 'wb')
TARGET = "192.168.88.90"

# Hydra template for WordPress bruteforce
ATTACK_STR = "&".join([
    "/wp-login.php:log=^USER^",
    "pwd=^PASS^",
    "wp-submit=Log In",
    "testcookie=1:S=Location"
])

while True:
    call(["nmap", "-O", "192.168.88.0/24"], stdout=DEVNULL, stderr=DEVNULL)
    # First, do some recognisance
    # Then either SSH or Wordpress bruteforcing
    if random.random() < 0.5:
        logger.debug("Doing SSH bruteforce")
        cmd = ["hydra", "-l", "root", "-P", "/usr/share/wordlists/rockyou.txt",
               "192.168.88.90", "ssh"]
    else:
        logger.debug("Doing Wordpress bruteforce")
        cmd = ["hydra", "-l admin", "-P", "/usr/share/wordlists/rockyou.txt",
               "192.168.88.90", "-V", "http-form-post", ATTACK_STR]
    p = Popen(cmd, shell=True)
    # Timeout after 10 minutes
    time.sleep(60 * 10)
    p.terminate()
    # Sleep for 15 minutes
    time.sleep(60 * 15)
```

A.2 simulate_user_traffic.py

```
#!/usr/bin/env python3

# ----- GENERAL DESIGN NOTES

# This script generates SSH, DNS and HTTP traffic towards designated targets.
# The traffic is distributed over an Erlang-N distribution with N workers, with
# Exp(1/l) holding times and traffic types distributed ~ N.
#
# All traffic types can be sorted into either false positives or true
# negatives. In the case of false positives, some deviating behavior is
# triggered in order to (possibly) trigger NSM alerts:
#
# - DNS: .xyz (blackholed) is generally considered suspicious, and FP DNS
#   requests will requests domains within this TLD. For false positives, a
#   random bytestring (similar to base64 encoded C2 payloads) is created.
#
# - SSH: FP will try to (unsuccessfully) log into the root user (possibly
#   triggering an HIDS alert)
```

```
#
# - HTTP: FP will try to post to the login page with a suspicious user agent
#   (triggering a Suricata rule written for this purpose).

# Requires the curl (libcurl), ssh (openssh) and dig (bind-tools) commands.

import binascii
import argparse
import random
import ipaddress
import time
# import scapy.all as scapy
import logging
import os
import multiprocessing

from subprocess import call, DEVNULL

HTTP_TARGET="192.168.88.1"
DNS_TARGET="192.168.88.1"
SSH_TARGET="192.168.88.90"
REQUESTS_PER_WORKER_SECOND = 1
FALSE_POSITIVE_RATE = 0.5
SIMULATION_TIME = 10e6
WORKERS = 40

# In order to use this, do:
#   ip link add inject0 type dummy
#   for i in $(seq 1 128); do
#     ip -4 addr add "172.168.54.${i}/24" dev inject0
#   done
#   for i in $(seq 128 250); do
#     ip -4 addr add "192.168.88.${i}/25" dev inject0
#   done
#   ip -4 route add 172.168.54.0/24 via 192.168.88.20
#   for i in $(seq 1 128); do
#     ip -4 addr del "172.168.54.${i}/24" dev inject0
#   done
#   for i in $(seq 128 250); do
#     ip -4 addr del "192.168.88.${i}/24" dev inject0
#   done
HOME_NET = "192.168.88.128/25"
EXTERNAL_NET = "172.168.54.0/25"

logger = logging.getLogger(__name__)

# EVIL_IPS
# NICE_IPS
def random_ip_address(false_positive=False):
    net = EXTERNAL_NET if random.random() < 0.5 else HOME_NET
    addresses = list(ipaddress.ip_network(net))
    return random.choice(addresses)

