Slow Port Scanning with Bro

Roger Larsen

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College 2013

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Slow Port Scanning with Bro

Roger Larsen

2013/11/27

Abstract

Today's society relies on computer networks. More and more data of vital importance are transmitted over them each day. Because of that, networks have become an interesting target for attackers, from ordinary criminals to foreign organizations and states. This has forced equipment providers and network administrators to make computer networks more robust. To this end, various countermeasures against cyber attacks are performed. One of the most commonly used ones is application of Intrusion Detection Systems (IDS). These systems are capable of classifying network traffic into several categories, according to the traffic features determined in advance. The basic classification performed by them is the classification in two classes – benign traffic and malicious traffic.

The classification methods that IDS implement are different, but classic pattern/signature matching and statistical parametric decision making are used very often. According to the intrusion detection model, IDS are classified into two categories: misuse detection systems and anomaly detection systems. Misuse detection systems use a database of known attacks and report if they recognize signatures of known attacks in the incoming traffic. Anomaly detection systems define profiles of normal host/network behavior and report discrepancies from that.

This thesis concentrates on methods of detection of special kind of reconnaissance activity in computer networks – so-called port scanning, which tries to determine what services are active on a target host. In addition, the scans are considered slow – this means that the time delay between scanning two ports is relatively long – from several minutes to several days. This kind of port scanning is in general harder to detect by IDS. The IDS of particular interest in this context is Bro – an open-source system that detects intrusions by semantic, highly stateful traffic analysis. This system also has advanced protocol detection capabilities. It can be configured to be either misuse or anomaly detection system, even a combination of both at the same time. As such, it has attracted much attention of the scientific community in the recent years. The goal of the thesis is to develop a method for slow port scanning detection with Bro and compare the capabilities of the new method with slow port scanning detection methods applied on other IDS, especially in the presence of noise.

Our results shows that our modified version of `scan.bro` policy script, gave improved slow port scanning detection capabilities in Bro.

Sammendrag (Abstract in Norwegian)

Dagens samfunn har i stor grad gjort seg avhengig av datamaskiner med nettverk og Internet forbindelse. Dette har ført til en stor økning av kriminell aktivitet mot disse datamaskinene, både fra individuelle og godt organiserte kriminelle samt statlig støttede organisasjoner. Produsenter og administratorer av datautstyr må hele tiden oppdatere og passe på at både fysiske enheter og programvare er tilstrekkelig rustet for å stå imot dette stadig økende presset. Det finnes forskjellige løsninger for å kontrollere nettverkstrafikk. Den vanligste løsningen er inntrengingsdeteksjons systemer (IDS). Disse systemene kan klassifisere datatrafikk. Normalt klassifisere datatrafikk i normal trafikk og uønsket trafikk.

Klassifiseringsmetodene IDS bruker er noe forskjellige. Den mest vanligste metoden er å bruke tidligere kjente mønster/signaturer av uønsket datatrafikk. Denne metoden kalles signaturbasert deteksjon. Den andre metoden er såkalt avviksmetoden. Her blir den normale datatrafikken brukt som en basisprofil og trafikk som avviker etter gitte statistiske parameter blir definert som uønsket.

Denne masteroppgaven fokuserer på deteksjon av portskanning. Portskanning er en typisk aktivitet i en tidlig fase av et angrep: rekognoseringsfasen. I denne rekognoseringsfasen er angriperen på jakt etter mulige angrepspunkter og evt. sårbarheter i datanettverket. Hvis angriperen minsker hastigheten mellom hver port som blir forsøkt skannet, gjerne med flere minutter eller kanskje til og med timer, kalles dette for sakte portskanning. Slike sakte portskanninger er vanskeligere å oppdage for inntrengingsdeteksjons systemer. Vi vil i vår masteroppgave fokusere på Bro IDS. Bro er gratis (åpen kildekode) og har kraftige protokollanalysemekanismer samt et omfattende skriptspråk. Bro kan konfigureres til å fungere som både signaturbasert og avviksbasert IDS. Bro har fått mye oppmerksomhet i forskningsmiljøer verden over. Vi har som mål i vår masteroppgave å forbedre Bro sin deteksjon av sakte portskanning, sammenligne andre inntrengingsdeteksjons systemer og vurdere evt. mengden av falske alarmer.

Våre resultater viser at vårt modifiserte `scan.bro` skript forbedret Bro sin egenskap til å detektere sakte portskanning.

Preface

The author is employed by Austevoll Kraftlag SA, LYSGLIMT dep. [1]. LYSGLIMT is the major broadband supplier in Austevoll municipal. LYSGLIMT have delivered triple play services (Internet,IPTV & VoIP) since 2003. The author has over twenty years experience in running and maintaining ICT networks for SMB sector. The author have managed LYSGLIMT's core, distribution and access network for the last eight years. This includes security, monitoring, email and DNS infrastructure. The authors formal background is engineer degree in electronics and computer from Narvik University College (1989).

Acknowledgments

I have been fortune to get much support and feedback during my master thesis. I would like to thank Prof. Dr. Slobodan Petrović for his patient and supporting supervising. Thanks to my fellow students Aud Gran and Ernst Kristian Henningsen for their support and unbiased feedback in our common journey for a master's degree. I have not made this as a remote part time student without you! I will also thank my friend and phd student Gaute B Wangen for guiding me the right way in science methodology. A great thank to Austevoll Kraftlag SA for giving me the chance to study and for their support and understanding.

Finally, but the greatest thanks to my family for their patience during my study.

Contents

List of Figures

List of Tables

Glossary

BDS License	Berkeley Software Distribution License (Regents of the University of California, University of California, Berkeley, 1998)
Cloud Computing	Cloud computing is referring to computing services on Internet. National Institute of Standards and Technology, U.S. (NIDS), have a publication where they define this term more in details [9].
Cloud Services	Services delivered by Cloud computing. Typical services; document storage, applications etc.
GIT	GIT is a distributed revision control and source code management system initially developed by Linus Torvalds
Metadata	blabla
Network TAP	A network TAP is a passive or active network equipment that are able to tap the network traffic without interfering it. It may also be virtual.
Port Scanning	A search for hosts and their open ports/services.
SYSLOG	A standard for collecting system log local or remote on a Linux/BSD/Unix machine. RFC-5424. [10].
Vulnerability Scanning	A search for vulnerabilities/weakness on computer equipment(s).

Table 1: Glossary

Abbreviations

ASCII	American Standard Code for Information Interchange. In plain English; clear text.
API	Application Programming Interface
BSD	Berkeley Software Distribution. Regents of the University of California, University of California
CIDR	Classless Inter-Domain Routing
CLI	Command Line
CVE	Common Vulnerabilities and Exposures
CPU	Central Processing Unit
DoS	Denial of Service
GUC	Gjøvik University College
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IANA	Internet Engineering Task Force
ICT	Information and Communication Technology
IDS	Intrusion Detection Systems
IBR	Internet Background Radiation
IETF	The Internet Engineering Task Force
IS	Information Security
ISO	International Standard Organization
IP	Internet Protocol
IPv4	IP version 4
IPv6	IP version 6
ISP	Internet Service Provider
NAT	Network Address Translation
NIDS	Network Intrusion Detection Systems
NISlab	Norwegian Information Security laboratory
NIST	National Institute of Standards and Technology, U.S.
P2P	Peer to Peer. Defined in RFC-5694.
RFC	Request For Comment. A de-facto standard defined by IETF.
SMTP	Simple Mail Transfer Protocol
SYSLOG	System Log
TCP	Transport Control Protocol
TCP/IP	Transmit Control Protocol / Internet Protocol
VoIP	Voice over Internet Protocol

Table 2: Abbreviations.

1 Introduction

"The Internet is a mirror of the population that uses it!"
– Vinton Cerf

This chapter is the introduction to the thesis. It presents the topics covered by the thesis; problem descriptions, the justification and motivation. We identify our research questions and scope of the thesis. Finally we list the outline of the thesis and summarize our contributions.

1.1 Topics

This thesis is a computer science research that focus on information security. We look closer on Intrusion Detection Systems (IDS). An IDS is capable of categorizing computer traffic into two different classes; (i) benign traffic or (ii) malicious traffic. They may be installed to work in either passive (monitoring) or active (filtering) mode. In information security field the IDS is an important piece of equipment for controlling traffic flow. This is especially important in highly trusted/secured area.

We will focus on Bro IDS [11]. Bro is open source software with powerful network analyzing capabilities. Bro has a powerful script language and good port scanning detection capabilities. Bro has been popular in research groups worldwide since its birth in 1999 [12].

We challenge Bro detection capabilities by using so called slow (stealthy) port scanning.

1.2 Keywords

Information Security, Intrusion Detection System, IDS, Bro, Slow Port Scanning.

1.3 Problem description

Port scanning in computer networks is analogue to window- and door handle rattling in our daily life. People often check if the door and/or window are actually locked by twisting the handle. Any burglar may rattle windows and/or doors in hope for an easy access into a building. In computer networks port scanning activity is the precursor for attacks [13] [14]. We classified port scanning into benign or malicious network activity. Network administrators may use quick port scanning to check if their servers are alive. Applications may use quick port scanning to be able to connect to Internet. Malicious software use port scanning in their eager to infect other networked hosts. Hackers/attackers use port scanning in their reconnaissance phase where they search for open ports / services. The first two are benign, but the last two are malicious port scanning [15–17].

Malicious port scanning has increased dramatically the last years [18–21]. We will in our master thesis strive to improve the detection capabilities in a security system.

Normal port scanning is in general easy to detect by security systems. However, a more stealthy way of port scanning is difficult to detect. This stealthy way of performing port scanning is done by slowing down the port scan interval [22, 23].

1.4 Justification, motivation and benefits

In this section we justify our efforts and describe our motivation and benefits with our research.

Information Security

Today, Internet plays a major role in our life, both personal and at work. We use all kinds of Internet services throughout the day; reading news, checking weather forecast, communicating (email, chat, video conference etc.), order holiday trips, saving our photos and perhaps buy our cars etc. Some companies have even moved all their documents to Internet based services (so called cloud computing) [9].

The famous so-called Internet evangelist, Vinton Cerf [24, 25], commented the human side of Internet like the quote in start of this chapter: *"The Internet is a mirror of the population that uses it!"*

Where there is human activity - there is criminal activity. Internet is the fastest growing arena regarding criminal activity, and the statistics are scary facts. The *2013 Internet Security Threat Report, Volume 18* states the following (not complete); (i) 42% increase in targeted attacks in 2012, (ii) 31% of all targeted attacks aimed at businesses with less than 250 employees, (iii) web-based attacks increased 30%, (vi) Spam volume continued to decrease, with 69% of all email being spam and (v) the number of phishing sites spoofing social networking sites increased 125% [26–29].

Intrusion Detection Systems (IDS)

IDS's are built to analyze computer network traffic and classify their input into (i) benign or (ii) malicious traffic.

An analogue to this kind of systems is control mechanisms on our roads. We accept high flow of all traffic on the big/main roads (cars, lorries, motorbikes etc.). Here we have traffic police that often uses cameras to monitor traffic. However, when we want to access a private company or even a military installation - we need to identify ourselves in an access control post. The first main road is similar to passive (monitoring), but the latter is the active (filtering) operation.

Today, all kind of rapid/normal port scanning is in general easy to detect and intrusion detection systems may easily be able to generate alerts. However, if this port scanning is slowed down regarding time (longer time between ports scanned), it may be more difficult to detect. This kind of port scanning is called slow or stealthy port scanning.

Bro IDS

Bro is an IDS initially developed by Vern Paxon in 1999 [30]. Bro is today developed by The Bro Project. Bro is open source software licensed under BSD Licensing [31]

Bro is a stand-alone system for detecting network intruders in real-time by passively monitoring a network link over which the intruder's traffic transits. Bro can be configured to be both misuse and anomaly based detection system. Bro is much more than an IDS, it is a powerful network analysis framework. Bro has since 1998 been embraced by research groups all around the world [12]. Bro strives to bridge the traditional gap between academia and operations since its inception.

With the use of smart software like IDS, filtering and monitoring computer network traffic, we can most likely detect and block unwanted traffic (attacks). The IDS we focus on in our thesis (Bro) is capable of detecting unwanted traffic. With our improvement in Bro we may go one step further towards a better system in the struggle against cyber criminals.

With this thesis, we try to add reflections, observations and experience, hoping that they are going to be useful for the Bro research community.

1.5 Research Questions

Our main research question: *Can we improve the detection rate regarding slow port scanning in Bro?*. We refer to this main research question in our text with the abbreviation RQ1.

In this thesis we seek answer to the following sub-questions:

RQ1.1 Will we be able to improve the slow port scan detect rate in Bro?

RQ1.2 What is the slow port scan detection rate in Bro?

1.6 Scope

Our focus in this thesis is Bro IDS and slow port scanning detection capabilities.
We will not use live traffic in our background traffic injection.
Performance measure regarding CPU, memory and bandwidth are not included in our work.
Our main focus regarding IP is version 4 (IPv4).

1.7 Thesis outline

We start this thesis with an introduction including problem description, justification, motivation, research questions, scope, outline and our contributions.

In Chapter 2 we describe some important technical details to better understand our thesis.
In Chapter 3 we describe previous work closely related to our topics.
In Chapter 4 we describe our research methods in details.
Chapter 5 is our core work in this thesis. We describe our improvements in Bro.
In Chapter 6 we describe our experimental setup and results.
In Chapter 7 we discuss our results.
In Chapter 8 we conclude our findings and address our research questions.
In Chapter 9 we present some ideas for future work.

1.8 Summary of contributions

Our results regarding slow port scanning detection capabilities with Bro represent a useful contribution to the IDS community in general and Bro user community in particular. We have not found any previous work where Bro is challenged low port scanning

2 TCP/IP, Port scanning and IDS

"The only truly secure system is one that is powered off,
cast in a block of concrete and sealed in a lead-lined
room with armed guards."
– Gene Spafford

In this chapter we explain more background details for better understanding. We cover TCP/IP, port scanning and intrusion detection systems (IDS) in more details. Our research is quite detailed and a thorough background regarding is crucial for better understanding.

2.1 TCP/IP, ports and services

In this section we explain briefly the TCP/IP protocol suite. We take a closer look into the TCP packet header with a special insight into the *flags* field.

2.1.1 TCP/IP background

TCP/IP is a protocol suite where its crucial precursors were initially developed in early 1960. Cerf, V.G. and Kahn, R.E. published in 1974 the famous paper *A Protocol for Packet Network Interconnection* where they describe the *Transport Control Protocol (TCP)* [32]. The last version of the TCP definition is found in a so-called Request For Comment document (RFC). The RFC's are documents created by different expert groups. RFC's may often be used as standards. RFC-793 [33] defines Transmission Control Protocol (TCP) by Internet Engineering Task Force (IETF [34, 35]).

Today, the TCP/IP protocol suite consists of many protocols. Typically well-known protocols that uses TCP as the base engine are; HTTP (TCP port 80), SMTP (TCP port 25) and FTP (TCP port 21). There has been an enormous evolution in communication protocols since this early stage. Still, there are several protocols that are rather unchanged in many decades that continue to produce online services 24/7/365 throughout Internet. The basic protocols that originally were developed in 1970's, were not built for security, but with functionality as main focus. In order to increase the level of security, we typically add security mechanisms such as IDS.

2.1.2 TCP flags and their usage

Fig. 1 shows the header of a TCP packet. The two fields; (i) *reserved* and (ii) *flags* are of most interest. The flags field is used to control the TCP connection.

Originally the RFC-793 defined the reserved and the flag field both 6-bits long. Resent RFC's have used 9 bits for the flag field at the expense of the reserved field (RFC-3168; (i) ECN flag (Explicit Congestion Notification Echo) + (ii) CWR flag (Congestion Window Reduced) and RFC-3540; (iii) NS flag (Nonce Sum). These new flags are used to control (or avoid) congestion. We will not include these three new flags further in this section, TCP congestion is not in focus here.

Here we give the description of the flags in RFC-793:

- URG – Urgent flag
- ACK – Acknowledge flag
- PSH – Push flag
- RST – Reset flag

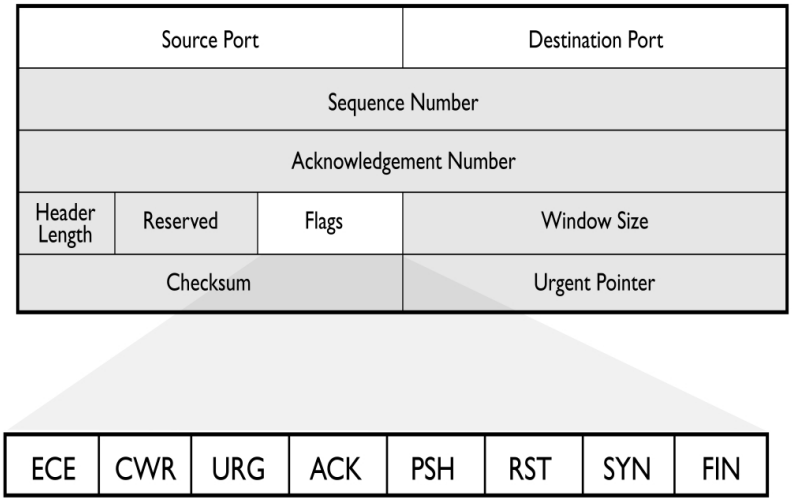

Figure 1: TCP Header - Flags. Illustration is taken from: [2].

- SYN – Synchronize flag
- FIN – Finish flag

Sources: [2, 33, 36–38]

2.1.3 TCP/UDP ports and services

Internet Assigned Numbers Authority (IANA) [39] is an organization responsible for the global coordination of the DNS Root, IP addressing, and other Internet protocol resources. This also includes the coordination and documentation of known services versus known TCP and UDP ports in TCP/IP. IANA continuously updates the list of known services vs. known ports etc. in a document available online. This is a huge list spanning over 22311 lines (7 Nov 2013); (i) header = 58 lines, (ii) service, port number, protocol, comment = 15893 lines, (iii) other known service names and description = 1283 lines, (iv) contributors (names, companies and email addresses etc.) = 4978 lines [40].

This list is a recommendation for how networked TCP/IP equipment can interconnect. Examples of this kind of naming and port standard is; (i) electronic mail (SMTP service on TCP port 25) and (ii) web (HTTP service on TCP port 80). This list that IANA coordinates is most likely one of the building bricks that make Internet worldwide a success in interconnecting matter.

The IPv4 TCP header's source and destination port fields consist of 16 bits length - and thus are capable of having 65536 different ports (0-65535). Port null ("0") is defined as reserved. This list shows how IANA have divided these 65535 ports:

0 – 1023 : System Ports – also known as the Well Known Ports (assigned by IANA)
1024 – 49151 : User Ports – also known as the Registered Ports (assigned by IANA).
49152 – 65535 : Dynamic Ports – also known as the Private or Ephemeral Ports (never assigned).

An example of service name and port mapping is shown in Appendix B.1. Sources: [6, 41–43].

2.2 Port Scanning

In this section we describe port scanning.

2.2.1 What is port scanning?

Port scanning is a technique used to survey one or more network connected hosts for availability. Port scanning is often called network scanning. We may scan a host for more specific services. Typically we may check that one server responds on TCP port 80 (HTTP) to ensure that our Web service is up and running. In our context, port scanning is in general considered malicious if not stated otherwise. Sources: [19, 44].

2.2.2 Who performs port scanning?

Port scanning may be one of the most typical activities in a computer network. Port scanning is in general performed by (i) network administrators and consultants, (ii) monitoring applications, (iii) non targeted attackers, (iv) targeted attackers and (v) applications. Sources: [16, 19, 44–46].

2.2.3 Why do we scan ports?

Network administrators perform port scanning in their local network/intranet for troubleshooting purpose. They may also perform external scans against their own equipment to perform penetration testing. Consultants do most port scanning as part of vulnerability and penetration testing (security audit). This process may also include internal port scanning.

Monitoring applications normally includes frequent port scanning in the detection process. This makes them detect new equipment.

Non targeted attackers are a category that includes; (i) malware that uses port scanning in their search for vulnerabilities, (ii) human individuals that do port scanning; just for fun (script kiddies) and/or are learning new skills and do not know what they play with.

Targeted attackers are fully aware of what they do in their port scanning. They search for interesting hosts/targets that may fulfill their needs. They most likely have a plan and often look for ways to earn money.

Many applications use port scanning. This is typical part of the initialization phase where they explore the environment on the actual computer (other services running?, ports open/closed?). Examples of applications that use port scanning; (i) VoIP applications (e.g. Skype^{TM} [16]), online game applications and other applications that use Peer-to-Peer functionality (P2P, RFC-5694 [47]).

Other categories may be faulty applications and/or servers that loose packets or struggle to manage to respond a normal TCP session timeout. Sources: [16, 19, 44–46].

2.2.4 Port Scanning Categories

Port scanning is divided into four main categories; (i) vertical, (ii) horizontal, (iii) strobe scan and (iv) block scan.

Vertical Scan

A vertical scan targets several hosts for the same port/service. E.g. the attacker searches a whole network subnet (e.g. 11.11.0.0/16) for the web service on TCP port 80 (HTTP).

Horizontal Scan

A horizontal scans targets one host for the availability of several ports/services. E.g. the attacker scan TCP ports 1-1023 on a single IP address (e.g. 11.11.11.11).

Strobe Scan

A strobe scan use both the vertical and the horizontal scan method.

Block Scan

A block scan is a complete scan on all ports/services on many hosts (e.g. network 11.11.0.0/16 + TCP ports 1-65535).

Port scanning may be executed by one or more hosts. A port scan process using several hosts is called a *distributed scan.* This kind of scan is typically performed by several hosts on different network (and IP ranges). A distributed scan is also called a *coordinated scan*. It is very efficient and stealthy to use several distributed/coordinated hosts to performing port scanning. To summarize;
– Single source port scan = One-to-One or One-to-Many
– Distributed port scan = Many-to-One or Many-to-Many

Sources: [44, 48, 49]

2.2.5 Port Scanning techniques

The simplest way of performing port scanning is to try to connect to every TCP port from 1 to 1023 on the victim host by e.g. an Internet browser using the following in address

```
1 (http://<victim-IP-address>:<TCP-port>)
```

A normal TCP connection is done by using the sequence shown leftmost in fig. 2. We complete all three sequences: SYN + SYN/ACK + ACK.

A SYN Scan is shown rightmost in fig. 2. This is not a complete sequence to establish a TCP session, and are therefore called a SYN scan. This scan may not be logged or registered by security systems if this is performed in a stealthy way (slowly).

Table 3 shows a TCP session and what TCP flags that normally are used when data are sent back and forth between hosts.

Figure 2: Normal TCP sequence (left) and SYN Scan (right). Ill. taken from [3].

There exists a lot of scans. We categorize port scans into the following list:

- Open Scan
- Half Open Scan
- Stealth Scan
- FTP Bounce Scan
- Fragment Packet Scan
- UDP Scan

Port scanning may also be used in an exhausting/denial of service (DoS) attack. This is terrifying easy to perform, but generates (of course) a lot of noise (alarms/logs etc.).

In our thesis we use scans that belongs to the category *"Stealth Scan"*, *"Open Scan"* and *"Half Open Scan"*.

Port scanning may also give the scanner much more information than the status of the actual port (open, closed or filtered (behind a firewall) port). Port scanning may also unveil the vendor, the operating system, the version of the application (that answers on the port) etc. This is important to know when evaluating our assets and our approach to detecting more slow port scanners.

Sources: [50–52]

#	Host 1	Host 2
1	SYN	
2		SYN/ACK
3	ACK	
4	PSH	
5	PSH	
6	PSH	
7		ACK
8		PSH
9		PSH
10	ACK	
11	PSH	
12		ACK
13	FIN	
14		FIN/ACK
15		PSH
16		PSH
17	ACK	
18		FIN
19	FIN/ACK	

Table 3: A typical TCP session. Table is taken from [6].

2.2.6 Commonly used TCP Scans

The following attacks are very often seen; (i) ACK, (ii) SYN, (ii) TCP Connect, (iv) FIN, (v) NULL and (vi) XMAS [14, 49]. We will now describe each of these scans. Note that we do not include any filtering mechanism (e.g. firewall) in this section, we just follow the normal responses that RFC-739 defines.

ACK Scan

sends TCP packets with only the ACK flag set. This scan may get two outcomes as a result; (i) no response (the port is open) or (ii) a RST packet (the port is closed).

SYN Scan

sends TCP packets with only the SYN flag set. This scan may get two outcomes as a result; (i) a TCP packet with SYN + ACK flags set (the port is open) or (ii) a RST packet (the port is closed. This scan starts a normal TCP session - but it does not finish the TCP session establishment with an ACK: it is only half finished. See fig. 2.

TCP Connect Scan

scan is a the odd scan in this collection of scans. A TCP Scan use the *connect ()* system call against its victim. TCP Connect Scan do not use TCP flags. This kind of scan is often recognized and logged by servers.

NULL Scan

sends TCP packets with no flag set. This scan may get two outcomes as a result; (i) no response (the port is open) or (ii) a RST packet (the port is closed).

FIN Scan

sends TCP packets with only the FIN flag set. This scan may get two outcomes as a result; (i) no response (the port is open) or (ii) a RST (the port is closed).

XMAS Scan

sends TCP packets with FIN, PSH and URG flags set (lighting the packet up like a Christmas three). This scan may get two outcomes as a result; (i) no response (the port is open) or (ii) a RST (the port is closed).

We summarize the different scans and the possible outcomes in Table 4.

Scan	TCP Packet	Possible Response	
Type	Flags Set	Port Open	Port Closed
ACK	ACK	none	RST
SYN	SYN	SYN+ACK	RST
TCP Connect			
NULL	none	none	RST
FIN	FIN	none	RST
XMAS	FIN+PSH+URG	none	RST

Table 4: Our different scans and possible responses.

Sources: [44, 53]

Scanning ports in milliseconds

A vertical port scan can typically be executed within seconds, given proper conditions (soft- and hardware). A normal port scan may not include the complete port range from 1-65535 but rather most common used/well-known ports. An NMAP command without any other options than an IP address (e.g. `nmap 192.168.1.146`) managed to ping 1000 TCP ports in 5.02 seconds (using an outdated portable computer with Linux). The command and output is listed below:

NMAP - Quick Port Scan: 1000 ports in 5.02 seconds

```
2013-10-07 20:56:33 root@piggy:~ ] # nmap 192.168.1.146

Starting Nmap 6.25 ( http://nmap.org ) at 2013-10-07 20:56 CEST
Nmap scan report for 192.168.1.146
Host is up (0.0031s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 28:C6:8E:F5:8C:D0 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 5.02 seconds
2013-10-07 20:56:46 root@piggy:~ ] #
```

2.2.7 Slow Port Scanning

Slow port scanning is performed for a reason: to avoiding being detected by monitoring systems like NIDS. Slow port scanning is the stealthy way of perform a port scan.