# cat local.rules  | grep "DNS Query for Suspicious" \
# | grep -Po "Suspicious .(.+) Domain" \
# | sed -rn 's/Suspicious (.+) Domain/"\1",/p'
BAD_ZONES=(
    ".com.ru",
    ".com.cn",
    ".co.cc",
    ".cz.cc",
    ".co.kr",
    ".co.be",
    ".net.tf",
    ".eu.tf",
    ".int.tf",
    ".edu.tf",
    ".us.tf",
    ".ca.tf",
    ".bg.tf",
    ".ru.tf",
    ".pl.tf",
    ".cz.tf",
    ".de.tf",
    ".at.tf",
    ".ch.tf",
    ".sg.tf",
    ".nl.ai",
    ".xe.cx",
    ".noip.cn",
    ".ch.vu",
    ".gr.com",
)

# Alexa top1m domains
with open("/data/top-1m.csv", "r") as f:
    NICE_ZONES = f.readlines()
logger.debug("Read and parsed Alexa top1m domain list")

def random_domain(false_positive=False):
    if false_positive:
        zone = random.choice(BAD_ZONES)
        label = "{}{}".format(
            binascii.b2a_hex(os.urandom(15)).decode("utf-8"),
            zone
        )
    else:
        label = random.choice(NICE_ZONES)
    return label

def request_sleep(requests_per_second=REQUESTS_PER_WORKER_SECOND):
    "Sample sleep times in order to achieve a mean of 1/'requests_per_second'"
    return random.expovariate(requests_per_second)

def gen_request_type():
    "Sample to achieve the desired distribution of traffic types"
    choice = random.random()
    # 20 % DNS
    if choice < 0.20:
        return "dns"
    # 15 % SSH
    if 0.20 <= choice < 0.35:
        return "ssh"
    # 65 % HTTP
    return "http"

def worker(ip_address):
    name = "{} ({})".format(ip_address, multiprocessing.current_process().name)
    while True:
        false_positive = True if random.random() < FALSE_POSITIVE_RATE else False
        r_type = gen_request_type()
        if r_type == "http":
            gen_http(ip_address, HTTP_TARGET, false_positive)
        if r_type == "dns":
            gen_dns(ip_address, DNS_TARGET, false_positive)
        if r_type == "ssh":
            # gen_ssh(ip_address, SSH_TARGET, false_positive)
        logger.debug("{} sent {} request (fp={})".format(
            name, r_type, false_positive))
        time.sleep(request_sleep())

def gen_ssh(src_ip, target, false_positive=False):
    username = "root" if false_positive else "guest"
    call(["ssh", "-b", str(src_ip), "-o", "PubkeyAuthentication=yes",
          "{}@{}".format(username, target)], stdout=DEVNULL, stderr=DEVNULL)

def gen_dns(src_ip, target, false_positive=False):
    zone = random_domain(false_positive)
    call(["dig", "-b", str(src_ip), "@{}".format(target), zone],
         stdout=DEVNULL, stderr=DEVNULL)

def gen_http(src_ip, target, false_positive=False):
    user_agent = "Mozilla/5.0 (Android 4.4; Mobile; rv:41.0) Gecko/41.0 Firefox/41.
    if false_positive:
        user_agent = "Mozilla/5.0 (FPfox)"
    target = "{}/wp-admin".format(HTTP_TARGET)
    call(["curl", "--interface", str(src_ip), "-H",
          "User-Agent: {}".format(user_agent), target],
          stdout=DEVNULL, stderr=DEVNULL)

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='Simulate user traffic')
    parser.add_argument(
        "--debug-level",
        type=int,
        help="debug level (10 = DEBUG, 20 = INFO, 30 = WARNING)",
        default=logging.WARNING,
        choices=(logging.DEBUG, logging.INFO, logging.WARNING)
    )
    args = parser.parse_args()

    logging.basicConfig(level=args.debug_level)

    ip_addrs = (random_ip_address() for _ in range(WORKERS))
    workers = [multiprocessing.Process(target=worker, args=(addr,))
               for addr in ip_addrs]
    for worker in workers:
        worker.start()
    time.sleep(SIMULATION_TIME)
    for worker in workers:
        worker.terminate()
```