Detecting slow port scanning is in general just a question of counting unsuccessful TCP connections over time. In addition to this we must record odd handshake behavior and non standardized protocol usage. Slow port scanning may not be difficult to detect - it is more a question of security level versus man hours and equipment expenses. Slow port scanning describes e.g. a port scan that from a victim and/or an IDS point of view is executed slower than today's normal performance of hardware and software. The slow scanning can typically scan ports and/or hosts in an interval (frequency) of 1, 10, 60, 300 seconds or even slower.

2.3 Intrusion Detection Systems (IDS)

In this section we defines what an IDS is. We describe detection models and classify them depending on their usage area. We finally describe how we normally evaluates IDS.

An IDS tries to classify input data into two different categories; normal/benign or malicious/unwanted. These IDS's can either be special made hardware and software bundled together, or software that is capable of running on industry standard operating systems and common computer hardware (server) (or even a mix of these categories).

The term IDS is very general. We tend to use two different terms for these systems - depending on how they are implemented/installed:

- Intrusion Detection Systems (IDS) – These intrusion systems are passively implemented and will trigger and send alerts etc. to a monitoring system without interfering any services/processes.
- Intrusion Prevention Systems (IPS) – These intrusion systems are installed in a more active way. They are in-line and may e.g. filter network traffic. In addition to triggering and sending alerts they may also block network traffic that is classified as malicious/unwanted.

Sources: [4, 49, 54]

2.3.1 Classifications of IDS's

IDS's are all specially adapted/built for their tasks. We group IDS into the following classes:

Host-based

This IDS are tightly integrated into a host operating system and monitor system activity like execution of applications, exchange of data, system and user events etc.

Network-based

This IDS analyses and classifies network traffic/data. These IDS are often called NIDS.

Application-based

This IDS are specially made for analyze logs, data exchange, system calls etc. for a specific application.

Target-based

This IDS is specialized to verify data communication integrity. They typically calculate check sums in data traffic.

2.3.2 IDS Detection Models

IDS's use two different detection models; (i) misuse and (ii) anomaly.

Misuse Detection

A misuse detection IDS uses previously known patterns/signatures in order to classify the malicious data pattern. We need to experience the actual malicious attack/signature before we can create a pattern for the IDS. This detection method makes us always a step behind any new attack pattern/signature.

Anomaly Detection

An anomaly detection IDS collects network statistics and defines the most normal network behavior as a baseline. When this baseline is challenged in large degree (large deviation), the alarms go off. This kind of IDS can produce large number of false positives, but are at the same time very efficient against new/unknown network attacks.

2.3.3 Evaluating an IDS

Evaluating an IDS may initially seem as an easy task. It is all about counting alerts versus actual attacks/unwanted traffic pattern detected. Unfortunately, this is not the case, on the contrary. Many articles and books have been written about evaluating IDS [6, 55–60]

The following list describes some measurable characteristics for evaluating IDS:

- Coverage: What does this IDS cover? Known and unknown attacks? Malware? Port Scan? etc.
- Percentage of false alarms: What is the probability of any false alarm?
- Detection Rate: What is the detection rate for this IDS in a given case?
- Capacity: How high bandwidth and/or CPU load will the IDS manage?
- Resistance to Attacks: How resistant is the IDS against a direct attack?
- Other: Learning capabilities? Configuration complexity? Upgrade/update capabilities? Operating complexity?

We will not perform any tests regarding traffic capacity (bandwidth), CPU, memory etc. in our thesis. However, we want to measure the detection capability. We want to calculate the so-called detection rate for our IDS.

Table 5 shows the so-called confusion matrix that gives us the four categories where the IDS will place its decisions weather the data (network traffic) is classified as benign or malicious. Table 6 shows the expression used in context to IDS.

		Predicted Class		
		p	**n**	**total**
Actual Class	**p′**	True Positive (TP)	False Negative (FN)	P′
	n′	False Positive (FP)	True Negative (TN)	N′
	total	P	N	

Table 5: Confusion matrix.

Term	Actual Intrusion	IDS Alarm
True Positive (TP)	1	1
True Negative (TN)	0	0
False Positive (FP)	0	1
False Negative (FN)	1	0

Table 6: Definition of True Positive, False Positive, True Negative and False Negative.

The most used evaluation metric is the True Positive Rate (TPR), also called the detection rate. The TPR is calculated using the following formula:

$$TPR = \frac{TP}{TP+FN}$$

2.4 Common Network Intrusion Detection Systems (NIDS)

In this section we mention some well-known NIDS, both commercial and open source.

Most prominent network equipment producers have in general one or more NIDS to offer; (i) Cisco Systems (Cisco IPS), (ii) Juniper Network (Juniper Network IDP), (iii) CheckPoint Software Technologies (Sentivist), (iv) Symantec Corporation (Symantec Network Security), (v) IBM, (vi) McAfee etc. to mention some of them [54, 61, 62].

Popular open source alternatives are Snort, Suricata and Bro [63–65].

2.5 Snort NIDS

In this section we describe Snort NIDS. We start with a brief explanation of Snort architecture and continue with rules and port scan detection mechanism.

Snort was released by Marin Roesch in 1998 [66]. Today Snort is a registered trademark under Sourcefire, Inc. [63]. Snort is open source and has had an active community since its birth. Snort has become the de facto standard in network intrusion detection systems (NIDS) [67–70].

2.5.1 Snort Elements

Snort is a so-called signature (or misuse) based NIDS. Snort consists of the following elements:

- Packet Decoder (e.g. `libpcap`)
- Pre-processors
- Detection Module
- Logging and Alerting Module

Snort elements are illustrated in fig. 3.

2.5.2 Snort Rules

Snort uses so-called *rules* in order to define the signatures of malicious patterns. These rules are placed in *rule files* with one rule defined in one line.

```
1 <action> <protocol> <source-IP-address> <source-port> <direction> <dest-IP-address> <
    dest-port> (<options>);
```

```
1 alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; dsize:>128; classtype:
    attempted-admin; priority:10 );
```

These rules are published by many communities in addition to commercial companies. Sourcefire offers both community rules and commercial rules [63]. These rules can be quite complex and may include regular expressions and several low lever inbuilt filters. Today, the number of Snort community rules = 2 753 and the Snort commercial rules = 22 059 (VRT Certified Rules).

2.5.3 Port Scan Detection

Snort's port scan detection mechanism is by the so-called preprocessor: `sfportscan`.

The following list show the possible configurable parameters for sfportscan found on Snort Web page:

```
preprocessor sfportscan: proto <protocols> \
        scan_type <portscan|portsweep|decoy_portscan|distributed_portscan|all> \
        sense_level <low|medium|high> \
        watch_ip <IP or IP/CIDR> \
        ignore_scanners <IP list> \
        ignore_scanned <IP list> \
```

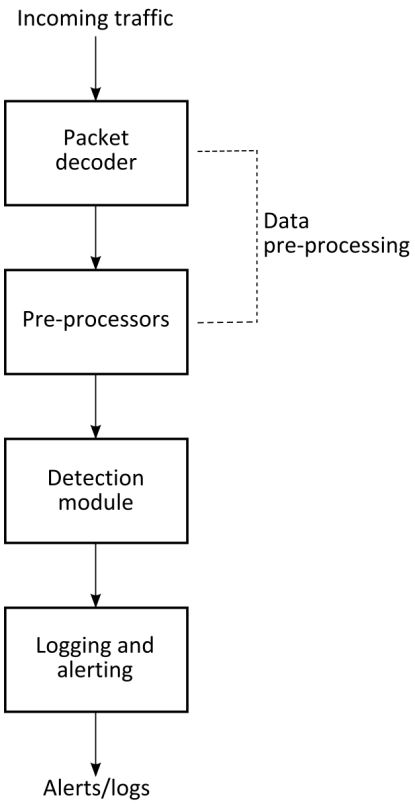

Figure 3: Snort Elements. Illustration taken from [4].

```
logfile <path and filename> \
disabled
```

The `sense_level` is the parameter that we will tune in order to detect slow port scanning. The differetn levels, low, medium and high are defined like this (quoted from Snort Web page):

> low - “Low” alerts are only generated on error packets sent from the target host, and because of the nature of error responses, this setting should see very few false positives. However, this setting will never trigger a Filtered Scan alert because of a lack of error responses. This setting is based on a static time window of 60 seconds, after which this window is reset.
>
> medium - “Medium” alerts track connection counts, and so will generate filtered scan alerts. This setting may false positive on active hosts (NATs, proxies, DNS caches, etc), so the user may need to deploy the use of Ignore directives to properly tune this directive.
>
> high - “High” alerts continuously track hosts on a network using a time window to evaluate portscan statistics for that host. A "High" setting will catch some slow scans because of the continuous monitoring, but is very sensitive to active hosts. This most definitely will require the user to tune sfPortscan.

Sources: [66, 70, 71].

2.6 Bro - Intrusion Detection System

In this section we describe Bro IDS in more details.

Paxson published in 1998 a paper called *"Bro: A System for Detecting Network Intruders in Real-Time"* [72]. This paper describes an intrusion detection system called Bro. Bro was intentionally a stand alone system for detecting network intruders in real time. Bro is open source software, written in C and is capable of deep & stateful packet inspection at very high speed. Bro has been embraced by research communities in many academic institutions in the last decade [12].

2.6.1 Bro NIDS

Bro NIDS (Bro for short) was initially designed to be a powerful real-time network traffic analyzing tool. Here is a list of The Bro Project's design philosophy criteria [72]:

- Real-time network analyzing framework
- Separate packet collector mechanism from policy/analysis mechanism (avoid packet filter drops et.al)
- Neither anomaly or misuse/signature architecture - a network analyzing framework
- Capable of analyzing high-performance networks in large scale
- A script language that helps operators avoid mistakes (because of its simplistic structure)
- Comprehensive log facilities (which makes forensics community pleased)
- Open interface to exchange data to other applications in real-time
- Open Source which makes this software available for free usage in general (BSD) [31]
- Powerful script engines for extensive customization
- Highly aware of the high possibility that Bro will be attacked

2.6.2 Bro - Internal Architecture

Bro internal architecture consists roughly of the following elements; (i) libpcap, (ii) an event engine and (iii) a policy script interpreter. Fig. 4, illustrates Bro internal architecture [72].

In order to understand how Bro works we need to go into more details regarding its architecture. The Bro architecture details are illustrated in fig. 4 [30]. We explain this fig. further in the following paragraphs.

Network

Bro needs a physical network connection to get a copy of the network traffic it will analyze. This is normally done by the use of port mirroring functionality in switches/routers or a TAP device [73, 74].

libpcap

In order to get traffic data from the physical network, a so-called Application Program Interface (API) is needed. Bro uses `libpcap` [75]. `libpcap` is a C/C++ library for network traffic capture. This is the abstract layer between the physical network medium and the operating system. With the use of `libpcap` Bro filters traffic that is sent to Bro Event Engine.

Event Engine

The filtered network data packages from `libpcap` are then fed into the next level; the `Event engine`. This event engine tries to reassemble all the network traffic it gets to known events/patterns as high as possible in the OSI ISO Model [76]. Typically the event engine finds connection attempts (transport-level), FTP requests/replies, HTTP requests/replies (application-level) and login failed/success (application level) [77, 78].

The event engine performs several health checks and tries to reassemble the packets:

- Integrity checks (are the packet headers intact? are the IP packet headers correct regarding check sums? etc.)
- If integrity checks $\neq$ OK; write an error event + discard packet
- If IP packets; Reassemble IP fragments into datagrams
- If integrity checks = OK; look up the connection state with associated; (i) source IP address, (ii) desti-

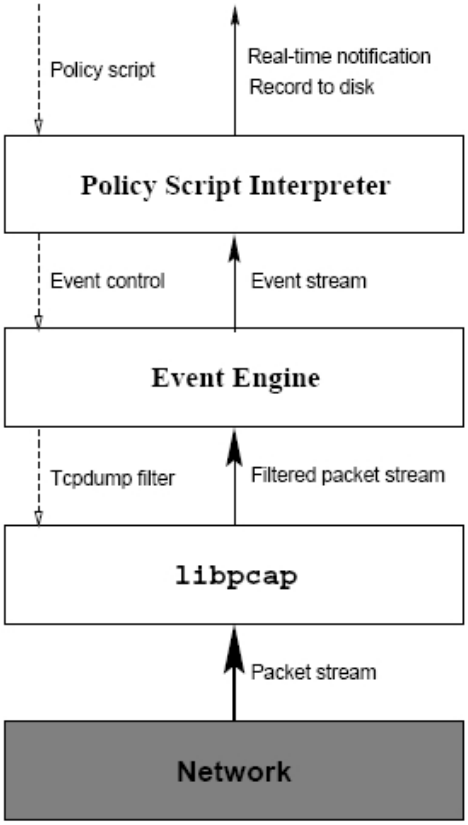

Figure 4: Bro Internal Architecture. Illustration taken from [5].

nation IP address and (iii) TCP or UDP port numbers

- Dispatch the packet to a connection handler (TCP or UDP) the further corresponding connection.

TCP connection handler

For every TCP package the connection handler performs the following actions;

- Verify; (i) TCP header, (ii) TCP packets header checksum and (iii) TCP packets payload checksum
- If verification above = OK; Are there any `SYN/FIN/RST` control bits/flags?
- If flags above are present; set the actual connection state to the active control bit/flag.
- Process other data acknowledgement in header (if any)
- Process payload data (if any).

UDP connection handler

The UDP connection handler is similar to TCP but much simpler because it is connectionless (e.g. no connection state). But – UDP sessions use different ports when starting a UDP packet stream then replying this UDP stream. This states are called pseudo connection states.

Policy Script Interpreter

The policy script interpreter processes events from the event engine. For every event handled to the policy script interpreter, it performs the following steps:

1. Look up the corresponding event handler's (semi-)compiled code/script
2. Bind the value(s) of the event(s) to the argument of the handler

3. Interpret the actual event code/script

The policy script interpreter is in general an event handler. The result of this process can execute further scripts/commands including; (i) generate new events, (ii) log events, (iii) invoke other event handlers.

Bro ships with a large number of readymade policy scripts for various types of analysis. When adding new functionality to Bro it writes a new protocol analyzer to the event engine and/or a new event handler in the policy script interpreter.

2.6.3 Bro Log Files

Bro is shipped with an interactive shell for management purpose: BroControl [79]. This application is able to control and monitor the Bro installation. In a cluster and multi Bro installation case BroControl is crucial. When using BroControl, Bro creates logs in the directory; `$BROHOME/log`. The "working" directory is `$BROHOME/log/current` but logs are frequently moved to `$BROHOME/log/YYYY-MM-DD` where YYYY, MM and DD are the digital representations of year, month and day, respectively. These log files are in clear text (ASCII) unless default configuration is changed [80].

When running from CLI, all log files are created in actual directory where we start Bro. The following log files are always created (even without any traffic detected):
`conn.log, loaded_scripts.log and notice_policy.log`. These filenames reveal much of the actual log file content, but some more description is necessary:

`conn.log` — consists of the complete connection log during Bro's run time. The file consists of twenty columns (timestamp, connection ID (unique), source IP, dest IP, source port, dest port etc.).
`loaded_scripts.log` — shows Bro scripts (*.bro) that were loaded during Bro startup.
`notice_policy.bro` — shows the current Bro Notice policy.

Bro create several new log files during run time. This overview shows more general and internal log files (incomplete list):

`communication.log` — logs for Bro's internal communication between remote and central instances, clusters etc.
`conn-summary.log` — generated when Bro is terminated. Post processing connection summaries
`known_hosts.log` — hosts that have performed complete TCP handshake
`notice.log` — notices that Bro rises
`reporter.log` — internal messages/warnings/errors for troubleshooting.

Bro also creates a lot of log files that are protocol/service specific (incomplete list):

`dns.log` — log over DNS queries
`dpd.log` — log over what port/service dependent dynamic protocol detection analysis that has been activated
`http.log` — log over HTTP request and responses including metadata
`software.log` — reports known/recognized software detected from protocol analyzers
`weird.log` — notices that Bro has tagged as weird. Odd protocol behavior will be logged here.

Other (rather self explaining) protocol scripts are found in `$BROHOME/share/bro/base/protocol/*`; `ftp, irc, modbus, smtp, socs, ssh, ssh, syslog`. We most likely get log files created by each of these scripts (depending on the criteria in the actual script).

Source: Bro 2.1 file structure (`$BROHOME/share/bro/base/frameworks/*`), see Appendix C.3 and Bro web page [78, 81].

2.6.4 Bro Policy Scripts Structure

Currently Bro describes more than 260 different scripts on their Web pages (including Bro 2.2 beta release) [82]. However, this includes every category of their scripts; (i) internal communication scripts, (ii) file & protocol analyzing scripts, (iii) built in functions (BIF), (iv) notice & logging scripts and (v) policy scripts.

We focus in the following paragraphs on the `scan.bro` Bro policy script. This kind of scripts follows a basic structure:

Header

The first lines describe the script. Lines with comments have an initial hash sign (#).

The next lines load scripts/modules that this script uses. This is done by the following syntax: `@load <module/script-name>` (in our case: the Notice Framework is loaded):
`@load base/frameworks/notice/main`.

The last part of the header defines the script name of the actual script by the command `module <script-name>` (in our case: (`module Scan`)).

Export Variables Declaration

In this section we declare what variables we want to export to Bro globally. We may redefine existing global variables (`redef` statement) and add new global variables (`global` statement). We can also add constants (`const` statement) that we then may use by other scripts. In the original `scan.bro` the export variables declaration spanned over 183 lines.

General Script Code

In this section we find functions (large scripts) that include general programming statements like;
`if{},while{},for{}` etc. Local variables are also declared as needed (`local` statement).

Event(s)

In this section we describe what action has to be taken when an event occurs/is triggered. These events may use local and/or external functions and variables to evaluate what action to perform.

3 Previous Work

"If I have seen further it is by standing on the shoulders of giants."
– Isaac Newton

In this chapter we focus on previous work regarding IDS, port scanning in general and slow port scanning in specific.

3.1 Intrusion Detection Systems

In this section we describe and discuss important research that define intrusion detection and are closly related to our thesis topics.

Denning, D.E. published in 1987 an article called *"An Intrusion-Detection Model"* [83]. Dennings introduce the term Intrusion Detection Expert System (IDES). This models goal is to detect, alert and block security violation in real-time. The IDES model is suggested as an independent system that is able to monitor other computer systems e.g. IDES is suggested as an framework that is highly adaptable/configurable in order to fit most systems and organizations needs. The model have six main components; (i) *Subjects* (indicators of activity on target system, user activity), (ii) *Objects* (resources managed by the system (system-files, devices, commands etc.)), (iii) *Audit records* (Log of audited activity), (vi) *Profiles* (knowledge that characterize different parts in system (statistics, automatic generated)), (v) *Anomaly Records* (generated when abnormal behavior is detected), (vi) *Activity Rule* (pre defined actions to take when anomalies are detected). The metrics used in monitoring is; (i) Event Counter, (ii) Interval Timer and (iii) Resource Measure. The article is easy understandable with detailed examples on how an IDES model could be implemented using different statistical and parametric approach.

The general challenge with this model is most likely the never ending story in intrusion detection: systems/applications/network and users etc. are continuously changing behavior. This way there will be many false positives to fight.

Anderson et.al. at SRI Int. published in 1995 a report called *"Next-generation Intrusion Detection Expert System (NIDES)"* [84]. NIDES is clearly built on ideas from IDES model by Dennings. This report describes in an IDS that was capable of anomaly detection in real-time. SRI Int. also built this software. They include both known intrusion scenarios (misuse detection) and advanced statistics (anomaly detection) in their model. This system logged extensively with both short term focus and long term focus (historical). The system compared long term data with short term data in order to continuously adapt (learn) the level of normal behavior (baseline). Critical voices thought this system was was to huge and complex (huge piles of log data), but NIDES have most likely been a reference model for many IDS projects.

They focus more on practical approach than the IDES model. The operating and maintenance of this kind of system is of course a crucial success factor, and their own experience and feedback after building this system is clearly supporting this model.

Roesch published in 1999 an article called *"Snort - Lightweight Intrusion Detection For Networks"* where he introduce Snort IDS. Roech calls his open source software "lightweight" because of its file size (appr. 100kb), its easy configurable files and easy installation procedure. Snort main elements are; (i) ap acket decoder, (ii) pre-processors, (iii) detection module/engine and (iv) a logging/alerting system. Snort is based on a misuse detection model where signatures are placed in so called rule files. These rule files

are easily understandable in plain English. Snort can be used as an inline, passive and a packet capturing NIDS.

Snort have been a popular NIDS after its birth. It is indeed not a "lightweight" anymore, and are now a mature and powerful NIDS with a port scan detection module called sfportscan. Malmedal challenged this module in his master thesis. We will also test out this module in this thesis.

Paxson published in 1998 an article called *"Bro: A System for Detecting Network Intruders in Real-Time"* [72]. This article introduce a new open source NIDS. Paxson wanted to build a NIDS in order to show how this could be done (the existing NIDS that did the same was commercial, closed code software). Bro is a stand-alone system with real-time traffic analyzing functionality. It is designed with the following goals; (i) high speed, large volume monitoring, (ii) no packet drops, (iii) real-time notification, (iv) mechanism separate from policy (for simplicity and flexibility), (v) extensible (module based in order to be in production and parts may be updated), (vi) Avoid simple mistakes (easy understandable script language), (vii) The monitor will be attacked (secure the IDS itself against attacks). Bro architecture is based on (i) libpcap, (ii) event engine and (iii) a policy script interpreter. Bro comes with an impressive script language with several ready made protocol analyzing scripts ready made. The scripts are compiled into C++ code. Bro uses both anomaly and misuse detection methods. Bro was developed and used in production at ICSI [85] from 1996 to 1998, so they had a lot of experience and valuable knowledge put into this software.

Bro have been developed further since its official birth in 1999. Today we have version 2.2 that was released in stable version 7 Nov 2013. The original adaption to Snort rules is no longer included in Bro. We have explored Bro quite thoroughly throughout our thesis.

Sommer et.al. published in 2010 a paper called *"Outside the Closed World: On Using Machine Learning For Network Intrusion Detection"* [86]. Sommer et.al. start this paper with the question; *Why is not machine learning (ML) techniques more used in NIDS?*. They mention other areas where ML have great success; (i) product recommendation systems (Amazon, NetFlix), (ii) optical character recognition systems, (iii) natural language translation and (iv) spam detection. These mentioned areas are discussed regarding ML. The diversity in computer network traffic is the challenge when applying ML in NIDS. They further suggest focus points when applying ML based NIDS in a network; (i) understand the threat model (know your network and its enemies), (ii) keep the scope narrow (what are the targets weakness), (iii) reducing the costs (do proper planning including risk assessment regarding changing systems vs. your needs), (iv) evaluation (always evaluate your NIDS performance). They conclude on the fact that there is a surprising imbalance between research in ML and NIDS and actual deployed systems in production.

There are published a lot of articles regarding machine learning in NIDS. This paper clearly states the challenges in applying ML in NIDS.

3.1.1 Network ScanningSurveys & Taxonomies

Barnett et.al. published in 2008 an article called *"Towards a taxonomy of network scanning techniques"* [44]. This paper summarize and categorize known scanning techniques and illustrates some of them in three dimensions figures. They also briefly describe some scan analyzing methods.

This paper is a nice approach to get an overview of known scanning categories techniques.

3.2 Port Scan Detection

We will in this section describe research that covers port scan detection.

Zhang et.al. published in 2008 a paper called *Scan Attack Detection Based on Distributed Cooperative Model* [87]. They propose a distributed cooperative model for intrusion detection. This model consists of five layers; (i) sensor, (ii) event generator, (iii) event detection agent, (iv) fusion center and (v) control

center. The model describes three different way of detecting an intrusion; (i) feature-based, (ii) scenario-based and (iii) statistics-based. The fusion center analyzes the results from the three detection mechanisms and may generate alerts up to the control center.

This article claims improved scan detection capabilities (also for slow port scanning). We have not found any research supporting this. The model seems smart though, but we need a lot of sensors and double analyzing units that can correlate the results. This way we will most likely get a better detection rate.

Ertöz et al. published in 2004 a paper named *The MINDS - Minnesota Intrusion Detection System* [88]. This paper present a new IDS model using machine learning techniques. The MINDS is a data mining based system for detecting malicious traffic/network intrusions. They use NetFlow to filter out traffic for further analyzing. The detection mechanism consists of both anomaly and misuse/pattern matching techniques in their detection model. The anomaly processing is based on an unsupervised learning technique. [13]

Ertöz et al. describe MINDS's capability to detect stealth port scanning, but we have not found any research supporting this.

Jung et.al. published in 2004 a paper called *"Fast Portscan Detection Using Sequential Hypothesis Testing"* [89]. Initial they discuss the problem with anomaly NIDS and false positives. They further use captured network traffic data and manage to find a connection with this data and the theory of sequential hypothesis testing. This data is captured by using Bro. With their research they purpose a new algorithm: TRW (Threshold Random Walk), an online detection algorithm that identifies malicious remote hosts. This method was proved to be 4-5 times more efficient than previous methods.

Bro uses TRW algorithm in the version we test in our thesis.

3.3 Detecting Stealthy Port Scans

We will in this section look at previous work related to our focus on slow port scan.

Staniford et.al. published in 2002 a paper called *"Practical automated detection of stealthy portscans"* [90]. This paper goes deep into how network scans are performed and show examples from logs. They look at several previous work. They introduce *SPICE (Stealthy Probing and Intrusion Correlating Engine)*, which consists of two components; (i) an anomaly sensor and (ii) a correlator. They collect traffic pattern that is defined as anomaly for later analysis. They use simulated annealing algorithm to cluster anomaly network traffic pattern. They also have adapted SPICE into the Snort plugin SPADE (Statistical Packet Anomaly Detection Engine). They show good results after their live tests (one of them over 3 months).

This paper have a very thorough introduction into network scans with some practical examples. We lacked more details regarding their slow port scanning. E.g. how did they defined slow port scan?

Kim et.al. published in 2008 a paper: *"A Slow Port Scan Attack Detection Mechanism Based on Fuzzy Logic and A Stepwise Policy"* [91]. This paper purpose a way of detecting slow port scanning by using fuzzy logic and a stepwise policy. Kim et.al. have developed an Abnormal Traffic Control Framework (ATCF) that consists of the following elements; (i) an intrusion detection module (with packet analyzing (PA) and intrusion analysis (IA) module, (ii) an intrusion prevention module (with packet filtering (PF) and queuing assignment (QA) module) and (iii) a control module. If anomaly traffic and/or an attack is detected, the PF alerts the QA and blocks this traffic. A slow port scan is handled by using stepwise shaping; (i) if classified as suspicious - bandwidth is reduced to minimum, (ii) if classified as attack - traffic is blocked. The decision regarding a port scan is normal, suspicious or an attack is done by an fuzzy logic rule table.

This paper defines slow port scan when delaying packets this long as; (i) 1, (ii) 10 and (iii) 1000 milliseconds. We do agree that the last category (iii) is somewhat slow, the other not. Kim et.al. did not include this last category (iii) in their test and we will therefore not follow this paper further.

Malmedal finished his master thesis in 2005 called "Using Netflows for slow portscan detection" [92]. Malmedal challenge Netflow, Snort and Network Flight Recorder (now CheckPoint IPS-1) regarding slow port scanning. He sat up a network flow (netflow) analysis system using Argus [93], PostgreSQL [94], php [95] and jgraph [96]. This netflow system stored all network data in a database for non real-time analyzing. Network Flight Recorder is Cisco Corp.'s IDS/IPS (a commercial product). Malmedal chose NMAP [51] to perform slow port scan using port range 0-1000 and scan interval 60 sec.

Malmedal's thesis show (again) that slow port scanning is not easy to detect. Especially in this case when scan interval is 60 sec.

Dabbagh et.al. published in 2011 an article called *"Slow Port Scanning Detection"* [14]. This article purpose a way to detect slow TCP port scan. They collect all traffic data over a longer time window and extract features of every TCP connection and/or connection attempt. They tested their software using three different slow port scan intervals; (i) 0,4ms, (ii) 4 min and (iii) 6 min.

In addition to their slow port scanning detection method they have a valuable introduction covering other articles. This down to earth counting of abnormal TCP connections may generate a great deal of false positive in live network traffic. A lot of port scanning is performed by non malicious intentions (applications and system administrators) [16, 19].

No previous work regarding Bro and slow port scanning

We did not manage to find any research where slow port scan and Bro was involved. On the other hand, there are many books and papers about port scanning techniques in general, which can be very useful in this research.

4 Choice of Methods

"Science is a method to keep yourself from kidding yourself."
– Edwin Land

In this chapter we describe our methods needed to answer our research question.

4.1 Scientific Methodology

In this section we describe the scientific method used in our thesis which seeks answer to the following sub questions:

RQ1.1 Will we be able to improve the slow port scan detect rate in Bro?

RQ1.2 What is the slow port scan detection rate in Bro?

We need to modify and test Bro IDS several times to be able to answer RQ1.1 & RQ1.2. This tells us that *quantitative* research method is necessary to answer our research questions. We directly compare results of repetitive tests and apply statistical calculations where needed [6, 97].

4.2 Experimental Method

In this section we explain our details in order to fulfill RQ1. We chose to challenge Bro NIDS regarding stealthy port scans. Bro is a very interesting NIDS with powerful script language and analyzing capabilities. We have most likely improved Bro's slow port scan capabilities and will compare our new script with the original.

We did not find any previous research regarding Bro and slow port scanning. We did find a lot of work that covered stealthy network scan detection. We also looked at Malmedal's work from 2005 [92].

4.2.1 Reliability and Validity

We will in the following paragraphs describe our experimental test strategy. It is crucial to bear in mind the meaning of these two terms: (i) reliability and (ii) validity. We may bring an average project up to a higher level with the right focus in our scientific research. We will also list up our more technical decisions in our test lab setup.

Reliability

Reliability in research is the focus on supporting our findings. Our findings must be; credible/trustworthy (reliable) and able to reconstruct.

We have in our thesis ensured reliability by the following strategy:

- Quality Assurance routines (risk assessments; backup, checklists etc.)
- Comprehensive logging of all tests
- Open source software on both; (i) operating systems, (ii) IDS software and (iii) monitoring software
- Standard computer hardware
- Repetitive tests
- Well documented test results
- Common statistics tool used on both IDS.

The daily checklist for our test lab is given in Table 4.2.1.

Validity

Validity in research is the focus on how valid our findings are. We must ask ourselves; *"Was our research worth our efforts?", "Do we add valuable information with our research?"*

We have in our thesis ensured validity by the following strategy:

- Literature study / Planning
- Scientific Research methods
- Open and frequent dialogue with our supervisor
- Open and frequent unfiltered discussions with our fellow students
- Critical thinking regarding our results

Sources: [6, 97, 98].

Experimental Test Phase - Daily Checklist

During our test phase we had to make sure that our experiments was executed as planned. The following list show our checklist:

#	Item	Description
1	Server OS	- Check system for general error messages (/var/log/messages) + root mail
2		- Time Sync (NTP Daemon running? Clock correct?)
3		- Disk Space OK (df -h)
4		- Running Processes (ps auxwww / dig)
5		- System health (grep 'warning|error|critical' /var/log/messages)
6		- Interfaces running OK? (netstat -i, ifconfig, PROMISC mode)
7	Appl.	- Test processes running? (Bro, Snort, NMAP, Tcpdump, Argus)
8		- Check when they started last started and look for possible error messages
9	Admin	- Backup Latex code, Logs and Argus files (Dropbox + JottaCloud)
9		- Document test results in Excel.

Table 7: Experimental Test Phase – Daily Checklist.

4.3 Improving Bro slow port scan detection capability

In this section we will describe our strategy and methods for improving slow port scanning detection in Bro.

4.3.1 Strategy for improving Bro Script

We will in this section list our strategy for improving Bro slow port scan detection capabilities, and thus detection rate.

Our strategy for improving `scan.bro` is given in the following list:

- Read up on Bro (search for available documentation online, in forums and in source code and in installed file structure)
- Learn Bro policy script language in general
- Understand the existing script code involved in slow port scan
- Debug actual script(s) above if necessary
- Avoid, if possible, unnecessary resource usage
- Comment inline for better understanding of code.

It is important to add comments in code. This is a good practice in order to let other developers/readers understand thoughts and decision made.

4.3.2 Bro version

We had to decide for a experiment version of Bro to avoid any uncertainties/errors/unknown behavior during our experimental phase. We chose Bro version 2.1-328 (timestamp 2013-02-05 01:34:29 -0500) [99].

There existed several hints on Bro Web page regarding the version 2.2 was under development. This started already in January 2013. Bro version 2.2 beta was released in 24 Sept 2013. Bro 2.2 version (stable) was released 7 November 2.2

4.4 Basic Test Regime

In this section we describe, discuss and decide our basic test regime.

4.4.1 Port Scan Interval

Slow scanning is not a specific scientific term. We found no specific definition of slow port scanning. Barnett et.al published in 2008 an article called *"Towards a Taxonomy of Network Scanning Techniques"* [44]. They use the term "very slowly" (with packets beeing sent with a gap of at least five minutes between them). Treurniet published in 2011 an article called *"A network activity classification schema and its application to scan detection"*. Treurniet defines in her research that slow port scanning is when the packet interval is more than 60 seconds.

Today, we are used to move data between computers and servers within milliseconds, even between computers with physically long distance between. We browse Internet and expect music, video and/or large pictures to show up more or less instant. We will in our thesis not redefine the definition of slow port scanning, but describe it as considerable slower as normal port scanning.

We adopted the use of 60 seconds scan interval as Malmedal used in his thesis [92]. This is a proper stealthy port scan interval that we strongly think the majority of NIDS will struggle to detect.

4.4.2 Port Range

Malmedal chose to scan all TCP ports from 1 – 1000. We considered this a bit "overdoing". Our initial tests with Bro showed alerts in log already from first packet above threshold. We chose to scan randomly the 100 most frequently used TCP ports in the port range 1-1023.

We started our tests with both Bro and Snort configured in the way that every port below 1024 was handled equally. This created a lot of alerts/log when we injected background traffic. We ended up using the initial configuration in both Bro and Snort. Our conclusion in this matter is that we just have to stand on shoulders to the knowledge that both community developers have put into their NIDS.

Tuning NMAP

Our interesting ports are what IANA calls "System Ports" or "well known ports" [39]. These are the ports below 1024. We chose to scan the 100 most used ports regarding NMAP's file `nmap-services` [51].

Fig. 8 shows the top ten most used TCP ports according to NMAP's previous experience. The complete list of the 100 most used ports is available in Appendix B.1.

Service	Port	Port Frequency	Description
http	80/tcp	0.484143	# World Wide Web HTTP
telnet	23/tcp	0.221265	
https	443/tcp	0.208669	# secure http (SSL)
ftp	21/tcp	0.197667	# File Transfer [Control]
ssh	22/tcp	0.182286	# Secure Shell Login
smtp	25/tcp	0.131314	# Simple Mail Transfer
pop3	110/tcp	0.077142	# PostOffice V.3
microsoft-ds	445/tcp	0.056944	# SMB directly over IP
netbios-ssn	139/tcp	0.050809	# NETBIOS Session Service
imap	143/tcp	0.050420	# Interim Mail Access Protocol v2

Table 8: NMAP Service definition file. Top of file sorted descending using third column (Port frequency).

We asked Gordon "Fyodor" Lyon for more background knowledge regarding NMAP's file `nmap-services` and got the following answer:

```
On Fri, Oct 11, 2013 at 8:33 AM, Roger Larsen <roger.larsen@hig.no> wrote:
> Dear Fyodor,
>
> I am writing my thesis on my research on slow port scanning and Bro IDS.
> I use your brilliant tool NMAP Security Scanner in order to scan slowly
> against a victim :-)
>
> I was hoping to use the option ‘‘--top-ports 100’’ in order to get the 100 most
> used TCP ports in my scan.
> What is your source for determining the ‘‘port frequency’’ column in this file?

Hi Roger.  We did some empirical scanning a few years back to generate the
frequencies.  They’re not perfect, but they’re helpful IMHO.

Cheers,
Fyodor
```

Ports below 1024, an outdated focus?

We have in our thesis only focused on ports below 1024. IANA calls this port range for: System Ports (or well known ports)IANA [40]. This list is a recommendation/guide for how TCP/IP interconnect can be done. However, Internet traffic tends to use more and more ports above these ports. The original NMAP file `nmap-services` includes two ports above 1023 in top ten list according to the term *"Port Frequency"* index (see NMAP version 6.25) [51, 100]. R. Pang published in 2008 his PhD dissertation called *"Towards understanding application semantics of network traffic"*. He have collected internet background radiation statistics. In his statistics over most used TCP ports he have 50% of the top eight ports above 1023, the first above is on fifth position [101].

4.4.3 Scans Attacks

We wanted to broaden our test regime instead of wasting time on scanning.

We decided to perform the following six well known port scan attacks; (i) ACK Scan, (ii) SYN Scan, (iii) TCP Connect Scan, (iv) NULL Scan, (v) FIN Scan and (vi) XMAS Scan.

4.4.4 Test Network Environment

We could perform our experiments in our isolated lab and be fully satisfied about our result. However, normal network environment is in general quite chaotic, and totally opposite of isolated [22, 48, 56]. We wanted to challenge our NIDS in order to see how they managed network environment closer to normal. We chose to apply two different network environments; (i) isolated (minimum of traffic) and (ii) background (traffic injected).

4.4.5 Scan Category

We need to limit our test lab in order to manage the amount of computers. We chose to scan one victim host with one attacker, e.g. we decided to use vertical scan technique.

4.4.6 Scan Repetitions

We need to support our research strategy regarding reliability. We therefore decided to repeat every test ten times.

4.4.7 Basic Test Regime Summarized

- Port Scan Interval = 60 seconds
- Port Range = 1–1023, 100 most frequent "port frequency", randomly scanned (NMAP)
- Scan Attacks = (i) ACK, (ii) SYN, (iii) TCP Connect, (iv) NULL, (v) FIN and (vi) XMAS.
- Test Network Environments = (i) Isolated & (ii) Background Injected Traffic
- Scan Category = Vertical Scan
- Scan Repetition = 10 times, each test

4.5 Test Lab

In this section we describe, discuss and decide how we set up our test lab.

4.5.1 Operating Systems

The following paragraphs explains our choices regarding operating systems.

Bro Server

Bro support most Linux and BSD based systems, even Apple was supported in autumn 2013 (with some comments) [102].

We have previously struggled installing Bro on Linux based operating systems. When first installing Bro we tried Linux Mint 13 Maya [103]. We ended up using a lot of time in troubleshooting before we manage to get our Bro installation up and running. Our next installation of Bro was in a virtual environment (VMWARE) with FreeBSD as operating system. This installation was done without any extra problems/efforts. When we started to build up our lab we chose an old DELL computer to be the Bro server. We installed FreeBSD 9.1 on this server. This installation of Bro (on FreeBSD) was also done without problem.

Attacker, Traffic Injection, Snort Server, Victim

All the four computers here were laptops. We used outdated laptops that we borrowed from LYSGLIMT, Austevoll Kraftlag SA [1]. These computers are a bit outdated (slow responsive) when Microsoft Windows is installed (the default at Austevoll Kraftlag SA). However, these computers are powerful enough when using Linux based operating system.

Snort support most operating systems, both Linux- and BSD-based, Apple OS X and Microsoft Windows [63]. The other applications supports both Linux, FreeBSD, and Apple OS X.

We tried to standardize on Linux Mint 13, but two of our old laptops was not happy with this choice. We experienced very slow startup time and some features that did not work as expected. This ended up

with a re-installation of on the Snort Server and Attacker computer. We chose our second most favorite operating system including a less resource demanding windows manager (Linux Debian 7 with XFCE Desktop Environment) [104, 105].

After this re-installation of Snort Server and Attacker computer, we did not experience any practical problems with the choices of operating systems during our tests.

The list of equipment details is found in 6.1.2.

Victim Computer

We wanted to simulate a server with a lot of ports and services running. This way we would get a lot traffic including both open and closed ports during the NMAP scan process. Our initial tests showed that NMAP used more time the more open ports it found. We must bear in mind that NMAP was not special built for our experiment. NMAP use more time analyzing several ports than analyzing one port, this is clearly by design. NMAP is a piece of software that tries to serve its master with best possible analyzing response.

We chose to use one active service on victim computer: the SSH service on TCP port 22 [106]. This saved us time in every scan process and in the same time we were able to manage this computer.

4.5.2 Network Equipment

Our needs regarding network equipment was not very ambitious. We needed maximum 1 Mb/s traffic bandwidth (when using background traffic injection). However, it is always good to have some power in the network when we will take backup. The most common network equipment have for many years been fast ethernet, which is 100Mb/s. This is enough for our test lab.

We decided to use network equipment with link speed 100Mb/s (fast ethernet). A stress test using `tcpreplay` without bandwidth shaping parameter gave us a bandwidth speed over 70 Mb/s, so there were plenty of overhead performance in our network.

Basic network environment

We had a wireless router and Internet connection for all lab equipment during all our tests. This choice was mainly done for practical reasons. However, there are several operating system processes that most likely will make more noise when an Internet access is unavailable. ARP protocol tend to send a lot of broadcast messages, DNS and email services may also generate some noise. For instance, NMAP insists on contacting DNS on every startup. The software update processes frequently "call home" and checks for updated software. If an Internet connection is missing, these processes tends to increase their activity in order to connect to Internet.

In our graphs describing the isolated network environment bandwidth, we found UDP, IPv6, ARP and IGMP traffic.

4.5.3 IP Addresses

A common used network model is defining different network zones into; (i) trusted, (ii) untrusted, (iii) DMZ, etc. These zones have typical different IP address ranges. We wanted to place our test lab computers in trusted zone except the attacker and the background injecting computer.

We have changed our configuration regarding trusted and untrusted network so that the traffic the IDS's is are from an untrusted network. The setup is like this;

Trusted/local network = 192.168.1.0/25

Untrusted/external network is all that does not match the addresses above.

Each of our NIDS has two interfaces active; (i) local network for monitoring/management and (ii) TAP interface. Our TAP interface IP addresses are set in IP range 10.10.10.0/24.

4.5.4 Tuning Operating Systems & Interfaces

We wanted to avoid other security mechanisms to influence our tests. We further wanted to follow known good practice regarding IDS usage (and packet capturing).

We decided to apply the following configuration:
– The operating systems software firewalls were disabled.
– Both Bro & Snort TAP interfaces was set in promiscuous mode (PROMISC).

We found a lot of hints regarding further tuning of the TAP interface - however, we do not focus on large packets and/or payload in our tests. We did not experience any problem situation where further tuning of our TAP interface seemed necessary. Sources: [107–110]

4.5.5 Practical Problems

We did not suffer any technical challenges in our test lab. We managed once to fill up the hard disk drive on Snort server. Thanks to our checklist - we discovered this quickly restarted our tests.

4.5.6 Test Lab Summarized

- Operating systems = FreeBSD, Linux Mint 13 and Linux Debian
- Network Equipment = Fast Ethernet (100 Mb/s)
- IP Addresses = Trusted: 192.168.1.0/25 (All computers except Attacker, Background Injector and TAP interfaces). Untrusted (all except trusted)
- Tuning Operating Systems & Interfaces = Firewalls DISABLED, TAP interfaces in promiscuous mode (PROMISC)
- Practical Problems = one (we filled up a hard disk drive, but restarted our tests)

4.6 Tools

In this section we describe, discuss and decide what tools we need during our tests.

4.6.1 Simulating a Scan Attack

In order to simulate slow port scanning we needed a powerful and flexible tool. We found Network Mapper (NMAP or NMAP Security Scanner) suitable for this job. NMAP was originally written by Gordon “Fyodor” Lyon (Fyodor) [111]. NMAP is open source software maintained by Fyodor. NMAP has been widely used by network administrators since its birth in 1997 and is a major part of several books published [51, 112]. Our strongest reference in our literature study did also used NMAP [92].

We used the following command (the dollar sign ($) represents the command prompt):

```
1 $nmap --dns-servers 94.143.64.11 -sS --top-ports 100 -P0 -T0 --scan-delay 60s --max-scan
    -delay 61s 192.168.1.102 --packet-trace
```

NMAP options explained:

- --dns-servers = this option tells NMAP what DNS server to use. We got some error messaged without using this option.
- -sS = what scan to perform. In this case; S = SYN Scan.
- --top-ports 100 = performs a scan using the top 100 most used TCP. ports(port frequency) iwth the file nmap-services as the source.
- -P0 = threat all hosts as online. This option avoided NMAP to perform a so-called discovery process.
- -T0 = calm down scan process (slow and nice with one port at a time).
- --scan-delay 60s = scanning delay is minimum 60 seconds.
- --max-scan-delay 61s = scanning delay is maximum 61 seconds. The parameter -T0 may get NMAP to go much longer than 60 seconds.

- 192.168.1.102 = Victim Computer.
- --packet-trace = Packet trace. Shows all packets sent and received.

Our initial experience with NMAP gave us Table 9 regarding execution time of each scans.

Scan	Start Time (s)	Execution Time (s)
ACK Scan	109	6624
SYN Scan	109	6634
TCP Connect Scan	109	6654
NULL Scan	109	6745
FIN Scan	109	6745
XMAS Scan	109	6745

Table 9: NMAP execution time for different scan attacks.

This seems first a bit weird. We do instruct NMAP to scan 100 ports in an 60 seconds interval.

In addition to this, NMAP is not starting to scan until 133 secs. have passed B.2. The first 133 secs. NMAP is clearly getting familiar with its network environment.

However, as mentioned in Section 4.5.1, NMAP is an advanced software that use time analyzing its response. This NMAP behavior is by design.

4.6.2 Network Statistics

We needed a neutral tool that could give us network statistics on both Bro and Snort servers. This is part of our quality assurance (QA). We want to see that both servers get the same traffic data.

We chose Argus for this purpose. Argus is short for "Audit Record Generation and Usage System". Argus is open source initial written by Carter Bullard (released as public domain in 1996) [93]. Argus has strong references and is often used in partnership with IDS systems [92, 113, 114]. We have used Argus Version 3.0.6. Our strongest reference in our literature study also used Argus [92].We installed Argus on both Bro and Snort server. Argus gives us a lot of information, but the most needed in our context is the following:

- TCP statistics
- Packets per seconds (bandwidth)
- Searchable traffic statistics (e.g. by time stamps)

4.6.3 Background Traffic

It is crucial to test our improved Bro script outside of an isolated network.

Testing our improved Bro scan.bro script with live traffic?

We may have tested Bro in live network traffic, but this is challenging to perform (if possible) when we in general insist on being able to reconstruct and repeat the identical situation (reliability focus). This data will also contain much sensitive data classified under the privacy law [115]. We may add several machines in our test lab that uses several different applications in order to generate network background. This is not difficult, but demand a lot of computer equipment (and space). This is outside our scope, but interesting. We will perhaps perform this in another experiment.

Choosing a dataset

We initially planned to create our own dataset with network traffic data from our local ISP: LYSGLIMT (a brand under Austevoll Kraftlag SA) [1]. In computer network traffic there exists a lot of different kind of data. We will also in this situation be trapped This kind of data must be handled with care. This data will also contain much sensitive data classified under the privacy law [115]. If we got permission and decided

to capture live network traffic, we had to sanitize/anonymize this data before using it in our tests.

After considering the workload regarding anonymizing/sanitizing process including learning new tools, we started looking for ready made datasets. We needed a dataset created by professionals. Quality in dataset are an important factor.

We manage to use dataset from Cooperative Association for Internet Data Analysis (CAIDA) [116]. CAIDA is located at the University of California's San Diego Supercomputer Center. We chose to use a dataset captured 15 Aug 2013 named `quinix-chicago.dirA.20130815-134900.UTC.anon.pcap` [7].

The dataset key facts; (i) the dataset was captured without packet loss, (ii) the dataset do not contain any data/payload and (iii) the dataset is anonymized.

The CAIDA dataset is described in Appendix D.

Setting our parameters

Our experience with NMAP showed us that the scan process was not executed in 100 times 60 seconds (6000 sec). See our NMAP scan experience in Table 9. We wanted this time window to be covered by background traffic, and we decided to round upwards to 7000 sec.

Our other parameter we needed to set was the amount of data injected per time (bandwidth). Malmedal used in his thesis 1 Mb/s and we decided to follow his settings.

In order to simulate a server or several computers behind one IP address using network address translating mechanisms (NAT [117]) we wanted to direct the background traffic against victim computer.

We wanted to simulate network traffic with the following parameters:
– Generate traffic during 7000 sec
– Generate 1 Mbps traffic
– Direct background traffic against victim computer (IP address = 192.168.1.102)

We later ended up tuning our traffic injection to 1.2 Mb/s. This was done because we now was closer to the time windows of 7000 sec. The dataset chosen do not include any payload. In this manner the amount of TCP packets is much larger than live traffic with this kind of bandwidth. We tried to find research statistics regarding average TCP packets size without success.

4.6.4 Injecting Background Traffic

We needed a tool that could inject our dataset into our network. We decided to use `tcpreplay` to inject our dataset. This tool was able to support our parameter criteria in Section 4.6.3. `tcpreplay` is a commonly used tool for simulating and/or replaying network traffic with strong references [118–120].

Malmedal also used `tcpreplay` in his master thesis.

4.6.5 Reference NIDS

We wanted to include a reference IDS. Our choice was quite easy: Malmedal used Snort in his thesis.

We tuned Snort's `sfportscan` in order to detect our scan interval (60 sec). Snort is a de facto choice in open source NIDS. We did not activate/include any rules in our configuration. Our only focus was how `sfportscan` would cope with our test scenarios.

We decided to use the following `sfportscan` configuration:

```
preprocessor sfportscan:\
        proto  { tcp } \
        scan_type { all } \
        memcap { 10000000 } \
        logfile { /var/log/snort/sfportscan-alert.log } \
        sense_level { high } \
        detect_ack_scans
```

In order to get performance data we activated the `perfmonitor` preprocessor with this configuration:

```
preprocessor perfmonitor: \
        time 60 \
        file /var/log/snort/snort.stats \
        pktcnt 10000
```

4.6.6 Tools Summarized

- Simulating a scan attack = NMAP ("Network Mapper")
- Network Statistics = Argus
- Background Traffic = CAIDA Dataset from 15 Aug 2013
- Injecting Background traffic = `tcpreplay`
- Reference NIDS = Snort

5 Slow Port Scanning in Bro

"Until some brilliant researcher comes up with a better technique,
scan detection will boil down to testing for X events
of interest across a Y-sized time window."
– Stephen Northcutt

In this chapter we explain our efforts towards improving slow port scanning capability in Bro. Initially, we describe our strategy. We further explain Bro architecture in details and important configuration and log files. We further go deeper into how we managed to improve Bro slow scan detection capability.

5.1 Strategy for improving Bro slow port scan detection capability

In this section we explain our strategy in improving Bro slow port scan detection capabilities, and thus detection rate.

In order to detect slow port scans we may extend every time depending variable and in that manner be able to find scanners over a large time window. In our isolated research we may just do this and be satisfied with our results. However, in real world, unnecessary high use of CPU cycles and memory are rarely acceptable.

5.2 Initial test of Bro

We will in this section describe our initial approach regarding Bro slow port scan detection capabilities.

5.2.1 Bro's Port Scan detection = `Scan.bro`

When installing Bro 2.1, we do not get any ready made scan detection mechanism. We may of course create a new scan script from scratch. However, we found a script called `scan.bro` in Bro's download area `"bro-scripts"` [121]. This script was rather old. We considered for a while building a brand new one, but decided to use this as a base. This script has clearly evolved over many years with several authors. The header states that they will rewrite the script. Due to 7 Sept 2013, this is still the case for Bro version 2.1. Last official version of `scan.bro` is 3 Nov 2011. The `scan.bro` header is quoted below:

```
##! Scan detector ported from Bro 1.x.
##!
##! This script has evolved over many years and is quite a mess right now. We
##! have adapted it to work with Bro 2.x, but eventually Bro 2.x will
##! get its own rewritten and generalized scan detector.
```

The `scan.bro` script is in total 621 lines with 10 events, 19 functions and several variable declarations (constants, tables, sets, vectors, local variables).

Initial test with Bro using `scan.bro`

The original version of `scan.bro` contained a small error (see details in Appendix C.1).
The complete listing of the original `scan.bro` is found in Appendix C.4.

5.3 Increasing Log Level for scan.bro = Add Notices

We will in this section focus on increasing the log level when using `scan.bro`.

We need to increase the log level of Bro regarding scan detection. This is done because we want to know every detail of how `scan.bro` is working. This may flood our log files, but in the early stage of analyzing and understanding it is crucial to look closely/deep into every detail of this script.

Logging in Bro is done by notices. In order to generate new alerts (obtained from notices) we need to understand Bro's Notice Framework [122]. To increase the log level we need to modifying the local notice policy in Bro. We need to add (redefine as Bro cals it) notices in `scan.bro`.

In the following list we shows what notices the original `scan.bro` have:

5.3.1 The Notice Definition in original scan.bro

In this section we list notices in the original `scan.bro` script. The notices are placed in the first part of our `scan.bro`.

```
12 redef enum Notice::Type += {
13     PortScan,
14     AddressScan,
15     BackscatterSeen,
16     ScanSummary,
17     PortScanSummary,
18     LowPortScanSummary,
19     ShutdownThresh,
20     LowPortTrolling,
21   };
```

We have removed the comments in order to save space and make this list more readable. The complete `scan.bro` including comments is found in Appendix C.4.

The script code above gives the following notices a chance to appear in log file `notice.log` (given the right conditions); (i) PortScan, (ii) AddressScan, (iii) BackscatterSeen, (iv) ScanSummary, (v) PortScanSummary, (vi) LowPortScanSummary, (vii) ShutdownThresh and (viii) LowPortTrolling.

5.3.2 We need to generate more alerts in our analyzing process (more Notices)

In analyzing phase, we add several notices to see more details. We wanted to see when events in `scan.bro` script occurred/ was triggered. The notices that we created were the following:

```
33 redef enum Notice::Type += {
34       ConnectionPartial,
35       ConnectionAttempt,
36       ConnectionHalfFinished,
37       ConnectionRejected,
38       ConnectionReset,
39       ConnectionPending,
40   };
```

The script code above gives the following notices a chance to appear in log file `notice.log` (given the right conditions); (i) ConnectionPartial, (ii) ConnectionAttempt, (iii) ConnectionHalfFinished, (iv) ConnectionRejected, (v) ConnectionReset and (vi) ConnectionPending.

These notices are not active in our final modified script, but commented (disabled). The original `scan.bro` including comments is found in Appendix C.4, the modified `scan.bro` is found in Appendix C.5.

5.3.3 Structure of Notice.log

The notice.log file have the following structure:

```
1 #separator \x09
2 #set_separator   ,
3 #empty_field     (empty)
4 #unset_field     -
5 #path    notice
6 #open    2013-11-13-22-42-14
```

```
7 #fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
    proto note msg sub src dst p n peer_descr
    actions policy_items suppress_for droppperemote_location.country_code
    remote_location.region remote_location.city remote_location.latitude
    remote_location.longitude metric_index.host metric_index.str
    metric_index.network
8 #types time string addr port addr port enum enum string string
    addr addr port count string table[enum] table[count] interval
         bool string string string doubledouble addr string subnet
9 .
10 .
11 .
12 #close 2013-11-14-00-30-00
```

We have removed the actual logs (shown with dots(".")), and show here above only the header and the tail of this log file. We will now explain briefly how this file is formatted:

- The first five lines are fixed (defined by Bro developers)
- Line 6 (#open) show when this file was opened/created. The format including year, month, day, hour, minutes and seconds.
- Line 7 (#fields) shows the field headers. This is a long line (428 characters). However, the 13 first are the most interesting.
- Line 8 (#types) shows the field header variable types. E.g. time, string, addr, port etc.
- Line 12 (#close) show when this file was closed. This is most likely when Bro was stopped (or by other cron script that may rotate logfiles).

Now we were able to start testing slow port scanning against `scan.bro`.

Appendix C.7 show an example of `notice.log` with notices, and not just heading and tail. More details regarding this file format can be found on Bro Web page [123].

5.4 Variables that influence scan detection

In this section we will describe what variables we tuned in order to get `scan.bro` to detect slow port scanning.

5.4.1 Variables controls detection and reporting = 48

We found 48 variables that directly influence the detection and/or reporting of alerts regarding scan attacks. We only changed a few of these. A minor part of these variables are given in Table 10 (constants) and Table 11 (table of sets). The leftmost column refers to line number in the original `scan.bro` script. We denote these variables in the third column, depending on their role; Detection role (D), Reporting role (R) and Not Active (N/A). The rightmost column show if we changed these variables (denoted with "Yes"). These variables are of most interest. If the variable is changed, the new parameter(s) is included in the second column.

There are a close relationship between parameters that influences reporting capability and those that influence detection capability. If we activate a service for being detected, we need to ensure that this detection is reported, and visa verse.

We will further now describe the variables we changed and why we did this.

Variable: suppress_UDP_scan_checks (boolean)

According to our plan, we only focus on TCP protocol. Analyzing UDP protocol is therefore suppressed (`suppress_UDP_scan_checks` = T).

Variable: report_port_scan (vector of count)

The variable `report_port_scan` define when `scan.bro` send log alerts into `notice.log`. We added several new report thresholds in order to increase logging. This report was original set to start on reporting on 50. We changed it to report the following thresholds; 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 250, etc. This was important because our demands for more log details.

#	Constant Variables	Role	Changed
35	`const suppress_UDP_scan_checks = T &redef;`	D	Yes
37	`const activate_priv_port_check = T &redef;`	D	
38	`const activate_landmine_check = F &redef;`	D	
39	`const landmine_thresh_trigger = 5 &redef;`	N/A	
41	`const landmine_address: set[addr] &redef;`	N/A	
43	`const scan_summary_trigger = 25 &redef;`	R	
44	`const port_summary_trigger = 20 &redef;`	R	
45	`const lowport_summary_trigger = 10 &redef;`	R	
48	`const shut_down_thresh = 100 &redef;`	R	
52	`const analyze_services: set[port] &redef;`	D	
53	`const analyze_all_services = T &redef;`	D	
56	`const addr_scan_trigger = 0 &redef;`	D	
61	`const ignore_scanners_threshold = 0 &redef;`	D	
64	`const report_peer_scan: vector of count = { 20, 100, 1000, 10000, 50000, 100000, 250000, 500000, 1000000, } &redef;`	R	
68	`const report_outbound_peer_scan: vector of count = { 100, 1000, 10000, } &redef;`	R	
73	`const report_port_scan: vector of count = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 250, 1000, 5000, 10000, 25000, 65000, } &redef;`	R	Yes

Table 10: Interesting Constant Variables in Export area, original `scan.bro`. Role column; Detection(D), Reporting(R), Not Activated (N/A). Table 1 of 2.

Variable: distinct_ports (table of set)

The table `distinct_ports` collect host IP address and specific/distinct ports covering the whole port range (port 1-65535). This table deletes entries set by minutes. The original `scan.bro` had 15 minutes set, we changed it to collect scan attempts for 99 minutes. This may be a high number, but we do not focus on this kind of unnecessary memory usage in our thesis. We are more focused on catching slow port scanners.

Variable: distinct_low_ports (table of set)

The table `distinct_low_ports` collect host IP address and specific/distinct ports covering only system ports (port 1-1023). This table deletes entries set by minutes. The original `scan.bro` had 15 minutes set, we changed it to collect scan attempts for 99 minutes. This may be a high number, but we do not focus on this kind of unnecessary memory usage in our thesis. We are more focused on catching slow port scanners.

Variable: possible_scan_sources (table of set)

The table `distinct_ports` collect host IP address and specific/distinct ports classified as possible scanners. If a host found in `distinct_ports` or `distinct_low_ports` is found as a possible port scanner, it is placed in this table. This table deletes entries set by minutes. The original `scan.bro` had 15 minutes set, we changed it to collect scan attempts for 99 minutes. This may be a high number, but we do not focus on this kind of unnecessary memory usage in our thesis. We are more focused on catching slow port scanners.

The original parameter settings are found in the `scan.bro` script, Appendix C.4.

#	Sets, Tables and Tables and Sets	Role	Changed
163 164	`global pre_distinct_peers: table[addr] of set[addr] &read_expire = 15 mins &redef;`	D	
166 167	`global distinct_peers: table[addr] of set[addr] &read_expire = 15 mins &expire_func=scan_summary &redef;`	D	
169 170	`global distinct_ports: table[addr] of set[port] &read_expire = 99 mins &expire_func=port_summary &redef;`	D	Yes
176 177	`global distinct_low_ports: table[addr] of set[port] &read_expire = 99 mins &expire_func=lowport_summary &redef;`	D	Yes
185 186	`global possible_scan_sources: set[addr] &expire_func=remove_possible_source &read_expire = 99 mins;`	D	Yes

Table 11: Interesting Sets, Tables and Vector variables in Export area, original `scan.bro`. Role column; Detection(D), Reporting(R), Not Activated (N/A).

5.5 TCP Connection Events in Bro

In this section we describe Bro TCP connection events.

5.5.1 Describing TCP Connection events in Bro

We need to find out what connection events Bro TCP analyzer can offer.

The TCP analyzer in Bro (Bro::TCP), generates several events during analyzing. There exists seventeen events in total. The first ten are easy to use/implement without any second thoughts (normal events). However, the last seven are very low level semantic and CPU exhaustive. We may use the low level events of course, but not in normal situations. The Bro documentation uses the following description on these low level TCP events; (i) "should be rarely used", (ii) "should be avoided in normal situations" [124, 125].

Normal TCP Connection Events

Table 12 shows the normal TCP connection events. The descriptions are extracted from Bro `events.bif` file. We have tried to compress the text here without thwart the content.

Low Level TCP Connection Events

The following TCP connection events are very low level. We see that most of these events focus on TCP flags:

1. `connection_SYN_packet` – generated for a SYN packet.
2. `connection_first_ACK` – generated for the first ACK packet seen for a TCP connection from its originator.
3. `connection_EOF` – generated at the end of reassembled TCP connections.
4. `tcp_packet` – generated for every TCP packet.
5. `tcp_option` – generated for each option found in a TCP header.
6. `tcp_contents` – generated for each chunk of reassembled TCP payload.
7. `tcp_rexmit` – TODO (quoted from script). NOTE; we have not found any explanation on this low level event. In our context this is not important.

#	Event:	Description:
1	`new_connection_contents`	Generated when Bro starts to analyze a new TCP connection.
2	`connection_attempt`	Generated for an unsuccessful connection attempt.
3	`connection_established`	Generated when a SYN-ACK packet is seen in response to a SYN packet during a TCP handshake. The final ACK of the handshake in response to SYN-ACK may or may not occur later, one way to tell is to check the history field of :bro:type:‘connection‘ to see if the originator sent an ACK, indicated by ’A’ in the history string.
4	`partial_connection`	Generated for a new active TCP connection if Bro did not see the initial handshake.
5	`connection_partial_close`	Generated when a previously inactive endpoint attempts to close a TCP connection via a normal FIN handshake or an abort RST sequence. When the endpoint sent one of these packets, Bro waits :bro:id:‘tcp_partial_close_delay‘ prior to generating the event, to give the other endpoint a chance to close the connection normally.
6	`connection_finished`	Generated for a TCP connection that finished normally. The event is raised when a regular FIN handshake from both endpoints was observed.
7	`connection_half_finished`	Generated when one endpoint of a TCP connection attempted to gracefully close the connection, but the other endpoint is in the TCP_INACTIVE state.
8	`connection_rejected`	Generated for a rejected TCP connection. This event is raised when an originator attempted to setup a TCP connection but the responder replied with a RST packet denying it.
9	`connection_reset`	Generated when an endpoint aborted a TCP connection. The event is raised when one endpoint of an established TCP connection aborted by sending a RST packet.
10	`connection_pending`	Generated for each still-open TCP connection when Bro terminates.

Table 12: Connection Events generated by TCP Analyzer.

5.6 Analyzing scan.bro script regarding Connection Events

We will in this section go deeper into the `scan.bro` script in order to improve slow scan detection.

Now, with our change regarding timing and reporting in Section 5.4, we were able to detect slow scanning. However, still only SYN & TCP Connect scan attacks.

We will now explain two important mechanisms that `scan.bro` make use of; (i) connection (endpoint) states and (ii) the history field in connection records.

5.6.1 Connection Endpoint State

In this paragraph we describe the so-called *endpoint states*. These states tells us how the TCP endpoint (the destination) of an actual TCP connection is responding.

These endpoint states is used in the original `scan.bro`. The following list shows important connection states found in Bro file `$BROHOME/share/base/init-bare.bro` [126]. Source: [126].

```
const TCP_INACTIVE =     0;      ##< Endpoint is still inactive.
const TCP_SYN_SENT =     1;      ##< Endpoint has sent SYN.
const TCP_SYN_ACK_SENT = 2;      ##< Endpoint has sent SYN/ACK.
```

```
const TCP_PARTIAL =      3;      ##< Endpoint has sent data but no initial SYN.
const TCP_ESTABLISHED =  4;      ##< Endpoint has finished initial handshake regularly.
const TCP_CLOSED =       5;      ##< Endpoint has closed connection.
const TCP_RESET =        6;      ##< Endpoint has sent RST.
```

A typical use of these connection endpoint states looks like this:

```
1 if c$orig$state == TCP_CLOSED
```

5.6.2 Connection Record, History State

We will now explain another important variable: the *history* variable in the *connection record*. The complete definition of the connection record is shown in Table 13. We found in Bro documentation (Web page) that the history variable had information regarding the TCP flag. This was very interesting.

connection		
Type:	Record	
	id: conn_id	The connections identifying 4-tuple.
	orig: endpoint	Statistics about originator side.
	resp: endpoint	Statistics about responder side.
	start_time: time	The timestamp of the connections first packet.
	duration: interval	The duration of the conversation. Roughly speaking, this is the interval between first and last data packet (low-level TCP details may adjust it somewhat in ambiguous cases).
	service: set [string]	The set of services the connection is using as determined by Bro's dynamic protocol detection. Each entry is the label of an analyzer that confirmed that it could parse the connection payload. While typically, there will be at most one entry for each connection, in principle it is possible that more than one protocol analyzer is able to parse the same data. If so, all will be recorded. Also note that the recorded services are independent of any transport-level protocols.
	addl: string	Deprecated.
	hot: count	Deprecated.
	history: string	State history of connections. See history in Conn::Info.
	uid: string	A globally unique connection identifier. For each connection, Bro creates an ID that is very likely unique across independent Bro runs. These IDs can thus be used to tag and locate information associated with that connection.
	tunnel: EncapsulatingConnVector &optional	If the connection is tunneled, this field contains information about the encapsulating "connection(s)" with the outermost one starting at index zero. Its also always the first such encapsulation seen for the connection unless the tunnel_changed event is handled and re-assigns this field to the new encapsulation.
A connection. This is Bro's basic connection type describing IP- and transport-layer information about the conversation. Note that Bro uses a liberal interpretation of "connection" and associates instances of this type also with UDP and ICMP flows.		

Table 13: Connection Record.

This `history` record is set by the TCP analyzing process, and tell us the state history of the actual connection. The `history` record is a string containing letters that tells us the following about a TCP packet; (i) flags state, (ii) payload, (iii) bad checksum and (iv) inconsistent TCP flag behavior. See Table 14details. Source: [127]

Conn::Info		
Type:	Record	
	history	string &log &optional Records the state history of connections as a string of letters. The meaning of those letters is: **Letter Meaning** s – a SYN w/o the ACK bit set h – a SYN+ACK (“handshake”) a – a pure ACK d – packet with payload (“data”) f – packet with FIN bit set r – packet with RST bit set c – packet with a bad checksum i – inconsistent packet (e.g. SYN+RST bits both set) If the event comes from the originator, the letter is in upper-case; if it comes from the responder, its in lower-case. Multiple packets of the same type will only be noted once (e.g. we only record one “d” in each direction, regardless of how many data packets were seen.)

Table 14: Connection Information (Conn::Info): the history record.

5.7 Modifying `scan.bro`

We will in this section implement our findings from previous section and thus improve the slow port scan capability in `scan.bro`.

Adding Event `new_connection_contents`

We manage to add one of the unused events found in Bro documentation. Note that we check both directions (both c$orig$state and c$resp$state). This is done because Bro may be confused by the wrongly used TCP flag. This way of checking both ways is also used in original `scan.bro`.

The improvement in adding event `new_connection_contents` was initial intended to detect FIN Scan attacks. However, XMAS Scan attacks also use the FIN flag. We strongly believe that this modifications was the strongest element in improving `scan.bro` regarding better detection of slow port scan.

```
678 event new_connection_contents(c: connection)
679   {
680   local is_reverse_scan = (c$resp$state == TCP_CLOSED );
681   if (( c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED ) &&
682     ( "f" in c$history || "F" in c$history ))
683     {
684     Scan::check_scan(c, F, is_reverse_scan);
685     }
686   else if (( c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED ) &&
687     ( "i" in c$history || "I" in c$history ))
688     {
689     Scan::check_scan(c, F, is_reverse_scan);
690     }
691   }
```

Modifying Event `partial_connection`

We modified event `partial_connection` in order to better detect partial TCP connection.

```
732 event partial_connection(c: connection)
733         {
734         local is_reverse_scan = (c$resp$state == TCP_PARTIAL );
735
736         if (c$orig$state == TCP_PARTIAL || c$resp$state == TCP_PARTIAL )
737                 {
738                 Scan::check_scan(c, F, is_reverse_scan);
739                 }
740         else
741                 {
742                 Scan::check_scan(c, T, is_reverse_scan);
743                 }
744         }
```

Modifying Event `connection_rejected`

We modified event `connection_rejected` in order to better detect partial TCP connection, and both FIN & XMAS Scan attacks.

```
800 event connection_rejected(c: connection)
801         {
802         local is_reverse_scan = c$orig$state == TCP_RESET;
803
804         if (c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED)
805                 {
806                 Scan::check_scan(c, F, F);
807                 }
808         else
809                 {
810                 Scan::check_scan(c, F, is_reverse_scan);
811                 }
812         }
```

5.8 Summary

We have in this chapter explained our process of improving Bro's slow port scan detection capabilities. We increased timeout for some variables, added an unused connection event and modified some other connection events.

We strongly believe that we are able to detect the following scan attacks:
– FIN scan (checking for TCP packets that is answered with RST)
– XMAS & NULL scan (checking for inconsistent/odd use of TCP flags).

6 Experimental Setup and Results

"The true method of knowledge is experiment."
– William Blake

In this chapter we describe our experimental setup and what results it gave us. We apply methods described in Chapter 4 with our improved Bro policy script found in Chapter 5.

6.1 Lab Setup

In this section we shows the details regarding our experimental test lab.

6.1.1 Network Diagram of our Test Lab

We have created an illustration showing our physical network connections. The thin lines and the thick do not symbol any different capabilities. All lines are all at the same speed: 100 MB/s (fast ethernet). We had this network connected to Internet with a wireless router which is not included in this illustration. Our network diagram is presented in fig. 5.

6.1.2 Equipment Details

We have created a list of all equipment we used in our tests. This list is given in Table 15 and 16.

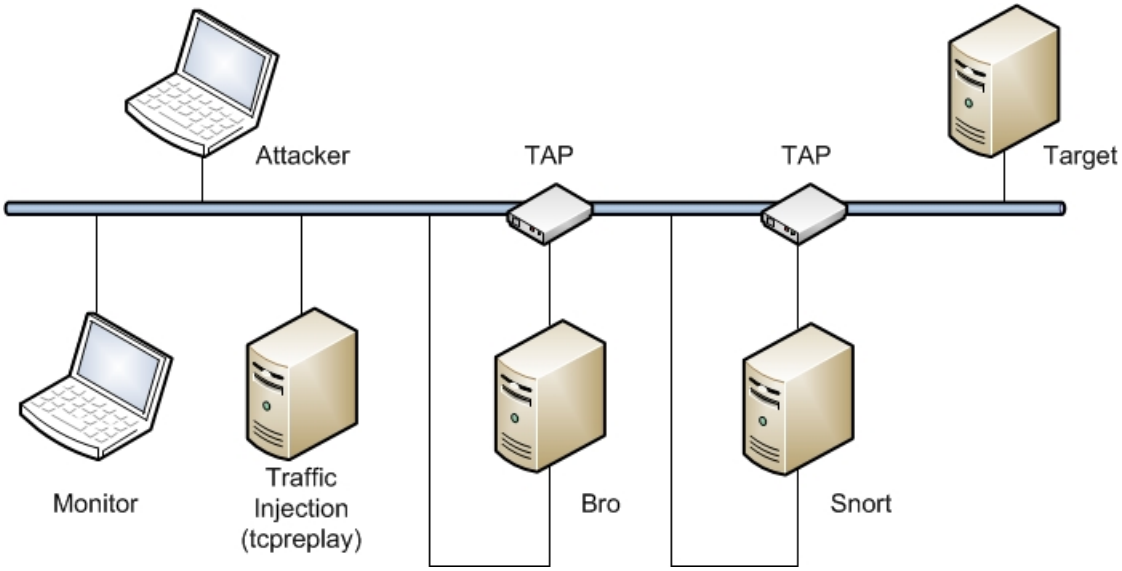

Figure 5: Test Lab Setup.

6.2 Bro in Practice

We will in this section describe how we got Bro IDS up and running. We will also describe where we find alerts that Bro generates (log files).

Role	Issue	Details	IP Address
Attacker	Hardware:	IBM ThinkPad T60	
	Operating System:	Linux Mint 13 Maya	
	Application:	NMAP 6.25	
	Networking:	wired interface eth0	192.168.1.200/24
Monitor	Hardware:	Apple Mac Mini	
	Operating System:	OS X 10.8	
	Application:	telnet, ftp, ssh etc.	
	Networking:	wired interface en0	192.168.1.140/24
Traffic Injecting	Hardware:	DELL Latitude D530	
	Operating System:	Linux Debian 7.2 Xfce	
	Application:	tcpreplay version: 3.4.3 (build 2375)	
		tcprewrite version: 3.4.3 (build 2375)	
	Networking:	wired interface eth0	192.168.1.201/24
Bro Server	Hardware:	Dell OptiPlex GX 260	
	Operating System:	FreeBSD 9.1	
	Application:	Bro version: 2.1-328	
		Argus 3.06	
	Networking:	wired interface rl0	192.168.1.123/24
		wired interface rl1	10.10.10.10/24
Snort Server	Hardware:	IBM ThinkPad T42	
	Operating System:	Linux Debian 7.2 Xfce	
	Application:	Snort version: 2.9.5.5 GRE (Build 205)	
		Argus 3.06	
	Networking:	wired interface eth0	192.168.1.141/24
		wireless interface wlan1 (external USB)	10.10.10.77/24
Target	Hardware:	ASUS Aspire One	
	Operating System:	Linux Mint 13 Maya	
	Application:	sshd	
	Networking:	wired interface eth0	192.168.1.102/24

Table 15: Test Lab Setup Details. Computer and Servers.

Role	Issue	Details	IP Address
TAP #1	Hardware:	Alcatel OmniSwitch 6224	
	Operating System:	Alcatel OS	
TAP #2	Hardware:	NetGear ProSafe 5-ports switch GS105E	
	Operating System:	unknown	
Router (Internet)	Hardware:	Cisco Linksys E4200	192.168.1.1/24
	Operating System:	unknown	
Switch	Hardware:	3Com 5-ports model: 3CSFU05	
	Operating System:	unknown	

Table 16: Test Lab Setup Details. Physical Network Infrastructure.

6.2.1 Getting Bro up and running

Bro is not a plug and play software. In general, no NIDS software works right "out of the box". This is because of the complexity of the software and the great variations/uniqueness in every organizations network environment. We may invest in an NIDS system that is ready installed with perhaps specially tuned hardware and management software. However, this kind of so-called appliance box will also need

a lot of initial tuning regarding the network environment it is installed into and of course the level of logging (alarm thresholds). In addition, NIDS will need some attention for every change in the network environment, which in our computerized world is more or less constant.

Local Site Policy

Bro was installed in directory: `/usr/local/bro/ ($BROHOME)`. Bro's complete directory structure can be found in Appendix C.3. Bro need some basic input after initial installation is done. We need to update the so-called *"Local Site Policy"* [128]. This is placed in the configuration file: `local.bro` located in directory
`/$BROHOME/share/bro/site/`. This file tells Bro what scripts to load during startup, what the IP addresses of the local network is etc. We did the following change to `local.bro` file regarding our network environment:

```
76 redef Site::local_nets = {
77 #    10.0.0.0/8,             # Untrusted IP range
78        192.168.1.0/25,       # Trusted IP range
79 };
```

Comments to configuration lines above:
Line #76 show us that this is a redefining (`redef`) of site variable `Site::local_net`.
Line #77 is commented out (initial # most left in line) and will only work as informational purpose. We may typical enter our untrusted IP address range here.
Line #78 tells Bro that IP address range 192.168.1.0./25 is local and trusted.
Line #79 ends the declaration of the variable mentioned above.

The complete `local.bro` is listed in Appendix C.2.

In order to use the script `scan.bro`, we copied `scan.bro` file into our Bro file structure. Bro's policy scripts are all placed in sub directories under directory:
`$BROHOME/share/bro/base/policy`. We placed `scan.bro` in sub directory `misc` (complete path: `$BROHOME/share/bro/base/policy/misc`). In order to tell Bro to use (load) the script `scan.bro` at startup, we added the following line to file `local.bro` :

```
72 #       Other scripts
73 @load policy/misc/scan
```

We used the following command in order to run Bro with our configuration file local.bro:

```
$bro -i rl1 local.bro
```

The option `"-i"` instructs Bro what interface to listen to.

6.2.2 Logfile: `notice.log`

When we ran our tests, we always wanted these kind of results in the log file `notice.log`. This states that Bro was able to detect the scan attack:

```
1 1384385400.480270      -       -       -       -       -       -       Scan::
    LowPortScanSummary          192.168.1.200 scanned a total of 94 low ports   -
    192.168.1.200   -       -       94      bro     Notice::ACTION_LOG      6
    3600.000000     F       -       -       -       -       -       -       -       -
```

6.3 Simulating Scanning using NMAP

This section shows some output of the command we used when simulating a scan attack.

Source: [51]

The following list show the first 20 lines of the NMAP command output:

```
1  Starting Nmap 6.25 ( http://nmap.org ) at 2013-11-19 09:25 CET
2  SENT (60.1111s) ARP who-has 192.168.1.102 tell 192.168.1.200
3  RCVD (60.1114s) ARP reply 192.168.1.102 is-at 00:23:5A:62:1B:35
4  NSOCK (60.1130s) nsi_new (IOD #1)
5  NSOCK (60.1130s) UDP connection requested to 95.143.64.11:53 (IOD #1) EID 8
6  NSOCK (60.1130s) Read request from IOD #1 [95.143.64.11:53] (timeout: -1ms) EID 18
7  NSOCK (60.1130s) Write request for 44 bytes to IOD #1 EID 27 [95.143.64.11:53]:
       -...........102.1.168.192.in-addr.arpa.....
8  NSOCK (60.1130s) Callback: CONNECT SUCCESS for EID 8 [95.143.64.11:53]
9  NSOCK (60.1130s) Callback: WRITE SUCCESS for EID 27 [95.143.64.11:53]
10 NSOCK (64.1150s) Write request for 44 bytes to IOD #1 EID 35 [95.143.64.11:53]:
       -...........102.1.168.192.in-addr.arpa.....
11 NSOCK (64.1150s) Callback: WRITE SUCCESS for EID 35 [95.143.64.11:53]
12 NSOCK (68.1170s) Write request for 44 bytes to IOD #1 EID 43 [95.143.64.11:53]:
       -...........102.1.168.192.in-addr.arpa.....
13 NSOCK (68.1170s) Callback: WRITE SUCCESS for EID 43 [95.143.64.11:53]
14 NSOCK (73.1180s) nsi_delete (IOD #1)
15 NSOCK (73.1180s) msevent_cancel on event #18 (type READ)
16 SENT (133.2189s) TCP 192.168.1.200:42392 > 192.168.1.102:110 S ttl=57 id=58115 iplen=44
       seq=4148668469 win=1024 <mss 1460>
17 RCVD (133.2191s) TCP 192.168.1.102:110 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
18 SENT (193.3190s) TCP 192.168.1.200:42392 > 192.168.1.102:23 S ttl=52 id=43087 iplen=44
      seq=4148668469 win=1024 <mss 1460>
19 RCVD (193.3194s) TCP 192.168.1.102:23 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
      =0 win=0
20 SENT (253.4191s) TCP 192.168.1.200:42392 > 192.168.1.102:995 S ttl=49 id=56741 iplen=44
       seq=4148668469 win=1024 <mss 1460>
```

The following list show the last 13 lines of the NMAP command output:

```
1  SENT (6563.8193s) TCP 192.168.1.200:42392 > 192.168.1.102:888 S ttl=57 id=51731 iplen=44
       seq=4148668469 win=1024 <mss 1460>
2  RCVD (6563.8198s) TCP 192.168.1.102:888 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
3  SENT (6623.9194s) TCP 192.168.1.200:42392 > 192.168.1.102:4 S ttl=47 id=4909 iplen=44
      seq=4148668469 win=1024 <mss 1460>
4  RCVD (6623.9199s) TCP 192.168.1.102:4 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
      =0 win=0
5  Nmap scan report for 192.168.1.102
6  Host is up (0.00034s latency).
7  Not shown: 99 closed ports
8  PORT    STATE SERVICE
9  22/tcp open  ssh
10 MAC Address: 00:23:5A:62:1B:35 (Compal Information (kunshan) CO.)
11
12 Nmap done: 1 IP address (1 host up) scanned in 6623.94 seconds
13 Nmap STOP: 2013-11-19-11:15:24 ## S Scan ##
```

6.4 Injecting Traffic to Simulate Background Traffic

In this section we describe the CAIDA dataset. We will also show our actions in order to prepare the dataset for our network environment.

6.4.1 CAIDA Dataset Statistics

The following table shows statistics regarding the CAIDA dataset:
`equinix-chicago.dirA.20130815-134900.UTC.anon.pcap`.

6.4.2 Preparing the CAIDA dataset

The dataset from CAIDA had to be prepared for our needs. The dataset is in PCAP fileformat [129]. In order to manipulate the dataset we use the command line tool `tcprewrite`, which was included in the installation of `tcpreplay` [118, 130].

Description	Number/Timestamp
Maximum capture length for interface 0:	65536
First timestamp:	1376574540.000000000
Last timestamp:	1376574599.999998000
Unknown encapsulation:	0
IPv4 bytes:	30210371534
IPv4 pkts:	34455742
IPv4 flows:	1432751
Unique IPv4 addresses:	766294
Unique IPv4 source addresses:	390990
Unique IPv4 destination addresses:	376431
Unique IPv4 TCP source ports:	61616
Unique IPv4 TCP destination ports:	61117
Unique IPv4 UDP source ports:	61855
Unique IPv4 UDP destination ports:	62080
Unique IPv4 ICMP type/codes:	15
IPv6 pkts:	11233
IPv6 bytes:	2516917
non-IP protocols:	0
non-IP pkts:	0

Table 17: Statistics for CAIDA dataset `equinix-chicago.dirA.20130815-134900.UTC.anon.pcap` [7].

Preparing the dataset for ethernet network

The original dataset is captured using RAW format. The following command rewrites/converts this into ethernet format:

```
1 tcprewrite --dlt=enet --enet-smac=00:11:22:33:44:55 --enet-dmac=66:77:88:99:aa:bb --
      infile=equinix-chicago.dirA.20130815-134900.UTC.anon.pcap --outfile=equinix-chicago.
      dirA.20130815-134900.UTC.anon_2.pcap
```

The options used are explained in the listing below.

`--dlt=str` convert to the correct Layer 2 (in our case; enet (ethernet))

`--enet-smac=str` set the source MAC address

`--enet-dmac=str` set the destination MAC address

`--infile=str` self explaining

`--outfile=str` self explaining

Preparing the dataset for our IP addresses

The original dataset is captured using a lot of other IP address than in our privat IP address environment. The following command rewrites/converts the source and destination IP addresses. In this command we also recalculates the layer 2 checksums:

```
1 tcprewrite --fixcsum --dstipmap=10.0.0.0/16:192.168.1.102/32 --infile=equinix-chicago.
      dirA.20130815-134900.UTC.anon_.pcap --outfile=equinix-chicago.dirA.20130815-134900.
      UTC.anon_3.pcap
```

Please find the options used explained in listing below.

`--fixsum` forces recalculation of IPv4/TCP/UDP header checksum

`--loop=num` set the replay to loop. Will stop when Ctrl+C is pressed

`--infile=str` self explaining

`--outfile=str` self explaining

Replaying the dataset

We replayed the dataset using `tcpreplay` [118].

```
1 tcpreplay -i eth0 -M 1.2 equinix-chicago.dirA.20130815-134900.UTC.anon_2.pcap
```

Please find the options used explained in listing below.

`-i` set the traffic output interface

`-M` set the traffic output rate in Mbps

6.4.3 Statistics from our test

By using data collected from Argus, we are able to show nice bandwidth graphs. We have used the CLI programs `ragraph` & `racount` Figure 9 shows the traffic during a scan where we have the isolated network.

First we use Argus's `ragraph` with option `proto` in order to get protocol statistics. We used the following command ($ is representing the command prompt):

```
$racount -M proto -r argus.out_2013-10-15-0729 -t 2013/10/14.23:03-2013/10/15.00:58
```

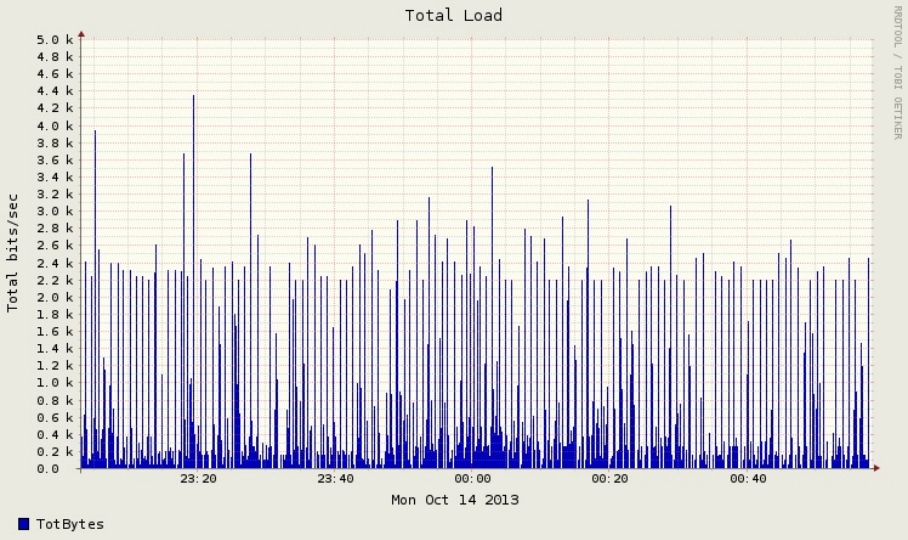

Figure 6: Graph showing bandwidth during scan attack in isolated traffic environment.

First we use the option `proto` in order to get protocol statistics. We used the following command ($ is representing the command prompt):

```
$racount -M proto -r argus.out_2013-10-15-0729 -t 2013/10/14.23:03-2013/10/15.00:58
```

Secondly we use the option `address` in order to get IP address summary statistics. We used the following command ($ is representing the command prompt):

```
$racount -M address -r argus.out_2013-10-15-0729 -t 2013/10/14.23:03-2013/10/15.00:58
```

Figure 9 shows the traffic during a scan test when we inject background traffic.

Tcpreplay: [118, 131, 132].

racount	records	total_pkts	src_pkts	dst_pkts	total_bytes	src_bytes
sum	10416	3591	3081	510	580584	522617
Protocol	Summary					
icmp	2	2	2	0	496	496
igmp	110	275	275	0	16500	16500
tcp	117	260	137	123	18436	9029
udp	1624	1845	1692	153	464865	430345
udp	1	2	2	0	266	266
ip	46	83	83	0	9065	9065
ipv6-icm	41	69	69	0	5934	5934
udp	10	10	10	0	2322	2322
llc	16	17	17	0	1020	1020
arp	443	798	564	234	47880	33840

Figure 7: Focus on protocols. Statistics created by use of Argus: `racount`. Scan sequence during isolated network environment.

racount		records	total_pkts	src_pkts	dst_pkts	total_bytes	src_bytes	dst_bytes
sum		10416	3591	3081	510	580584	522617	57967
Address	Summary							
IPv4	Unicast	src		0 dst		1		
IPv4	Unicast This Network	src		1 dst		0		
IPv4	Unicast Private	src		7 dst		6		
IPv6	LinkLocal	src		98 dst		0		
IPv6	Multicast Link Local	src		0 dst		98		

Figure 8: Focus on address. Statistics created by use of Argus: `racount`. Scan sequence during isolated network environment

The following output is generated from the same Argus statistics file found in figure 9. The output is generated by use of the Argus client program `racount`.

First we use the option `proto` in order to get protocol statistics. We used the following command ($ is representing the command prompt):

```
$racount -M proto -r argus.out_2013-10-27-1836
```

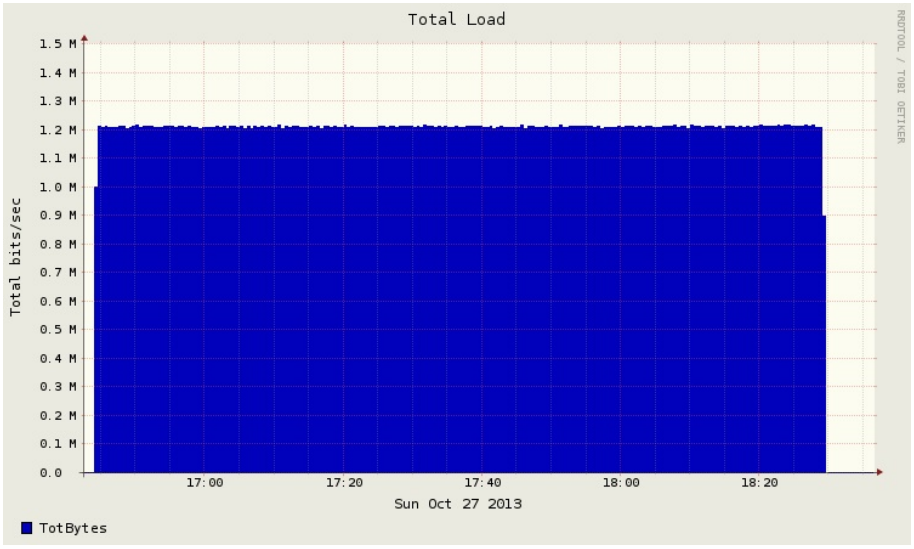

Figure 9: Graph showing bandwidth during scan attack in background traffic environment.

racount	records	total_pkts	src_pkts	dst_pkts	total_bytes	src_bytes	dst_bytes
sum	3777976	14030799	14028193	2606	899631701	899422337	209364
Protocol	Summary						
ip	2448	3424	3424	0	242528	242528	0
icmp	21711	29199	29023	176	1751940	1741380	10560
igmp	112	280	280	0	16800	16800	0
tcp	3113420	12063564	12061320	2244	781150251	780982285	167966
udp	629786	1883376	1883257	119	113360942	113334124	26818
ipv6	1890	2207	2207	0	170262	170262	0
ip	14	27	27	0	8667	8667	0
gre	3662	23852	23852	0	1431120	1431120	0
esp	4166	24002	24002	0	1440120	1440120	0
udp	4	6	6	0	666	666	0
ip	72	132	132	0	14353	14353	0
udp	3	3	3	0	432	432	0
llc	16	16	16	0	960	960	0
arp	318	476	409	67	28560	24540	4020

Figure 10: Focus on protocols. Statistics created by use of Argus: `racount`. Scan sequence during background network environment.

Secondly we use the option `address` in order to get IP address summary statistics. We used the following command ($ is representing the command prompt):

```
$racount -M address -r argus.out_2013-10-27-1836
```

racount	records	total_pkts	src_pkts	dst_pkts	total_bytes
sum	3777976	14030799	14028193	2606	899631701
Address	Summary				
IPv4	Unicast	src	89525	dst	128
IPv4	Unicast This Network	src	29	dst	0
IPv4	Unicast Private	src	241	dst	3
IPv4	Unicast Reserved	src	30543	dst	119
IPv4	Multicast Reserved	src	2887	dst	0
IPv4	Multicast Src Spec	src	1784	dst	28
IPv6	Unspecified	src	2448	dst	2446
IPv6	LinkLocal	src	93	dst	0
IPv6	Multicast Link Local	src	0	dst	95

Figure 11: Focus on address. Statistics created by use of Argus: `racount`. Scan sequence during background network environment.

6.5 Snort

We will in this section show the output from pre-processor: `sfportscan`.

When Snort detected our slow port scanning, we got output in log file: `/var/log/snort/sfportscan.log`.

The following list shows an example of this output:

```
Time: 11/18/13-12:55:13.860208
event_ref: 0
192.168.1.200 -> 192.168.1.102 (portscan) TCP Portscan
Priority Count: 9
Connection Count: 10
IP Count: 1
Scanner IP Range: 192.168.1.200:192.168.1.200
Port/Proto Count: 10
Port/Proto Range: 24:1022

Time: 11/18/13-13:06:13.867026
event_ref: 0
192.168.1.200 -> 192.168.1.102 (portscan) TCP Portscan
Priority Count: 9
Connection Count: 10
IP Count: 1
Scanner IP Range: 192.168.1.200:192.168.1.200
Port/Proto Count: 10
Port/Proto Range: 17:880

Time: 11/18/13-13:17:13.873567
event_ref: 0
192.168.1.200 -> 192.168.1.102 (portscan) TCP Portscan
Priority Count: 9
Connection Count: 10
IP Count: 1
Scanner IP Range: 192.168.1.200:192.168.1.200
Port/Proto Count: 10
Port/Proto Range: 3:1000
```

6.6 Results

We will in this section present our test results. We managed to get 10 rounds for each scan our two cases; (i) isolated and (ii) background traffic injected.

We manage to improve Bro slow port scan detection capabilities by our modified `scan.bro` script. The results are shown in Table 18.

Attack	Bro - Original Script	Bro - Modified Script	Snort
ACK Scan	Not Detected	Not Detected	Not Detected
SYN Scan	Detected	Detected	Detected
TCP Connect Scan	Detected	Detected	Detected
NULL Scan	Not Detected	Not Detected	Not Detected
FIN Scan	Not Detected	Detected	Not Detected
XMAS Scan	Not Detected	Detected	Not Detected

Table 18: Test results. Isolated and Background Traffic environment (identical results).

When using the original `scan.bro` script we managed to detect two of six scans. The TPR (True Positive Rate (often called the detection rate)):

$$TPR = \frac{TP}{TP+FN} = \frac{2}{6} = \frac{1}{3} = 0,33$$

When using our modified `scan.bro` script we managed to detect four of six scans. The TPR:

$$TPR = \frac{TP}{TP+FN} = \frac{4}{6} = \frac{2}{3} = 0,66$$

7 Discussion

"If you really struggle with a scientific problem: try talk to yourself!
This will most likely scare you, but also give you some new ideas."
– Roger Larsen

In this chapter we discuss our master thesis. We need to be our own critical thinker and ask all the questions that begins with "Why".

We managed to improve the slow port scan detection capabilities in Bro by modifying `scan.bro` policy script.

7.1 Test Lab Experience

In this section we discuss our test lab experience.

7.1.1 Bro Log without IP address

During background traffic injection, Bro produced log that misses an IP address. An example of this log output is listed here:

```
1  #separator \x09
2  #set_separator   ,
3  #empty_field     (empty)
4  #unset_field     -
5  #path    notice
6  #open    2013-11-24-09-58-04
7  #fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
       proto   note    msg     sub     src     dst     p       n       peer_descr
       actions policy_items    suppress_for    dropperemote_location.country_code
       remote_location.region  remote_location.city    remote_location.latitude
       remote_location.longitude       metric_index.host       metric_index.str
       metric_index.network
8  #types  time    string  addr    port    addr    port    enum    enum    string  string
       addr    addr    port    count   string  table[enum]     table[count]    interval
              bool    string  string  string  doubledouble   addr    string  subnet
9  1385283484.875938       -       -       -       -       -       tcp     Scan::PortScan
       :: has scanned 20 ports of ::   -       ::      -       179     20      bro
       Notice::ACTION_LOG      6       3600.000000     -
10 1385284075.140645       -       -       -       -       -       tcp     Scan::PortScan
       :: has scanned 30 ports of ::   -       ::      -       60909   30      bro
       Notice::ACTION_LOG      6       3600.000000     -
11 1385284200.638245       -       -       -       -       -       -       Scan::
       PortScanSummary    :: scanned a total of 41 ports -       ::      -       -       41
              bro      Notice::ACTION_LOG      6       3600.000000     F       -       -
              -       -       -       -       -       -
12 #close   2013-11-24-10-10-00
```

This output came for all tests were we injected background traffic, all six scans and both original and modified `scan.bro` scripts. An output were our normal scan detection log is listed in Appendix C.8.

We used many hours in search for an answer to this empty log, without getting it. We found no specific tracks in the complete log file `conn.log`. There percentage of IP protocol (IPv6 [133]) is very low in CAIDA dataset17. There are 11233 IPv4 packets out of total 34455742 packets both IPv4 & IPv6.). Calculated in % we get the number: 0,03%. Bro supports IPv6, but the log output did not reveal any use of IPv6.

In retrospect we should perhaps had disabled IPv6 support on all ethernet interfaces in our lab. Another solution to this log output may also be error(s) in the `scan.bro` script and/or Bro software.

7.1.2 Error messages from Bro

We found no errors and only one warning message during our tests. This output came for all tests were we injected background traffic, all six scans and both original and modified `scan.bro` scripts. The warning is listed below:

```
1 1385332500.489219 warning in /usr/local/bro/share/bro/base/misc/find-checksum-offloading
    .bro, line 42: Your interface is likely receiving invalid TCP checksums, most likely
     from NIC checksum offloading.
```

This Bro warning message tells us that one or more TCP packets had "most likely" invalid TCP checksums. We prepared our dataset for our network environment by using the tool `tcprewrite` (see Section 6.4). Was this preparing process enough? Should we used other tools to double check our results? We have strong references to `tcpreplay` package (were `tcprewrite` is included) [118–120]. In retrospect, we should perhaps have used an other injecting tool (for reference) during our troubleshooting process.

7.1.3 Isolated network with Internet access

We defined our network environment as isolated during the experimental tests (when we did not inject background traffic). We do not completely managed to reach the level of "isolation" with a router connected to Internet in the same network in addition to other wireless clients.

Statistics in Chapter 6.4.3 shows the level of traffic in per definition "isolated". However, the amount of this noise is quite small. We did not experience that our isolated network environment was challenged by more than this amount of traffic we see in the graphs mentioned.

In retrospect, we should perhaps have considered a more isolated network that in larger degree supports the term "isolated". This completely isolated network should perhaps have been without both Internet access and other clients.

7.2 Evaluating our results

In this section we discuss our modifications of `scan.bro`.

7.2.1 We managed to detect two new scans with our improved script

We managed to add detection of both FIN and XMAS scan in our modification shown in Chapter 5.7. We compare in our modification FIN flag against the endpoint response (TCP_RESET). XMAS scan is using FIN flag in the TCP packet (in addition to other flags). We strongly believe that this is the reason for our improved slow port scan detection capabilities in Bro.

7.2.2 Why did we not manage to detect an ACK Scan

We did not manage to detect an ACK Scan during our experimental tests. We had initially some script code that we were sure could detect ACK scan, but it did not. This puzzled us a bit. Our explanation to this may be that this TCP flag is a very commonly used flag.

Table 3 in Section 2.1.2 show an normal TCP session with respect to TCP flag usage. We see the ACK flag is most frequent except PSH (push, which tells the receiver that it contains data/payload).

Appendix A show the so-called "TCP State Machine". This is an illustration that show all possible states in TCP communication with the respect to TCP flag.

We have in Table 19 gathered and ranked our six scan attacks according to NMAP documentation [8, 51]. Bear in mind that this ranking is only in context with TCP port scanning. NMAP community have implemented many scans that has their uniqueness.

Efficiency	Scan	NMAP Comment
1	SYN	high quality answer
2	TCP Connect	high quality answer, may give logs on victim server
3	FIN, NULL, XMAS	medium quality answer (struggle to detect if ports are open or closed)
4	ACK	low quality answer (will never detect if ports are open or closed)

Table 19: Our different scans and their efficiency according to NMAP [8].

We conclude our lack of detecting ACK scans is because of this flag setting is very commonly seen.

7.2.3 Why did we not manage to detect a NULL Scan

We did not manage to detect a NULL scan in our modified script. We modified the `scan.bro` script and compare the history variable for "i" (or "I", opposite direction checked) with the following code:

```
1 if (( c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED ) && ( "i" in c$history
    || "I" in c$history ))
```

Bro documentation explain an "i" in history like this:

```
i - inconsistent packet (e.g. SYN+RST bits both set)
```

We could not find better/other explanation to this expression "inconsistent packet" in Bro documentation. In our definition of "inconsistent packet" would both NULL & XMAS scan be detected using this script code. However, there may be underlying definitions in Bro software why we did not manage to detect both these scans. There may also be error(s) in Bro software and/or TCP analyzer script.

7.2.4 Other Comments to our Results

Our modified `scan.bro` is counting 893 lines (the original is 621 lines). We have decided not to remove our extra comments and notices used during analyzing phase. These lines may come handy when further analysis of this policy script take place.

We have in this thesis managed to improve the slow port scan detection capabilities in Bro. The old `scan.bro` script was most likely outdated. We strongly believe that Bro have added more functionality since the last version of `scan.bro` script was finalized ((3 Nov 2011).

7.3 Snort Results

We will in this section discuss our results in Snort IDS.

7.3.1 Limited slow port scan detection in Snort

We have used the open source IDS software Snort as our reference in our research. Our initial scan tests with increasing scan intervals against Snort gave us the slow port scan limit: 65 sec. This was by using Snort port scan detection preprocessor `sfportscan` configured as found in 6.5.

The same test with Bro indicated that this is only a matter of memory (resources). We tested at maximum 300 seconds scan interval and the scan was detected.

7.3.2 Slow port scan detection capabilities in Snort

We manage to detect the SYN and TCP Connect scan by Snort during all our experimental test. This was both in isolated and background traffic environment. The `sfportscan` configuration was tuned to level *high*, the most sensitive according to Snort documentation:

```
1 ( sense_level { high } )
```

In retrospect, we may have included Snort rules in our configuration to better detect slow port scanning. However, we did not find documentation that stated this. The preprocessor `sfportscan` is the port scan detection mechanism in Snort.

8 Conclusion

"I am extraordinarily patient,
provided I get my own way in the end."
– Margaret Thatcher

This thesis has shown that by modifying policy script `scan.bro`, we managed to improve slow port scan detection capabilities in Bro.

We introduce and explain the challenge regarding slow port scanning in Chapter 1. We strongly believe that this chapter explains the importance of detecting slow port scanning. Bro IDS is one of the systems that are improving our resistance to these attacks. We introduced our research questions in Chapter 1. Our Main Research Question (RQ1):
"Can we improve the detection rate regarding slow port scanning in Bro?
The main research question is answered by our sub-questions:

RQ1.1 Will we be able to improve the slow port scan detect rate in Bro?
RQ1.2 What is the slow port scan detection rate in Bro?

In Chapter 2 we describe some important technical details to better understand our thesis. We describe TCP/IP, port scanning, general IDS systems (including Snort) and especially Bro IDS.

In Chapter 3 we describe and briefly discuss previous work closely related to our topics. We did not find any previous research regarding Bro and slow port scanning. Malmedal does our most relevant research in his thesis from 2005 [92]. He writes about slow port scanning and Netflows and is using Snort IDS.

In Chapter 4 we describe our research methods in details. We start this chapter focusing on reliability and validity. We describes and discuss our decisions in details regarding; (i) strategy for improving `scan.bro`, (ii) out experimental test lab, (iii) our choices during tests, (iv) our tools needed and (v) adding background traffic by using a dataset. This is an important chapter that makes us prepared to perform our tests in proper scientific manner.

Chapter 5 is our core work in this thesis. We describe more Bro details and our efforts in improving slow port scan detection capabilities in Bro. We show here how we modified the `scan.bro` script.

In Chapter 6 we describe our experimental tests by applying our methods and improvements described in Chapter 4 and Chapter 5 respectively. We describe our test lab in details, the initial startup phase getting Bro up and running and how we simulated a scan attack using NMAP. We further show how we prepared CAIDA dataset for our needs, and show some statistics of our test experiment using Argus. We describe log output from Snort's `sfportscan` and finally we present our test results.

In chapter 7 we discuss our results. We first discuss some practical experiences in our test lab before we discussing our results in details. We are puzzled regarding Bro and connection history status "i" (inconsistency TCP packet), this was not properly documented on Bro Web.

Finally we conclude our findings and address our research questions in this chapter and present our ideas for future work in Chapter 9.

Addressing our Research Questions
RQ1.1 – Will we be able to improve the slow port scan detect rate in Bro?
Yes, we managed to improve Bro slow port scan detection capabilities and thus the detection rate.

RQ1.2 – What is the slow port scan detection rate in Bro?

We managed to improve Bro slow port scan detection capabilities for our six scan attacks with: 66%.

9 Further Work

"You are forgiven for your happiness and your successes
only if you generously consent to share them."
– Albert Camus

This thesis has shown that by modifying Bro policy script `scan.bro`, we managed to improve the slow port scan detection capabilities. Bro IDS is an interesting piece of software that most likely will be a source for many research papers (and perhaps thesis) in near future.

Perform a new slow port scan test with latest version of Bro

We would like to perform the same test by using latest version of Bro. The stable version of Bro 2.2 was finally released 7 Nov 2013.

Develop a Black Box solution based on Bro

We would like to create a "black box" to fulfill the needs The European Parliament Data Retention Directive that the Norwegian government most likely will apply in Norway [134]. We think Bro will be a brilliant software in this matter.

Bibliography

[1] Austevoll Kraftlag SA. Om LYSGLIMT.
http://www.lysglimt.net. Visited 2013-05-13.

[2] Shimeall, T., Faber, S., DeShon, M., & Kompanek, A. September 2010. SiLK - Analysis Handbook.
http://tools.netsa.cert.org/silk/analysis-handbook.pdf. Visited: 2013-11-07.

[3] Dale R. "Zai" Fox, Ph.D. Syn attacks.
http://zaielacademic.net/security/syn_attacks.htm. Visited: 2013-10-18.

[4] Petrović, S. IMT-4741-Intrusion Detection and Prevention. Class Lectures. Presentations. Gjøvik University College, 2010.

[5] The Bro Project. Introduction.
http://www.bro.org/sphinx/intro/index.html. Visited: 2013-10-23.

[6] Marchette, D. J. Sep 2011. *Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint (Information Science and Statistics)*. Springer.

[7] CAIDA. The CAIDA UCSD Anonymized Internet Traces 2013 - 20130815-134900.
http://www.caida.org/data/passive/passive_2013_dataset.xml.

[8] Lyon, G. F. Nmap - free security scanner for network exploration & security audits.
http://nmap.org,. Visited: 2013-10-19.

[9] Mell, P. & Grance, T. The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[10] Gerhards, R. March 2009. The Syslog Protocol. RFC 5424 (Proposed Standard).

[11] The Bro Project. The Bro Network Security Monitor.
http://bro.org/. Visited 2013-05-13.

[12] The Bro Project. Bro Research.
http://www.bro.org/research/index.html. Visited 2013-05-13.

[13] Chandola, V., Eilertson, E., Ertoz, L., Simon, G., & Kumar, V. January 2007. MINDS: Architecture & Design. In *Data Warehousing and Data Mining Techniques for Cyber Security*, number 31 in Advances in Information Security, 83–107. Springer US.

[14] Dabbagh, M., Ghandour, A., Fawaz, K., Hajj, W., & Hajj, H. December 2011. Slow port scanning detection. In *2011 7th International Conference on Information Assurance and Security (IAS)*, 228 –233.

[15] Barford, P. & Yegneswaran, V. 2007. An inside look at botnets. In *Malware Detection*, Christodorescu, M., Jha, S., Maughan, D., Song, D., & Wang, C., eds, volume 27 of *Advances in Information Security*, 171–191. Springer US. http://dx.doi.org/10.1007/978-0-387-44599-1_8.

[16] Bonfiglio, D., Mellia, M., Meo, M., Rossi, D., & Tofanelli, P. August 2007. Revealing skype traffic: when randomness plays with you. *SIGCOMM Comput. Commun. Rev.*, 37(4), 37–48.

[17] Zeidanloo, H., Shooshtari, M., Amoli, P., Safari, M., & Zamani, M. 2010. A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, 158 –162.

[18] Brownlee, N. 2012. One-way traffic monitoring with iatmon. In *Passive and Active Measurement*, Taft, N. & Ricciato, F., eds, volume 7192 of *Lecture Notes in Computer Science*, 179–188. Springer Berlin Heidelberg.

[19] Allman, M., Paxson, V., & Terrell, J. 2007. A brief history of scanning. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, 77–82, New York, NY, USA. ACM.

[20] Treurniet, J. 2011. A network activity classification schema and its application to scan detection. *Networking, IEEE/ACM Transactions on*, 19(5), 1396–1404.

[21] Glatz, E. & Dimitropoulos, X. 2012. Classifying internet one-way traffic. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, 37–50, New York, NY, USA. ACM.

[22] Pang, R., Yegneswaran, V., Barford, P., Paxson, V., & Peterson, L. 2004. Characteristics of internet background radiation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, 27–40, New York, NY, USA. ACM.

[23] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., & Kruegel, C. 2009. A view on current malware behaviors. In *USENIX workshop on large-scale exploits and emergent threats (LEET)*.

[24] Cerf, Vinton G. ICANN. About Vinton G Cerf. `http://www.icann.org/en/groups/board/cerf.htm`, May 2013.

[25] Ribeiro, J. February 2007. Cerf: Internet is a reflection of society. Infoworld, Inc. `http://www.infoworld.com/d/security-central/cerf-internet-reflection-society-625`.

[26] Hyman, P. March 2013. Cybercrime: it's serious, but exactly how serious? *Commun. ACM*, 56(3), 18–20.

[27] Clough, J. 2010. *Principles of cybercrime*. Cambridge University Press, Cambridge,UK.

[28] Symantec Corp. May 2013. 2013 Internet Security Threat Report, Volume 18. `http://www.symantec.com/security_response/publications/threatreport.jsp`,.

[29] Verizon RISK Team. 2013. The 2013 Data Breach Investigations Report. `http://www.verizonenterprise.com/DBIR/2013/`. Accessed: 2013-09-05.

[30] Paxson, V. 1999. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24), 2435–2463.

[31] Open Source Initiative OSI. March 2010. The BSD License. `http://www.opensource.org/licenses/bsd-license.php`.

[32] Cerf, V. & Kahn, R. 1974. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5), 637–648.

[33] Postel, J. September 1981. Transmission Control Protocol. RFC 793 (INTERNET STANDARD). Updated by RFCs 1122, 3168, 6093, 6528.

[34] Alvestrand, H. October 2004. A Mission Statement for the IETF. RFC 3935 (Best Current Practice).

[35] October 2004. About the IETF. `http://www.ietf.org/about/`,.

[36] Ramakrishnan, K., Floyd, S., & Black, D. September 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard). Updated by RFCs 4301, 6040.

[37] Spring, N., Wetherall, D., & Ely, D. June 2003. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. RFC 3540 (Experimental).

[38] Das, T. & Sivalingam, K. 2013. Tcp improvements for data center networks. In *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, 1–10.

[39] Cotton, M., Eggert, L., Touch, J., Westerlund, M., & Cheshire, S. August 2011. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335 (Best Current Practice).

[40] IANA. nov 2013. Service Name and Transport Protocol Port Number Registry. `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`.

[41] Messer, J. 2013. Secrets of Network Cartography: A Comprehensive Guide to nmap. `http://www.networkuptime.com/nmap/index.shtml`.

[42] Guang, C. april 2007. Tcp analysis based on flags. `http://www.nordu.net/development/2nd-cnnw/tcp-analysis-based-on-flags.pdf`. Visited: 2013-10-18.

[43] Weaver, N., Sommer, R., & Paxson, V. 2009. Detecting forged tcp reset packets. In *NDSS*. Lawrence Berkeley National Laboratory (LBNL), International Computer Science Institute (ICSI).

[44] Barnett, R. J. & Irwin, B. 2008. Towards a taxonomy of network scanning techniques. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, SAICSIT '08, 1–7, New York, NY, USA. ACM.

[45] Larsen, R. Fast-flux Service Networks in botnet malware. Visited: 2013-10-18, November 2010.

[46] Wolthusen, S. IMT 4651 Applied Information Security. Class Lectures. Presentations. Gjøvik University College, 2010.

[47] Camarillo, G. & IAB. November 2009. Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples and Applicability. RFC 5694 (Informational).

[48] Yegneswaran, V., Barford, P., & Ullrich, J. June 2003. Internet intrusions: global characteristics and prevalence. *SIGMETRICS Perform. Eval. Rev.*, 31(1), 138–147.

[49] Bhuyan, M. H., Bhattacharyya, D., & Kalita, J. 2011. Surveying port scans and their detection methodologies. *The Computer Journal*, 54(10), 1565–1581.

[50] de Vivo, M., Carrasco, E., Isern, G., & de Vivo, G. O. April 1999. A review of port scanning techniques. *SIGCOMM Comput. Commun. Rev.*, 29(2), 41–48.

[51] Lyon, G. F. 2009. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA.

[52] Orebaugh, A. & Pinkard, B. 2011. *Nmap in the Enterprise: Your Guide to Network Scanning*. Elsevier Science.

[53] Gadge, J. & Patil, A. 2008. Port scan detection. In *Networks, 2008. ICON 2008. 16th IEEE International Conference on*, 1–6.

[54] Scarfone, K. & Mell, P. Guide to Intrusion Detection and Prevention Systems (IDPS). Technical Report 800-94, National Institute of Standards and Technology (NIST), Gaithersburg, MD, februar 2007.

[55] November 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4), 262–294.

[56] Wang, X., Kordas, A., Hu, L., Gaedke, M., & Smith, D. 2013. Administrative evaluation of intrusion detection system. In *Proceedings of the 2nd annual conference on Research in information technology*, RIIT '13, 47–52, New York, NY, USA. ACM.

[57] Antonatos, S., Anagnostakis, K. G., & Markatos, E. P. January 2004. Generating realistic workloads for network intrusion detection systems. *SIGSOFT Softw. Eng. Notes*, 29(1), 207–215.

[58] Wilkison, M. Intrusion Detection FAQ: How to Evaluate Network Intrusion Detection Systems? `http://www.sans.org/security-resources/idfaq/eval_ids.php` . Visited: 2013-11-14.

[59] Fink, G., O'Donoghue, K. F., Chappell, B. L., & Turner, T. G. 2002. A metrics-based approach to intrusion detection system evaluation for distributed real-time systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, 17–, Washington, DC, USA. IEEE Computer Society.

[60] Alessandri, D. 2000. Using rule-based activity descriptions to evaluate intrusion-detection systems. In *Recent Advances in Intrusion Detection*, Debar, H., Mé, L., & Wu, S., eds, volume 1907 of *Lecture Notes in Computer Science*, 183–196. Springer Berlin Heidelberg.

[61] Srivastava, R. & Richhariya, V. 2013. Survey of Current Network Intrusion Detection Techniques. *Journal of Information Engineering and Applications*, 3(6), 27–33.

[62] Kibirkstis, A. November 2009. Intrusion detection faq: What are the top selling ids/ips and what differentiates them from each other? `http://www.sans.org/security-resources/idfaq/top-selling-ids-ips.php` . Visited: 2013-11-14.

[63] Sourcefire, Inc. 2013. About Snort. URL `http://www.snort.org/snort`. Visited 2013-11-11.

[64] 2013. About Suricata.
http://suricata-ids.org/. Visited: 2013-11-11.

[65] Larsen, R. BRO - an Intrusion Detection System. Visited: 2013-10-18, September 2011.

[66] Roesch, M. 1999. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, LISA '99, 229–238, Berkeley, CA, USA. USENIX Association.

[67] Catania, C. A. & Garino, C. G. September 2012. Automatic network intrusion detection: Current techniques and open issues. *Comput. Electr. Eng.*, 38(5), 1062–1072.

[68] Meier, M., Schmerl, S., & Koenig, H. 2005. Improving the efficiency of misuse detection. In *Proceedings of the Second international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'05, 188–205, Berlin, Heidelberg. Springer-Verlag.

[69] Chabchoub, Y., Fricker, C., & Robert, P. oct. 2012. Improving the detection of on-line vertical port scan in ip traffic. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, 1 –6.

[70] Koziol, J. 2003. *Intrusion Detection with Snort*. Sams.

[71] Sourcefire, Inc. 2013. About Snort Preprocessors.
http://manual.snort.org/node17.html,. Visited 2013-11-26.

[72] Paxson, V. 1998. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, 3–3, Berkeley, CA, USA. USENIX Association.

[73] Zhang, J. & Moore, A. May 2007. Traffic trace artifacts due to monitoring via port mirroring. In *End-to-End Monitoring Techniques and Services, 2007. E2EMON '07. Workshop on*, 1 –8.

[74] Risso, F. & Degioanni, L. 2001. An architecture for high performance network analysis. In *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*, 686 –693.

[75] The TCPdump Team. September 2013. Tcpdump/libpcap public repository.
http://www.tcpdump.org/index.html. Visited 2013-10-18.

[76] Day, J. D. & Zimmermann, H. 1983. The OSI reference model. *Proceedings of the IEEE*, 71(12), 1334–1340.

[77] The Bro Project. Bro Workshop 2011.
http://www.bro.org/bro-workshop-2011/.

[78] The Bro Project. Bro base/event.bif.bro.
http://www.bro.org/sphinx/scripts/base/event.bif.html
. Visited 2013-09-19.

[79] The Bro Project. BroControl.
http://www.bro.org/documentation/broctl.broctl.html
. Visited 2013-10-18.

[80] The Bro Project. Customizing Bro's Logging.
http://www.bro.org/sphinx/logging.html. Visited 2013-10-13.

[81] The Bro Project. Bro Downloads Archive.
http://bro.org/downloads/archive/. Visited 2013-10-13.

[82] The Bro Project. All Bro Scrips. Visited: 2013-10-19.

[83] Denning, D. 1987. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, SE-13(2), 222–232.

[84] Anderson, D., Lunt, T., Javitz, H., Tamaru, A., & Valdes, A. 1995. Next-generation intrusion detection expert system (nides): A summary.

[85] The International Computer Science Institute (ICSI), University of California at Berkeley.
http://www.icsi.berkeley.edu/icsi/about,.

[86] Sommer, R. & Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, 305–316, Washington, DC, USA. IEEE Computer Society.

[87] Zhang, W., Teng, S., & Fu, X. 2008. Scan attack detection based on distributed cooperative model. In *12th International Conference on Computer Supported Cooperative Work in Design, 2008. CSCWD 2008*, 743–748.

[88] Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P., Srivastava, J., Kumar, V., Dokas, P. 2004. The MINDS - minnesota intrusion detection system. In *Next Generation Data Mining*. MIT Press.

[89] Jung, J., Paxson, V., Berger, A. W., & Balakrishnan, H. May 2004. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA.

[90] Staniford, S., Hoagland, J. A., & McAlerney, J. M. 2002. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1), 105–136.

[91] Kim, J. & Lee, J.-H. july 2008. A slow port scan attack detection mechanism based on fuzzy logic and a stepwise policy. In *2008 IET 4th International Conference on Intelligent Environments*, 1–5.

[92] Malmedal, B. 2005. Using netflows for slow portscan detection.

[93] QoSient, LLC. . ARGUS - Auditing Network Activity.
http://qosient.com/argus/index.shtml. Visited: 2013-09-16.

[94] The PostgreSQL Global Development Group. About PostgreSQL.
http://www.postgresql.org/about/,.

[95] The PHP Group. About PHP.
http://us1.php.net/,.

[96] JGraph Ltd. About JGraph.
http://www.jgraph.com/company.html,.

[97] Paul D. Leedy, J. E. O. 2010. *Practical Research: Planning and Design*. Pearson Education Inc., 9 edition.

[98] Volden, F. August 2010. Lecture notes. imt4421 - scientific methodology - 2010 - 5 ects.
http://english.hig.no/content/view/full/21608/language/eng-US.

[99] Bejtlich, R. Bro Change Log.
http://www.bro.org/download/CHANGES.bro.txt,. Visited: 2013-11-24.

[100] Lyon, G. F. Nmap network scanning: Download. Visited: 2013-10-19.

[101] Pang, R. *Towards understanding application semantics of network traffic*. PhD thesis, Princeton, NJ, USA, 2008. AAI3305770.

[102] The Bro Project. Bro Installation Prerequisites.
http://bro.org/sphinx/install/install.html#prerequisites,. Visited: 2013-11-13.

[103] Linux mint 13 xfce.
http://linuxmint.com/rel_maya_xfce.php,. Visited: 2013-11-17.

[104] Software in the Public Interest, Inc. Linux debian.
http://www.debian.org/intro/about. Visited: 2013-11-17.

[105] Xfce Development Team. Xfce Desktop Environment.
http://www.xfce.org/. Visited: 2013-11-19.

[106] About OpenSSH.
http://www.openssh.com/,. Visited: 2013-11-24.

[107] The Bro Project. September 2013. Installation and configuration.
http://www.bro.org/documentation/faq.html#installation-and-configuration.
Visited 2013-09-27.

[108] Braun, L., Didebulidze, A., Kammenhuber, N., & Carle, G. 2010. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, 206–217, New York, NY, USA. ACM.

[109] Burks, D. September 2013. When is full packet capture NOT full packet capture?
http://securityonion.blogspot.no/2011/10/when-is-full-packet-capture-not-full.
html. Visited: 2013-05-17.

[110] Currid, A. May 2004. TCP Offload to the Rescue. *Queue*, 2(3), 58–65.

[111] Gordon "Fyodor" Lyon. sep 2013. About Gordon "Fyodor" Lyon.
http://insecure.org/fyodor/. Last visited:2013-09-26.

[112] Pale, P. 2012. *Nmap 6: Network Exploration and Security Auditing Cookbook*. Packt open source. Packt Publishing, Limited.

[113] Bejtlich, R. 2004. *The Tao of network security monitoring: beyond intrusion detection*. Addison-Wesley.

[114] Pallis, G. A comparative study of in-band and out-of-band VOIP protocols in layer 3 and layer 2.5 environments. 2010.

[115] Lovdata. Lov om behandling av personopplysninger (personopplysningsloven) / Personal Data Act 2000-04.
http://www.lovdata.no/all/hl-20000414-031.html, September 2013.

[116] CAIDA. The Cooperative Association for Internet Data Analysis (CAIDA). http://www.caida.org/home/about/.

[117] Srisuresh, P. & Egevang, K. January 2001. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational).

[118] Turner, A. & Bing, M. November 2013. Tcpreplay.

[119] Hillestad, O. I., Libak, B., & Perkis, A. 2005. Performance evaluation of multimedia services over ip networks. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, 1464–1467.

[120] Khan, Z. A., Javaid, N., Arshad, M. H., Bibi, A., & Qasim, B. 2012. Performance evaluation of widely used portknoking algorithms. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, 903–907.

[121] The Bro Project. September 2013. The Bro Script; SCAN.BRO. https://github.com/bro/bro-scripts. Visited 2013-09-16.

[122] The Bro Project. Bro - Notice Framework. http://www.bro.org/sphinx-git/frameworks/notice.html. Visited 2013-11-13.

[123] The Bro Project. Bro - Notice.log (base/frameworks/notice/main.bro). http://www.bro.org/sphinx/scripts/base/frameworks/notice/main.html,. Visited: 2013-11-07.

[124] The Bro Project. Protocol Connection Tracking and Logging Events. http://www.bro.org/sphinx-git/scripts/proto-analyzers.html#bro-tcp . Visited 2013-05-13.

[125] The Bro Project. Protocol Connection Tracking and Logging - Conn::Info Script. http://www.bro.org/sphinx/_downloads/main14.bro . Visited 2013-09-19.

[126] The Bro Project. Bro - base/init-bare.bro. http://www.bro.org/sphinx-git/scripts/base/init-bare.html. Visited: 2013-10-23.

[127] The Bro Project. base/protocols/conn/main.bro - Conn::info. Visited: 2013-10-23.

[128] The Bro Project. Local Site Policy. http://www.bro.org/sphinx/scripts/site/local.html. Visited: 2013-10-18.

[129] The TCPdump Team. Pcap next generation dump file format. http://www.tcpdump.org/pcap/pcap.html. Visited 2013-11-11.

[130] Turner, A. & Bing, M. November 2013. Tcprewrite.

[131] Botta, A., Dainotti, A., & Pescapé, A. October 2012. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput. Netw.*, 56(15), 3531–3547.

[132] Deri, L., Netikos, Via, & La Figuretta, L. 2004. Improving passive packet capture:beyond device polling. In *In Proceedings of SANE 2004*.

[133] Deering, S. & Hinden, R. December 1998. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[134] The Europe Parliament. March 2006. Directive 2006/24/ec of the european parliament and of the council. Electronic. Visited: 2013-09-05.

A TCP State Machine

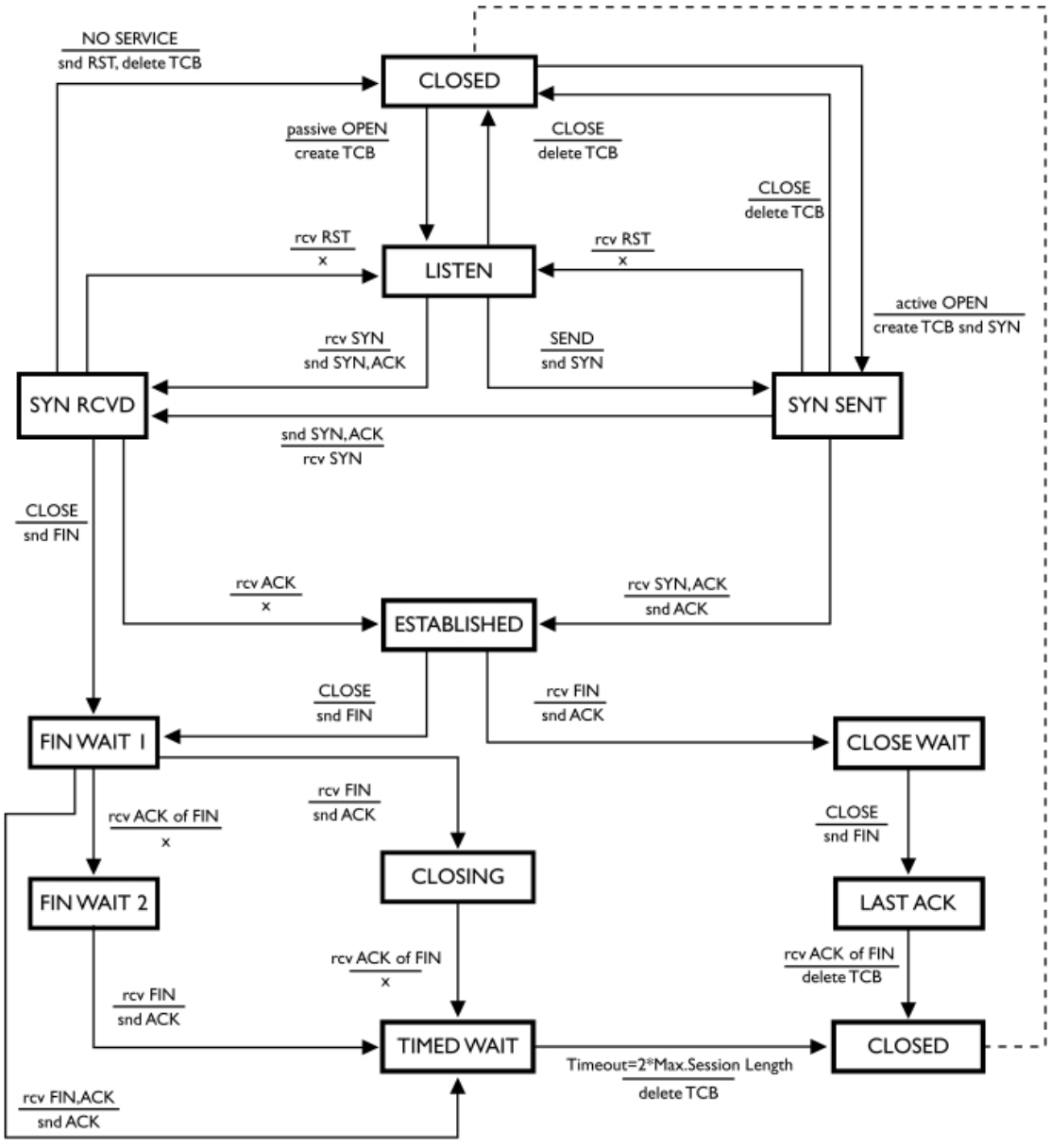

Figure 12: TCP State Machine. Illustration is taken from [2]

B NMAP

B.1 NMAP - 100 most used ports below 1024

We show here the modified NMAP file `nmap-services` file. All lines defining ports over 1023 are not included. The list below are sorted descending with third column (open-frequency) as source.

```
1  # THIS FILE IS GENERATED AUTOMATICALLY FROM A MASTER - DO NOT EDIT.
2  # EDIT /nmap-private-dev/nmap-services-all IN SVN INSTEAD.
3  # Well known service port numbers -*- mode: fundamental; -*-
4  # From the Nmap Security Scanner ( http://nmap.org )
5  #
6  # $Id: nmap-services 30210 2012-11-07 21:34:43Z fyodor $
7  #
8  # Derived from IANA data and our own research
9  #
10 # This collection of service data is (C) 1996-2011 by Insecure.Com
11 # LLC.  It is distributed under the Nmap Open Source license as
12 # provided in the COPYING file of the source distribution or at
13 # http://nmap.org/data/COPYING .  Note that this license
14 # requires you to license your own work under a compatable open source
15 # license.  If you wish to embed Nmap technology into proprietary
16 # software, we sell alternative licenses (contact sales@insecure.com).
17 # Dozens of software vendors already license Nmap technology such as
18 # host discovery, port scanning, OS detection, and version detection.
19 # For more details, see http://nmap.org/book/man-legal.html
20 #
21 # Fields in this file are: Service name, portnum/protocol, open-frequency, optional
       comments
22 #
23 http  80/tcp  0.484143  # World Wide Web HTTP
24 telnet  23/tcp  0.221265
25 https 443/tcp 0.208669  # secure http (SSL)
26 ftp 21/tcp  0.197667  # File Transfer [Control]
27 ssh 22/tcp  0.182286  # Secure Shell Login
28 smtp  25/tcp  0.131314  # Simple Mail Transfer
29 pop3  110/tcp 0.077142  # PostOffice V.3
30 microsoft-ds  445/tcp 0.056944  # SMB directly over IP
31 netbios-ssn 139/tcp 0.050809  # NETBIOS Session Service
32 imap  143/tcp 0.050420  # Interim Mail Access Protocol v2
33 domain  53/tcp  0.048463  # Domain Name Server
34 msrpc 135/tcp 0.047798  # Microsoft RPC services
35 rpcbind 111/tcp 0.030034  # portmapper, rpcbind
36 pop3s 995/tcp 0.029921  # POP3 protocol over TLS/SSL
37 imaps 993/tcp 0.027199  # imap4 protocol over TLS/SSL
38 submission  587/tcp 0.019721
39 smux  199/tcp 0.015945  # SNMP Unix Multiplexer
40 smtps 465/tcp 0.013888  # smtp protocol over TLS/SSL (was ssmtp)
41 afp 548/tcp 0.012395  # AFP over TCP
42 ident 113/tcp 0.012370  # ident, tap, Authentication Service
43 hosts2-ns 81/tcp  0.012056  # HOSTS2 Name Server
44 shell 514/tcp 0.011078  # BSD rshd(8)
45 bgp 179/tcp 0.010538  # Border Gateway Protocol
46 rtsp  554/tcp 0.008104  # Real Time Stream Control Protocol
47 rsftp 26/tcp  0.007991  # RSFTP
48 printer 515/tcp 0.007214  # spooler (lpd)
49 ldp 646/tcp 0.006549  # Label Distribution
50 ipp 631/tcp 0.006160  # Internet Printing Protocol -- for one implementation see
       http://www.cups.org (Common UNIX Printing System)
51 kerberos-sec  88/tcp  0.006072  # Kerberos (v5)
52 finger  79/tcp  0.006022
```

```
53 pop3pw 106/tcp 0.005934 # Eudora compatible PW changer
54 login 513/tcp 0.005595 # BSD rlogind(8)
55 ftps 990/tcp 0.005570 # ftp protocol, control, over TLS/SSL
56 svrloc 427/tcp 0.005382 # Server Location
57 klogin 543/tcp 0.005282 # Kerberos (v4/v5)
58 kshell 544/tcp 0.005269 # krcmd Kerberos (v4/v5)
59 news 144/tcp 0.004981 # NewS window system
60 echo 7/tcp 0.004855
61 ldap 389/tcp 0.004717 # Lightweight Directory Access Protocol
62 snpp 444/tcp 0.004466 # Simple Network Paging Protocol
63 daytime 13/tcp 0.003927
64 discard 9/tcp 0.003764 # sink null
65 rsync 873/tcp 0.003400 # Rsync server ( http://rsync.samba.org )
66 nntp 119/tcp 0.003262 # Network News Transfer Protocol
67 time 37/tcp 0.003161 # timserver
68 cadlock 1000/tcp 0.003149
69 xfer 82/tcp 0.002923 # XFER Utility
70 chargen 19/tcp 0.002559 # ttytst source Character Generator
71 unknown 255/tcp 0.002409
72 qotd 17/tcp 0.002346 # Quote of the Day
73 ccproxy-http 808/tcp 0.002296 # CCProxy HTTP/Gopher/FTP (over HTTP) proxy
74 newacct 100/tcp 0.002133 # [unauthorized use]
75 ldapssl 636/tcp 0.002083 # LDAP over SSL
76 tcpmux 1/tcp 0.001995 # TCP Port Service Multiplexer [rfc-1078]
77 apple-xsrvr-admin 625/tcp 0.001869 # Apple Mac Xserver admin
78 asip-webadmin 311/tcp 0.001857 # appleshare ip webadmin
79 http-mgmt 280/tcp 0.001844
80 unknown 254/tcp 0.001832
81 iss-realsecure 902/tcp 0.001468 # ISS RealSecure Sensor
82 qsc 787/tcp 0.001455
83 compressnet 3/tcp 0.001242 # Compression Process
84 http-rpc-epmap 593/tcp 0.001242 # HTTP RPC Ep Map
85 exp2 1022/tcp 0.001217 # RFC3692-style Experiment 2 (*) [RFC4727]
86 kpasswd5 464/tcp 0.001192 # Kerberos (v5)
87 retrospect 497/tcp 0.001179
88 priv-mail 24/tcp 0.001154 # any private mail system
89 timbuktu 407/tcp 0.001129
90 isakmp 500/tcp 0.001129
91 ftp-data 20/tcp 0.001079 # File Transfer [Default Data]
92 bgmp 264/tcp 0.001029
93 dsp 33/tcp 0.001016 # Display Support Protocol
94 garcon 999/tcp 0.000966
95 netvenuechat 1023/tcp 0.000953 # Nortel NetVenue Notification, Chat, Intercom
96 rsh-spx 222/tcp 0.000941 # Berkeley rshd with SPX auth
97 accessbuilder 888/tcp 0.000928 # or Audio CD Database
98 snews 563/tcp 0.000916
99 telnets 992/tcp 0.000903 # telnet protocol over TLS/SSL
100 exec 512/tcp 0.000841 # BSD rexecd(8)
101 nameserver 42/tcp 0.000803 # Host Name Server
102 snmp 161/tcp 0.000790
103 odmr 366/tcp 0.000715
104 mit-ml-dev 85/tcp 0.000690 # MIT ML Device
105 windows-icfw 1002/tcp 0.000690 # Windows Internet Connection Firewall or Internet
        Locator Server for NetMeeting.
106 tacacs 49/tcp 0.000665 # Login Host Protocol (TACACS)
107 dnsix 90/tcp 0.000652 # DNSIX Securit Attribute Token Map
108 unknown 340/tcp 0.000627
109 cmip-man 163/tcp 0.000590 # CMIP/TCP Manager
110 iso-tp0 146/tcp 0.000577
111 rrp 648/tcp 0.000577 # Registry Registrar Protocol (RRP)
112 samba-swat 901/tcp 0.000552 # Samba SWAT tool. Also used by ISS RealSecure.
113 mit-ml-dev 83/tcp 0.000539 # MIT ML Device
114 unknown 30/tcp 0.000527
115 apex-mesh 912/tcp 0.000527 # APEX relay-relay service
116 unknown 6/tcp 0.000502
117 uucp-rlogin 541/tcp 0.000489
118 unknown 4/tcp 0.000477
```

```
119 unknown 306/tcp 0.000464
120 unknown 880/tcp 0.000464
121 omginitialrefs 900/tcp 0.000452 # OMG Initial Refs
122 914c-g 211/tcp 0.000427 # Texas Instruments 914C/G Terminal
```

B.2 NMAP - Sample output, SYN Scan

We used the following command (the dollar sign ($) represents the command prompt):

```
$nmap --dns-servers 94.143.64.11 -sS --top-ports 100 -P0 -T0 --scan-delay 60s --max-scan-delay 61s 192.168.1.102 --packet-trace
```

```
Starting Nmap 6.25 ( http://nmap.org ) at 2013-11-19 09:25 CET
SENT (0.1111s) ARP who-has 192.168.1.102 tell 192.168.1.200
RCVD (0.1114s) ARP reply 192.168.1.102 is-at 00:23:5A:62:1B:35
NSOCK (0.1130s) nsi_new (IOD #1)
NSOCK (0.1130s) UDP connection requested to 95.143.64.11:53 (IOD #1) EID 8
NSOCK (0.1130s) Read request from IOD #1 [95.143.64.11:53] (timeout: -1ms) EID 18
NSOCK (0.1130s) Write request for 44 bytes to IOD #1 EID 27 [95.143.64.11:53]: -...........102.1.168.192.in-addr.arpa.....
NSOCK (0.1130s) Callback: CONNECT SUCCESS for EID 8 [95.143.64.11:53]
NSOCK (0.1130s) Callback: WRITE SUCCESS for EID 27 [95.143.64.11:53]
NSOCK (64.1150s) Write request for 44 bytes to IOD #1 EID 35 [95.143.64.11:53]: -...........102.1.168.192.in-addr.arpa.....
NSOCK (64.1150s) Callback: WRITE SUCCESS for EID 35 [95.143.64.11:53]
NSOCK (68.1170s) Write request for 44 bytes to IOD #1 EID 43 [95.143.64.11:53]: -...........102.1.168.192.in-addr.arpa.....
NSOCK (68.1170s) Callback: WRITE SUCCESS for EID 43 [95.143.64.11:53]
NSOCK (73.1180s) nsi_delete (IOD #1)
NSOCK (73.1180s) msevent_cancel on event #18 (type READ)
SENT (133.2189s) TCP 192.168.1.200:42392 > 192.168.1.102:110 S ttl=57 id=58115 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (133.2191s) TCP 192.168.1.102:110 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (193.3190s) TCP 192.168.1.200:42392 > 192.168.1.102:23 S ttl=52 id=43087 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (193.3194s) TCP 192.168.1.102:23 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (253.4191s) TCP 192.168.1.200:42392 > 192.168.1.102:995 S ttl=49 id=56741 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (253.4195s) TCP 192.168.1.102:995 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (313.5192s) TCP 192.168.1.200:42392 > 192.168.1.102:443 S ttl=45 id=39187 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (313.5196s) TCP 192.168.1.102:443 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (373.6193s) TCP 192.168.1.200:42392 > 192.168.1.102:53 S ttl=58 id=54611 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (373.6196s) TCP 192.168.1.102:53 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (433.7194s) TCP 192.168.1.200:42392 > 192.168.1.102:199 S ttl=58 id=48291 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (433.7197s) TCP 192.168.1.102:199 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (493.8176s) TCP 192.168.1.200:42392 > 192.168.1.102:22 S ttl=48 id=21024 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (493.8179s) TCP 192.168.1.102:22 > 192.168.1.200:42392 SA ttl=64 id=0 iplen=44 seq=3912834382 win=14600 <mss 1460>
SENT (553.9177s) TCP 192.168.1.200:42392 > 192.168.1.102:993 S ttl=49 id=40736 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (553.9179s) TCP 192.168.1.102:993 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (614.0178s) TCP 192.168.1.200:42392 > 192.168.1.102:445 S ttl=59 id=63215 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (614.0180s) TCP 192.168.1.102:445 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (674.1179s) TCP 192.168.1.200:42392 > 192.168.1.102:587 S ttl=41 id=56201 iplen=44 seq=4148668469 win=1024 <mss 1460>
RCVD (674.1182s) TCP 192.168.1.102:587 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40 seq=0 win=0
SENT (734.2180s) TCP 192.168.1.200:42403 > 192.168.1.102:110 S ttl=50 id=36408 iplen=44 seq=4131891509 win=1024 <mss 1460>
```

```
37 RCVD (734.2184s) TCP 192.168.1.102:110 > 192.168.1.200:42403 RA ttl=64 id=0 iplen=40
       seq=0 win=0
38 SENT (794.3181s) TCP 192.168.1.200:42392 > 192.168.1.102:143 S ttl=45 id=25647 iplen=44
        seq=4148668469 win=1024 <mss 1460>
39 RCVD (794.3184s) TCP 192.168.1.102:143 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
40 SENT (854.4182s) TCP 192.168.1.200:42392 > 192.168.1.102:135 S ttl=54 id=7395 iplen=44
       seq=4148668469 win=1024 <mss 1460>
41 RCVD (854.4185s) TCP 192.168.1.102:135 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
42 SENT (914.5182s) TCP 192.168.1.200:42392 > 192.168.1.102:111 S ttl=38 id=10926 iplen=44
        seq=4148668469 win=1024 <mss 1460>
43 RCVD (914.5185s) TCP 192.168.1.102:111 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
44 SENT (974.6183s) TCP 192.168.1.200:42392 > 192.168.1.102:21 S ttl=57 id=16525 iplen=44
       seq=4148668469 win=1024 <mss 1460>
45 RCVD (974.6186s) TCP 192.168.1.102:21 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
       =0 win=0
46 SENT (1034.7185s) TCP 192.168.1.200:42392 > 192.168.1.102:139 S ttl=54 id=18237 iplen=44
         seq=4148668469 win=1024 <mss 1460>
47 RCVD (1034.7188s) TCP 192.168.1.102:139 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
48 SENT (1094.8186s) TCP 192.168.1.200:42392 > 192.168.1.102:80 S ttl=56 id=11017 iplen=44
        seq=4148668469 win=1024 <mss 1460>
49 RCVD (1094.8188s) TCP 192.168.1.102:80 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
50 SENT (1154.9187s) TCP 192.168.1.200:42392 > 192.168.1.102:113 S ttl=52 id=19553 iplen=44
         seq=4148668469 win=1024 <mss 1460>
51 RCVD (1154.9189s) TCP 192.168.1.102:113 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
52 SENT (1215.0188s) TCP 192.168.1.200:42392 > 192.168.1.102:25 S ttl=52 id=53356 iplen=44
        seq=4148668469 win=1024 <mss 1460>
53 RCVD (1215.0190s) TCP 192.168.1.102:25 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
54 SENT (1275.1189s) TCP 192.168.1.200:42392 > 192.168.1.102:554 S ttl=58 id=12992 iplen=44
         seq=4148668469 win=1024 <mss 1460>
55 RCVD (1275.1191s) TCP 192.168.1.102:554 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
56 SENT (1335.2190s) TCP 192.168.1.200:42392 > 192.168.1.102:444 S ttl=50 id=11616 iplen=44
         seq=4148668469 win=1024 <mss 1460>
57 RCVD (1335.2192s) TCP 192.168.1.102:444 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
58 SENT (1395.2910s) TCP 192.168.1.200:42404 > 192.168.1.102:110 S ttl=59 id=36192 iplen=44
         seq=4115114549 win=1024 <mss 1460>
59 RCVD (1395.2913s) TCP 192.168.1.102:110 > 192.168.1.200:42404 RA ttl=64 id=0 iplen=40
       seq=0 win=0
60 SENT (1455.3911s) TCP 192.168.1.200:42392 > 192.168.1.102:311 S ttl=58 id=54338 iplen=44
         seq=4148668469 win=1024 <mss 1460>
61 RCVD (1455.3915s) TCP 192.168.1.102:311 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
62 SENT (1515.4912s) TCP 192.168.1.200:42392 > 192.168.1.102:83 S ttl=48 id=64467 iplen=44
        seq=4148668469 win=1024 <mss 1460>
63 RCVD (1515.4917s) TCP 192.168.1.102:83 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
64 SENT (1575.5913s) TCP 192.168.1.200:42392 > 192.168.1.102:161 S ttl=53 id=31053 iplen=44
         seq=4148668469 win=1024 <mss 1460>
65 RCVD (1575.5916s) TCP 192.168.1.102:161 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
66 SENT (1635.6914s) TCP 192.168.1.200:42392 > 192.168.1.102:3 S ttl=42 id=36322 iplen=44
       seq=4148668469 win=1024 <mss 1460>
67 RCVD (1635.6917s) TCP 192.168.1.102:3 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
       =0 win=0
68 SENT (1695.7915s) TCP 192.168.1.200:42392 > 192.168.1.102:1002 S ttl=46 id=56268 iplen
       =44  seq=4148668469 win=1024 <mss 1460>
69 RCVD (1695.7919s) TCP 192.168.1.102:1002 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
       seq=0 win=0
```

```
70 SENT (1755.8916s) TCP 192.168.1.200:42392 > 192.168.1.102:512 S ttl=38 id=36946 iplen=44
       seq=4148668469 win=1024 <mss 1460>
71 RCVD (1755.8918s) TCP 192.168.1.102:512 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
72 SENT (1815.9917s) TCP 192.168.1.200:42392 > 192.168.1.102:24 S ttl=49 id=18134 iplen=44
       seq=4148668469 win=1024 <mss 1460>
73 RCVD (1815.9919s) TCP 192.168.1.102:24 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
74 SENT (1876.0918s) TCP 192.168.1.200:42392 > 192.168.1.102:407 S ttl=39 id=33167 iplen=44
       seq=4148668469 win=1024 <mss 1460>
75 RCVD (1876.0920s) TCP 192.168.1.102:407 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
76 SENT (1936.1919s) TCP 192.168.1.200:42392 > 192.168.1.102:625 S ttl=48 id=31324 iplen=44
       seq=4148668469 win=1024 <mss 1460>
77 RCVD (1936.1921s) TCP 192.168.1.102:625 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
78 SENT (1996.2920s) TCP 192.168.1.200:42392 > 192.168.1.102:88 S ttl=54 id=10998 iplen=44
       seq=4148668469 win=1024 <mss 1460>
79 RCVD (1996.2922s) TCP 192.168.1.102:88 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
80 SENT (2056.3921s) TCP 192.168.1.200:42405 > 192.168.1.102:110 S ttl=58 id=63205 iplen=44
       seq=4098337589 win=1024 <mss 1460>
81 RCVD (2056.3923s) TCP 192.168.1.102:110 > 192.168.1.200:42405 RA ttl=64 id=0 iplen=40
     seq=0 win=0
82 SENT (2116.4922s) TCP 192.168.1.200:42392 > 192.168.1.102:79 S ttl=55 id=13380 iplen=44
       seq=4148668469 win=1024 <mss 1460>
83 RCVD (2116.4927s) TCP 192.168.1.102:79 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
84 SENT (2176.5923s) TCP 192.168.1.200:42392 > 192.168.1.102:9 S ttl=49 id=22035 iplen=44
     seq=4148668469 win=1024 <mss 1460>
85 RCVD (2176.5925s) TCP 192.168.1.102:9 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
     =0 win=0
86 SENT (2236.6924s) TCP 192.168.1.200:42392 > 192.168.1.102:514 S ttl=54 id=26554 iplen=44
       seq=4148668469 win=1024 <mss 1460>
87 RCVD (2236.6926s) TCP 192.168.1.102:514 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
88 SENT (2296.7925s) TCP 192.168.1.200:42392 > 192.168.1.102:264 S ttl=52 id=58861 iplen=44
       seq=4148668469 win=1024 <mss 1460>
89 RCVD (2296.7927s) TCP 192.168.1.102:264 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
90 SENT (2356.8816s) TCP 192.168.1.200:42392 > 192.168.1.102:163 S ttl=48 id=57669 iplen=44
       seq=4148668469 win=1024 <mss 1460>
91 RCVD (2356.8818s) TCP 192.168.1.102:163 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
92 SENT (2416.9817s) TCP 192.168.1.200:42392 > 192.168.1.102:1022 S ttl=56 id=22550 iplen
     =44  seq=4148668469 win=1024 <mss 1460>
93 RCVD (2416.9819s) TCP 192.168.1.102:1022 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
94 SENT (2477.0818s) TCP 192.168.1.200:42392 > 192.168.1.102:13 S ttl=50 id=42303 iplen=44
       seq=4148668469 win=1024 <mss 1460>
95 RCVD (2477.0820s) TCP 192.168.1.102:13 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
96 SENT (2537.1819s) TCP 192.168.1.200:42392 > 192.168.1.102:1 S ttl=53 id=46234 iplen=44
     seq=4148668469 win=1024 <mss 1460>
97 RCVD (2537.1821s) TCP 192.168.1.102:1 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
     =0 win=0
98 SENT (2597.2820s) TCP 192.168.1.200:42392 > 192.168.1.102:992 S ttl=58 id=2956 iplen=44
       seq=4148668469 win=1024 <mss 1460>
99 RCVD (2597.2825s) TCP 192.168.1.102:992 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
100 SENT (2657.3821s) TCP 192.168.1.200:42392 > 192.168.1.102:500 S ttl=40 id=6402 iplen=44
       seq=4148668469 win=1024 <mss 1460>
101 RCVD (2657.3826s) TCP 192.168.1.102:500 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
     seq=0 win=0
102 SENT (2717.4822s) TCP 192.168.1.200:42406 > 192.168.1.102:110 S ttl=47 id=3933 iplen=44
       seq=4081560629 win=1024 <mss 1460>
```

```
103 RCVD (2717.4824s) TCP 192.168.1.102:110 > 192.168.1.200:42406 RA ttl=64 id=0 iplen=40
      seq=0 win=0
104 SENT (2777.5823s) TCP 192.168.1.200:42392 > 192.168.1.102:515 S ttl=40 id=47943 iplen=44
       seq=4148668469 win=1024 <mss 1460>
105 RCVD (2777.5825s) TCP 192.168.1.102:515 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
106 SENT (2837.6824s) TCP 192.168.1.200:42392 > 192.168.1.102:646 S ttl=45 id=21001 iplen=44
       seq=4148668469 win=1024 <mss 1460>
107 RCVD (2837.6829s) TCP 192.168.1.102:646 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
108 SENT (2897.7825s) TCP 192.168.1.200:42392 > 192.168.1.102:33 S ttl=37 id=5374 iplen=44
      seq=4148668469 win=1024 <mss 1460>
109 RCVD (2897.7827s) TCP 192.168.1.102:33 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
110 SENT (2957.8826s) TCP 192.168.1.200:42392 > 192.168.1.102:119 S ttl=48 id=16382 iplen=44
       seq=4148668469 win=1024 <mss 1460>
111 RCVD (2957.8828s) TCP 192.168.1.102:119 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
112 SENT (3017.9827s) TCP 192.168.1.200:42392 > 192.168.1.102:211 S ttl=42 id=61034 iplen=44
       seq=4148668469 win=1024 <mss 1460>
113 RCVD (3017.9829s) TCP 192.168.1.102:211 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
114 SENT (3078.0828s) TCP 192.168.1.200:42392 > 192.168.1.102:366 S ttl=44 id=12459 iplen=44
       seq=4148668469 win=1024 <mss 1460>
115 RCVD (3078.0830s) TCP 192.168.1.102:366 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
116 SENT (3138.1829s) TCP 192.168.1.200:42392 > 192.168.1.102:880 S ttl=49 id=21945 iplen=44
       seq=4148668469 win=1024 <mss 1460>
117 RCVD (3138.1831s) TCP 192.168.1.102:880 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
118 SENT (3198.2830s) TCP 192.168.1.200:42392 > 192.168.1.102:17 S ttl=48 id=2667 iplen=44
      seq=4148668469 win=1024 <mss 1460>
119 RCVD (3198.2832s) TCP 192.168.1.102:17 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
120 SENT (3258.3831s) TCP 192.168.1.200:42392 > 192.168.1.102:636 S ttl=39 id=19972 iplen=44
       seq=4148668469 win=1024 <mss 1460>
121 RCVD (3258.3833s) TCP 192.168.1.102:636 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
122 SENT (3318.4832s) TCP 192.168.1.200:42392 > 192.168.1.102:255 S ttl=54 id=37020 iplen=44
       seq=4148668469 win=1024 <mss 1460>
123 RCVD (3318.4834s) TCP 192.168.1.102:255 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
124 SENT (3378.5835s) TCP 192.168.1.200:42407 > 192.168.1.102:110 S ttl=44 id=44436 iplen=44
       seq=4064783669 win=1024 <mss 1460>
125 RCVD (3378.5837s) TCP 192.168.1.102:110 > 192.168.1.200:42407 RA ttl=64 id=0 iplen=40
      seq=0 win=0
126 SENT (3438.6836s) TCP 192.168.1.200:42392 > 192.168.1.102:427 S ttl=37 id=1005 iplen=44
       seq=4148668469 win=1024 <mss 1460>
127 RCVD (3438.6838s) TCP 192.168.1.102:427 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
128 SENT (3498.7837s) TCP 192.168.1.200:42392 > 192.168.1.102:7 S ttl=58 id=3379 iplen=44
      seq=4148668469 win=1024 <mss 1460>
129 RCVD (3498.7839s) TCP 192.168.1.102:7 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
      =0 win=0
130 SENT (3558.8835s) TCP 192.168.1.200:42392 > 192.168.1.102:464 S ttl=44 id=27922 iplen=44
       seq=4148668469 win=1024 <mss 1460>
131 RCVD (3558.8840s) TCP 192.168.1.102:464 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
132 SENT (3618.9839s) TCP 192.168.1.200:42392 > 192.168.1.102:20 S ttl=42 id=40335 iplen=44
       seq=4148668469 win=1024 <mss 1460>
133 RCVD (3618.9841s) TCP 192.168.1.102:20 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
134 SENT (3679.0840s) TCP 192.168.1.200:42392 > 192.168.1.102:593 S ttl=41 id=31500 iplen=44
       seq=4148668469 win=1024 <mss 1460>
135 RCVD (3679.0842s) TCP 192.168.1.102:593 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
```

```
136 SENT (3739.1841s) TCP 192.168.1.200:42392 > 192.168.1.102:543 S ttl=51 id=33430 iplen=44
        seq=4148668469 win=1024 <mss 1460>
137 RCVD (3739.1843s) TCP 192.168.1.102:543 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
138 SENT (3799.2842s) TCP 192.168.1.200:42392 > 192.168.1.102:544 S ttl=39 id=5739 iplen=44
        seq=4148668469 win=1024 <mss 1460>
139 RCVD (3799.2844s) TCP 192.168.1.102:544 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
140 SENT (3859.3843s) TCP 192.168.1.200:42392 > 192.168.1.102:513 S ttl=54 id=35475 iplen=44
        seq=4148668469 win=1024 <mss 1460>
141 RCVD (3859.3845s) TCP 192.168.1.102:513 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
142 SENT (3919.4844s) TCP 192.168.1.200:42392 > 192.168.1.102:563 S ttl=47 id=36758 iplen=44
        seq=4148668469 win=1024 <mss 1460>
143 RCVD (3919.4846s) TCP 192.168.1.102:563 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
144 SENT (3979.5845s) TCP 192.168.1.200:42392 > 192.168.1.102:901 S ttl=53 id=30526 iplen=44
        seq=4148668469 win=1024 <mss 1460>
145 RCVD (3979.5847s) TCP 192.168.1.102:901 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
146 SENT (4039.6816s) TCP 192.168.1.200:42408 > 192.168.1.102:110 S ttl=38 id=19777 iplen=44
        seq=4048006709 win=1024 <mss 1460>
147 RCVD (4039.6819s) TCP 192.168.1.102:110 > 192.168.1.200:42408 RA ttl=64 id=0 iplen=40
      seq=0 win=0
148 SENT (4099.7817s) TCP 192.168.1.200:42392 > 192.168.1.102:179 S ttl=43 id=6754 iplen=44
        seq=4148668469 win=1024 <mss 1460>
149 RCVD (4099.7817s) TCP 192.168.1.102:179 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
150 SENT (4159.8818s) TCP 192.168.1.200:42392 > 192.168.1.102:648 S ttl=54 id=4347 iplen=44
        seq=4148668469 win=1024 <mss 1460>
151 RCVD (4159.8818s) TCP 192.168.1.102:648 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
152 SENT (4219.9819s) TCP 192.168.1.200:42392 > 192.168.1.102:548 S ttl=51 id=36149 iplen=44
        seq=4148668469 win=1024 <mss 1460>
153 RCVD (4219.9819s) TCP 192.168.1.102:548 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
154 SENT (4280.0820s) TCP 192.168.1.200:42392 > 192.168.1.102:787 S ttl=38 id=25964 iplen=44
        seq=4148668469 win=1024 <mss 1460>
155 RCVD (4280.0820s) TCP 192.168.1.102:787 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
156 SENT (4340.1821s) TCP 192.168.1.200:42392 > 192.168.1.102:999 S ttl=53 id=13640 iplen=44
        seq=4148668469 win=1024 <mss 1460>
157 RCVD (4340.1821s) TCP 192.168.1.102:999 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
158 SENT (4400.2822s) TCP 192.168.1.200:42392 > 192.168.1.102:37 S ttl=44 id=13782 iplen=44
        seq=4148668469 win=1024 <mss 1460>
159 RCVD (4400.2822s) TCP 192.168.1.102:37 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
160 SENT (4460.3823s) TCP 192.168.1.200:42392 > 192.168.1.102:30 S ttl=43 id=22354 iplen=44
        seq=4148668469 win=1024 <mss 1460>
161 RCVD (4460.3823s) TCP 192.168.1.102:30 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
162 SENT (4520.4824s) TCP 192.168.1.200:42392 > 192.168.1.102:340 S ttl=42 id=55526 iplen=44
        seq=4148668469 win=1024 <mss 1460>
163 RCVD (4520.4824s) TCP 192.168.1.102:340 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
164 SENT (4580.5825s) TCP 192.168.1.200:42392 > 192.168.1.102:42 S ttl=44 id=45682 iplen=44
        seq=4148668469 win=1024 <mss 1460>
165 RCVD (4580.5825s) TCP 192.168.1.102:42 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
166 SENT (4640.6826s) TCP 192.168.1.200:42392 > 192.168.1.102:19 S ttl=43 id=37072 iplen=44
        seq=4148668469 win=1024 <mss 1460>
167 RCVD (4640.6826s) TCP 192.168.1.102:19 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
168 SENT (4700.7827s) TCP 192.168.1.200:42409 > 192.168.1.102:110 S ttl=40 id=42772 iplen=44
        seq=4031229749 win=1024 <mss 1460>
```

```
169 RCVD (4700.7827s) TCP 192.168.1.102:110 > 192.168.1.200:42409 RA ttl=64 id=0 iplen=40
      seq=0 win=0
170 SENT (4760.8818s) TCP 192.168.1.200:42392 > 192.168.1.102:82 S ttl=55 id=52829 iplen=44
       seq=4148668469 win=1024 <mss 1460>
171 RCVD (4760.8818s) TCP 192.168.1.102:82 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
172 SENT (4820.9819s) TCP 192.168.1.200:42392 > 192.168.1.102:900 S ttl=58 id=65408 iplen=44
       seq=4148668469 win=1024 <mss 1460>
173 RCVD (4820.9819s) TCP 192.168.1.102:900 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
174 SENT (4881.0820s) TCP 192.168.1.200:42392 > 192.168.1.102:541 S ttl=49 id=49879 iplen=44
       seq=4148668469 win=1024 <mss 1460>
175 RCVD (4881.0820s) TCP 192.168.1.102:541 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
176 SENT (4941.1821s) TCP 192.168.1.200:42392 > 192.168.1.102:26 S ttl=39 id=41678 iplen=44
       seq=4148668469 win=1024 <mss 1460>
177 RCVD (4941.1821s) TCP 192.168.1.102:26 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
178 SENT (5001.2822s) TCP 192.168.1.200:42392 > 192.168.1.102:100 S ttl=56 id=32398 iplen=44
       seq=4148668469 win=1024 <mss 1460>
179 RCVD (5001.2822s) TCP 192.168.1.102:100 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
180 SENT (5061.3823s) TCP 192.168.1.200:42392 > 192.168.1.102:1023 S ttl=55 id=46206 iplen
      =44  seq=4148668469 win=1024 <mss 1460>
181 RCVD (5061.3823s) TCP 192.168.1.102:1023 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
182 SENT (5121.4824s) TCP 192.168.1.200:42392 > 192.168.1.102:254 S ttl=55 id=58957 iplen=44
       seq=4148668469 win=1024 <mss 1460>
183 RCVD (5121.4824s) TCP 192.168.1.102:254 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
184 SENT (5181.5825s) TCP 192.168.1.200:42392 > 192.168.1.102:873 S ttl=42 id=32460 iplen=44
       seq=4148668469 win=1024 <mss 1460>
185 RCVD (5181.5825s) TCP 192.168.1.102:873 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
186 SENT (5241.6826s) TCP 192.168.1.200:42392 > 192.168.1.102:49 S ttl=57 id=37512 iplen=44
       seq=4148668469 win=1024 <mss 1460>
187 RCVD (5241.6826s) TCP 192.168.1.102:49 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
188 SENT (5301.7827s) TCP 192.168.1.200:42392 > 192.168.1.102:144 S ttl=58 id=56000 iplen=44
       seq=4148668469 win=1024 <mss 1460>
189 RCVD (5301.7827s) TCP 192.168.1.102:144 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
190 SENT (5361.8174s) TCP 192.168.1.200:42410 > 192.168.1.102:110 S ttl=44 id=52209 iplen=44
       seq=4282884149 win=1024 <mss 1460>
191 RCVD (5361.8179s) TCP 192.168.1.102:110 > 192.168.1.200:42410 RA ttl=64 id=0 iplen=40
      seq=0 win=0
192 SENT (5421.9175s) TCP 192.168.1.200:42392 > 192.168.1.102:990 S ttl=59 id=14753 iplen=44
       seq=4148668469 win=1024 <mss 1460>
193 RCVD (5421.9180s) TCP 192.168.1.102:990 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
194 SENT (5482.0176s) TCP 192.168.1.200:42392 > 192.168.1.102:85 S ttl=59 id=39579 iplen=44
       seq=4148668469 win=1024 <mss 1460>
195 RCVD (5482.0181s) TCP 192.168.1.102:85 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
196 SENT (5542.1177s) TCP 192.168.1.200:42392 > 192.168.1.102:465 S ttl=42 id=49722 iplen=44
       seq=4148668469 win=1024 <mss 1460>
197 RCVD (5542.1182s) TCP 192.168.1.102:465 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
198 SENT (5602.2178s) TCP 192.168.1.200:42392 > 192.168.1.102:389 S ttl=50 id=60110 iplen=44
       seq=4148668469 win=1024 <mss 1460>
199 RCVD (5602.2183s) TCP 192.168.1.102:389 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
200 SENT (5662.3179s) TCP 192.168.1.200:42392 > 192.168.1.102:631 S ttl=51 id=15321 iplen=44
       seq=4148668469 win=1024 <mss 1460>
201 RCVD (5662.3184s) TCP 192.168.1.102:631 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
```

```
202 SENT (5722.4180s) TCP 192.168.1.200:42392 > 192.168.1.102:902 S ttl=53 id=46450 iplen=44
        seq=4148668469 win=1024 <mss 1460>
203 RCVD (5722.4185s) TCP 192.168.1.102:902 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
204 SENT (5782.5181s) TCP 192.168.1.200:42392 > 192.168.1.102:280 S ttl=49 id=58846 iplen=44
        seq=4148668469 win=1024 <mss 1460>
205 RCVD (5782.5186s) TCP 192.168.1.102:280 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
206 SENT (5842.6182s) TCP 192.168.1.200:42392 > 192.168.1.102:912 S ttl=52 id=65034 iplen=44
        seq=4148668469 win=1024 <mss 1460>
207 RCVD (5842.6187s) TCP 192.168.1.102:912 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
208 SENT (5902.7183s) TCP 192.168.1.200:42392 > 192.168.1.102:81 S ttl=37 id=10218 iplen=44
        seq=4148668469 win=1024 <mss 1460>
209 RCVD (5902.7188s) TCP 192.168.1.102:81 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
210 SENT (5962.8184s) TCP 192.168.1.200:42392 > 192.168.1.102:6 S ttl=56 id=3811 iplen=44
      seq=4148668469 win=1024 <mss 1460>
211 RCVD (5962.8188s) TCP 192.168.1.102:6 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
      =0 win=0
212 SENT (6022.9185s) TCP 192.168.1.200:42411 > 192.168.1.102:110 S ttl=48 id=60083 iplen=44
        seq=4266107189 win=1024 <mss 1460>
213 RCVD (6022.9189s) TCP 192.168.1.102:110 > 192.168.1.200:42411 RA ttl=64 id=0 iplen=40
      seq=0 win=0
214 SENT (6083.0186s) TCP 192.168.1.200:42392 > 192.168.1.102:306 S ttl=48 id=32114 iplen=44
        seq=4148668469 win=1024 <mss 1460>
215 RCVD (6083.0190s) TCP 192.168.1.102:306 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
216 SENT (6143.1187s) TCP 192.168.1.200:42392 > 192.168.1.102:497 S ttl=40 id=52835 iplen=44
        seq=4148668469 win=1024 <mss 1460>
217 RCVD (6143.1191s) TCP 192.168.1.102:497 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
218 SENT (6203.2188s) TCP 192.168.1.200:42392 > 192.168.1.102:90 S ttl=45 id=14040 iplen=44
        seq=4148668469 win=1024 <mss 1460>
219 RCVD (6203.2192s) TCP 192.168.1.102:90 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
220 SENT (6263.3188s) TCP 192.168.1.200:42392 > 192.168.1.102:222 S ttl=44 id=24645 iplen=44
        seq=4148668469 win=1024 <mss 1460>
221 RCVD (6263.3193s) TCP 192.168.1.102:222 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
222 SENT (6323.4189s) TCP 192.168.1.200:42392 > 192.168.1.102:106 S ttl=59 id=56892 iplen=44
        seq=4148668469 win=1024 <mss 1460>
223 RCVD (6323.4194s) TCP 192.168.1.102:106 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
224 SENT (6383.5190s) TCP 192.168.1.200:42392 > 192.168.1.102:1000 S ttl=41 id=50151 iplen
      =44  seq=4148668469 win=1024 <mss 1460>
225 RCVD (6383.5195s) TCP 192.168.1.102:1000 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
226 SENT (6443.6191s) TCP 192.168.1.200:42392 > 192.168.1.102:808 S ttl=57 id=60440 iplen=44
        seq=4148668469 win=1024 <mss 1460>
227 RCVD (6443.6196s) TCP 192.168.1.102:808 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
228 SENT (6503.7192s) TCP 192.168.1.200:42392 > 192.168.1.102:146 S ttl=54 id=29299 iplen=44
        seq=4148668469 win=1024 <mss 1460>
229 RCVD (6503.7197s) TCP 192.168.1.102:146 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
230 SENT (6563.8193s) TCP 192.168.1.200:42392 > 192.168.1.102:888 S ttl=57 id=51731 iplen=44
        seq=4148668469 win=1024 <mss 1460>
231 RCVD (6563.8198s) TCP 192.168.1.102:888 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40
      seq=0 win=0
232 SENT (6623.9194s) TCP 192.168.1.200:42392 > 192.168.1.102:4 S ttl=47 id=4909 iplen=44
      seq=4148668469 win=1024 <mss 1460>
233 RCVD (6623.9199s) TCP 192.168.1.102:4 > 192.168.1.200:42392 RA ttl=64 id=0 iplen=40  seq
      =0 win=0
234 Nmap scan report for 192.168.1.102
235 Host is up (0.00034s latency).
236 Not shown: 99 closed ports
```

```
237 PORT    STATE SERVICE
238 22/tcp open  ssh
239 MAC Address: 00:23:5A:62:1B:35 (Compal Information (kunshan) CO.)
240
241 Nmap done: 1 IP address (1 host up) scanned in 6623.94 seconds
```

C Bro

C.1 Error in scan.bro found by Dr. Slobodan Petrović (GUC)

Dr. Slobodan Petrović (GUC) found an error in the original `scan.bro` script. In line number 490 the variable "n" was most likely suppose to be "m".

The following listing show line 490 in the original `scan.bro` from 28 Aug 2012:

```
490     $identifier=fmt("%s-%d", orig, n),
```

The following listing show line 490 after the change was applied `scan.bro`:

```
490     $identifier=fmt("%s-%d", orig, m),
```

C.2 Bro Configuration File: local.bro

```
##! Local site policy. Customize as appropriate.
##!
##! This file will not be overwritten when upgrading or reinstalling!

# This script logs which scripts were loaded during each run.
@load misc/loaded-scripts

# Apply the default tuning scripts for common tuning settings.
@load tuning/defaults

# Generate notices when vulnerable versions of software are discovered.
# The default is to only monitor software found in the address space defined
# as "local".  Refer to the software framework's documentation for more
# information.
@load frameworks/software/vulnerable

# Example vulnerable software.  This needs to be updated and maintained over
# time as new vulnerabilities are discovered.
redef Software::vulnerable_versions += {
        ["Flash"] = [$major=10,$minor=2,$minor2=153,$addl="1"],
        ["Java"] = [$major=1,$minor=6,$minor2=0,$addl="22"],
};

# Detect software changing (e.g. attacker installing hacked SSHD).
@load frameworks/software/version-changes

# This adds signatures to detect cleartext forward and reverse windows shells.
@load-sigs frameworks/signatures/detect-windows-shells

# Uncomment the following line to begin receiving (by default hourly) emails
# containing all of your notices.
# redef Notice::policy += { [$action = Notice::ACTION_ALARM, $priority = 0] };

# Load all of the scripts that detect software in various protocols.
@load protocols/ftp/software
@load protocols/smtp/software
@load protocols/ssh/software
@load protocols/http/software
# The detect-webapps script could possibly cause performance trouble when
# running on live traffic.  Enable it cautiously.
#@load protocols/http/detect-webapps

# This script detects DNS results pointing toward your Site::local_nets
# where the name is not part of your local DNS zone and is being hosted
# externally.  Requires that the Site::local_zones variable is defined.
@load protocols/dns/detect-external-names

# Script to detect various activity in FTP sessions.
@load protocols/ftp/detect

# Scripts that do asset tracking.
@load protocols/conn/known-hosts
@load protocols/conn/known-services
@load protocols/ssl/known-certs

# This script enables SSL/TLS certificate validation.
@load protocols/ssl/validate-certs

# If you have libGeoIP support built in, do some geographic detections and
# logging for SSH traffic.
@load protocols/ssh/geo-data
# Detect hosts doing SSH bruteforce attacks.
@load protocols/ssh/detect-bruteforcing
# Detect logins using "interesting" hostnames.
@load protocols/ssh/interesting-hostnames
```

```
66
67 # Detect MD5 sums in Team Cymru's Malware Hash Registry.
68 @load protocols/http/detect-MHR
69 # Detect SQL injection attacks.
70 @load protocols/http/detect-sqli
71
72 #       Other scripts
73 #       @load policy/misc/scan
74 @load policy/misc/scan4
75
76 redef Site::local_nets = {
77 #    10.0.0.0/8,          # Untrusted IP range
78         192.168.1.0/25,      # Trusted IP range
79 };
```

C.3 Bro File Structure

Bro file structure is listed below. We start from /usr/local/bro:

```
2013-10-17 09:54:58 root@bigbro:/usr/local/bro ] # tree -d
.
|-- bin
|-- etc
|-- include
|-- lib
|   |-- broctl
|   |   |-- BroControl
|   |   `-- plugins
|   `-- ruby
|       `-- Broccoli
|-- logs
|   |-- YYYY-MM-DD
|   `-- current -> /usr/local/bro/spool/bro
|-- share
|   |-- bro
|   |   |-- base
|   |   |   |-- frameworks
|   |   |   |   |-- cluster
|   |   |   |   |   `-- nodes
|   |   |   |   |-- communication
|   |   |   |   |-- control
|   |   |   |   |-- dpd
|   |   |   |   |-- input
|   |   |   |   |   `-- readers
|   |   |   |   |-- intel
|   |   |   |   |-- logging
|   |   |   |   |   |-- postprocessors
|   |   |   |   |   `-- writers
|   |   |   |   |-- metrics
|   |   |   |   |-- notice
|   |   |   |   |   |-- actions
|   |   |   |   |   `-- extend-email
|   |   |   |   |-- packet-filter
|   |   |   |   |-- reporter
|   |   |   |   |-- signatures
|   |   |   |   |-- software
|   |   |   |   `-- tunnels
|   |   |   |-- misc
|   |   |   |-- protocols
|   |   |   |   |-- conn
|   |   |   |   |-- dns
|   |   |   |   |-- ftp
```

```
|   |   |   |   |-- http
|   |   |   |   |-- irc
|   |   |   |   |-- modbus
|   |   |   |   |-- smtp
|   |   |   |   |-- socks
|   |   |   |   |-- ssh
|   |   |   |   |-- ssl
|   |   |   |   `-- syslog
|   |   |   `-- utils
|   |   |-- broctl
|   |   |-- policy
|   |   |   |-- frameworks
|   |   |   |   |-- communication
|   |   |   |   |-- control
|   |   |   |   |-- dpd
|   |   |   |   |-- intel
|   |   |   |   |-- metrics
|   |   |   |   |-- signatures
|   |   |   |   `-- software
|   |   |   |-- integration
|   |   |   |   |-- barnyard2
|   |   |   |   `-- collective-intel
|   |   |   |-- misc
|   |   |   |-- protocols
|   |   |   |   |-- conn
|   |   |   |   |-- dns
|   |   |   |   |-- ftp
|   |   |   |   |-- http
|   |   |   |   |-- modbus
|   |   |   |   |-- smtp
|   |   |   |   |-- ssh
|   |   |   |   `-- ssl
|   |   |   `-- tuning
|   |   |       `-- defaults
|   |   `-- site
|   `-- broctl
|       `-- scripts
|           |-- helpers
|           `-- postprocessors
`-- spool
    |-- bro
    |-- installed-scripts-do-not-touch
    |   |-- auto
    |   `-- site
    |-- scripts
    `-- tmp
```

C.4 Bro Script Scan.bro - Original Version from 28 Aug 2012

```
##! Scan detector ported from Bro 1.x.
##!
##! This script has evolved over many years and is quite a mess right now. We
##! have adapted it to work with Bro 2.x, but eventually Bro 2.x will
##! get its own rewritten and generalized scan detector.

@load base/frameworks/notice/main

module Scan;

export {
  redef enum Notice::Type += {
    ## The source has scanned a number of ports.
    PortScan,
    ## The source has scanned a number of addresses.
    AddressScan,
    ## Apparent flooding backscatter seen from source.
    BackscatterSeen,

    ## Summary of scanning activity.
    ScanSummary,
    ## Summary of distinct ports per scanner.
    PortScanSummary,
    ## Summary of distinct low ports per scanner.
    LowPortScanSummary,

    ## Source reached :bro:id:`Scan::shut_down_thresh`
    ShutdownThresh,
    ## Source touched privileged ports.
    LowPortTrolling,
  };

  # Whether to consider UDP "connections" for scan detection.
  # Can lead to false positives due to UDP fanout from some P2P apps.
  const suppress_UDP_scan_checks = F &redef;

  const activate_priv_port_check = T &redef;
  const activate_landmine_check = F &redef;
  const landmine_thresh_trigger = 5 &redef;

  const landmine_address: set[addr] &redef;

  const scan_summary_trigger = 25 &redef;
  const port_summary_trigger = 20 &redef;
  const lowport_summary_trigger = 10 &redef;

  # Raise ShutdownThresh after this many failed attempts
  const shut_down_thresh = 100 &redef;

  # Which services should be analyzed when detecting scanning
  # (not consulted if analyze_all_services is set).
  const analyze_services: set[port] &redef;
  const analyze_all_services = T &redef;

  # Track address scaners only if at least these many hosts contacted.
  const addr_scan_trigger = 0 &redef;

  # Ignore address scanners for further scan detection after
  # scanning this many hosts.
  # 0 disables.
  const ignore_scanners_threshold = 0 &redef;

  # Report a scan of peers at each of these points.
  const report_peer_scan: vector of count = {
    20, 100, 1000, 10000, 50000, 100000, 250000, 500000, 1000000,
```

```
66  } &redef;
67
68  const report_outbound_peer_scan: vector of count = {
69    100, 1000, 10000,
70  } &redef;
71
72  # Report a scan of ports at each of these points.
73  const report_port_scan: vector of count = {
74    50, 250, 1000, 5000, 10000, 25000, 65000,
75  } &redef;
76
77  # Once a source has scanned this many different ports (to however many
78  # different remote hosts), start tracking its per-destination access.
79  const possible_port_scan_thresh = 20 &redef;
80
81  # Threshold for scanning privileged ports.
82  const priv_scan_trigger = 5 &redef;
83  const troll_skip_service = {
84    25/tcp, 21/tcp, 22/tcp, 20/tcp, 80/tcp,
85  } &redef;
86
87  const report_accounts_tried: vector of count = {
88    20, 100, 1000, 10000, 100000, 1000000,
89  } &redef;
90
91  const report_remote_accounts_tried: vector of count = {
92    100, 500,
93  } &redef;
94
95  # Report a successful password guessing if the source attempted
96  # at least this many.
97  const password_guessing_success_threshhold = 20 &redef;
98
99  const skip_accounts_tried: set[addr] &redef;
100
101 const addl_web = {
102   81/tcp, 443/tcp, 8000/tcp, 8001/tcp, 8080/tcp, }
103 &redef;
104
105 const skip_services = { 113/tcp, } &redef;
106 const skip_outbound_services = { 21/tcp, addl_web, }
107   &redef;
108
109 const skip_scan_sources = {
110   255.255.255.255,  # who knows why we see these, but we do
111 } &redef;
112
113 const skip_scan_nets: set[subnet] = {} &redef;
114
115 # List of well known local server/ports to exclude for scanning
116 # purposes.
117 const skip_dest_server_ports: set[addr, port] = {} &redef;
118
119 # Reverse (SYN-ack) scans seen from these ports are considered
120 # to reflect possible SYN-flooding backscatter, and not true
121 # (stealth) scans.
122 const backscatter_ports = {
123   80/tcp, 8080/tcp, 53/tcp, 53/udp, 179/tcp, 6666/tcp, 6667/tcp,
124 } &redef;
125
126 const report_backscatter: vector of count = {
127   20,
128 } &redef;
129
130 global check_scan:
131   function(c: connection, established: bool, reverse: bool): bool;
132
```

```
133   # The following tables are defined here so that we can redef
134   # the expire timeouts.
135   # FIXME: should we allow redef of attributes on IDs which
136   # are not exported?
137
138   # How many different hosts connected to with a possible
139   # backscatter signature.
140   global distinct_backscatter_peers: table[addr] of table[addr] of count
141     &read_expire = 15 min;
142
143   # Expire functions that trigger summaries.
144   global scan_summary:
145     function(t: table[addr] of set[addr], orig: addr): interval;
146   global port_summary:
147     function(t: table[addr] of set[port], orig: addr): interval;
148   global lowport_summary:
149     function(t: table[addr] of set[port], orig: addr): interval;
150
151   # Indexed by scanner address, yields # distinct peers scanned.
152   # pre_distinct_peers tracks until addr_scan_trigger hosts first.
153   global pre_distinct_peers: table[addr] of set[addr]
154     &read_expire = 15 mins &redef;
155
156   global distinct_peers: table[addr] of set[addr]
157     &read_expire = 15 mins &expire_func=scan_summary &redef;
158   global distinct_ports: table[addr] of set[port]
159     &read_expire = 15 mins &expire_func=port_summary &redef;
160   global distinct_low_ports: table[addr] of set[port]
161     &read_expire = 15 mins &expire_func=lowport_summary &redef;
162
163   # Indexed by scanner address, yields a table with scanned hosts
164   # (and ports).
165   global scan_triples: table[addr] of table[addr] of set[port];
166
167   global remove_possible_source:
168     function(s: set[addr], idx: addr): interval;
169   global possible_scan_sources: set[addr]
170     &expire_func=remove_possible_source &read_expire = 15 mins;
171
172   # Indexed by source address, yields user name & password tried.
173   global accounts_tried: table[addr] of set[string, string]
174     &read_expire = 1 days;
175
176   global ignored_scanners: set[addr] &create_expire = 1 day &redef;
177
178   # These tables track whether a threshold has been reached.
179   # More precisely, the counter is the next index of threshold vector.
180   global shut_down_thresh_reached: table[addr] of bool &default=F;
181   global rb_idx: table[addr] of count
182       &default=1 &read_expire = 1 days &redef;
183   global rps_idx: table[addr] of count
184       &default=1 &read_expire = 1 days &redef;
185   global rops_idx: table[addr] of count
186       &default=1 &read_expire = 1 days &redef;
187   global rpts_idx: table[addr,addr] of count
188       &default=1 &read_expire = 1 days &redef;
189   global rat_idx: table[addr] of count
190       &default=1 &read_expire = 1 days &redef;
191   global rrat_idx: table[addr] of count
192       &default=1 &read_expire = 1 days &redef;
193 }
194
195 global thresh_check: function(v: vector of count, idx: table[addr] of count,
196         orig: addr, n: count): bool;
197 global thresh_check_2: function(v: vector of count,
198         idx: table[addr,addr] of count, orig: addr,
199         resp: addr, n: count): bool;
```

```
200
201 function scan_summary(t: table[addr] of set[addr], orig: addr): interval
202   {
203   local num_distinct_peers = orig in t ? |t[orig]| : 0;
204
205   if ( num_distinct_peers >= scan_summary_trigger )
206     NOTICE([$note=ScanSummary, $src=orig, $n=num_distinct_peers,
207       $identifier=fmt("%s", orig),
208       $msg=fmt("%s scanned a total of %d hosts",
209           orig, num_distinct_peers)]);
210
211   return 0 secs;
212   }
213
214 function port_summary(t: table[addr] of set[port], orig: addr): interval
215   {
216   local num_distinct_ports = orig in t ? |t[orig]| : 0;
217
218   if ( num_distinct_ports >= port_summary_trigger )
219     NOTICE([$note=PortScanSummary, $src=orig, $n=num_distinct_ports,
220       $identifier=fmt("%s", orig),
221       $msg=fmt("%s scanned a total of %d ports",
222           orig, num_distinct_ports)]);
223
224   return 0 secs;
225   }
226
227 function lowport_summary(t: table[addr] of set[port], orig: addr): interval
228   {
229   local num_distinct_lowports = orig in t ? |t[orig]| : 0;
230
231   if ( num_distinct_lowports >= lowport_summary_trigger )
232     NOTICE([$note=LowPortScanSummary, $src=orig,
233       $n=num_distinct_lowports,
234       $identifier=fmt("%s", orig),
235       $msg=fmt("%s scanned a total of %d low ports",
236           orig, num_distinct_lowports)]);
237
238   return 0 secs;
239   }
240
241 function clear_addr(a: addr)
242   {
243   delete distinct_peers[a];
244   delete distinct_ports[a];
245   delete distinct_low_ports[a];
246   delete scan_triples[a];
247   delete possible_scan_sources[a];
248   delete distinct_backscatter_peers[a];
249   delete pre_distinct_peers[a];
250   delete rb_idx[a];
251   delete rps_idx[a];
252   delete rops_idx[a];
253   delete rat_idx[a];
254   delete rrat_idx[a];
255   delete shut_down_thresh_reached[a];
256   delete ignored_scanners[a];
257   }
258
259 function ignore_addr(a: addr)
260   {
261   clear_addr(a);
262   add ignored_scanners[a];
263   }
264
265 function check_scan(c: connection, established: bool, reverse: bool): bool
266   {
```

```
267   local id = c$id;
268
269   local service = "ftp-data" in c$service ? 20/tcp
270       : (reverse ? id$orig_p : id$resp_p);
271   local rev_service = reverse ? id$resp_p : id$orig_p;
272   local orig = reverse ? id$resp_h : id$orig_h;
273   local resp = reverse ? id$orig_h : id$resp_h;
274   local outbound = Site::is_local_addr(orig);
275
276   # The following works better than using get_conn_transport_proto()
277   # because c might not correspond to an active connection (which
278   # causes the function to fail).
279   if ( suppress_UDP_scan_checks &&
280        service >= 0/udp && service <= 65535/udp )
281     return F;
282
283   if ( service in skip_services && ! outbound )
284     return F;
285
286   if ( outbound && service in skip_outbound_services )
287     return F;
288
289   if ( orig in skip_scan_sources )
290     return F;
291
292   if ( orig in skip_scan_nets )
293     return F;
294
295   # Don't include well known server/ports for scanning purposes.
296   if ( ! outbound && [resp, service] in skip_dest_server_ports )
297     return F;
298
299   if ( orig in ignored_scanners)
300     return F;
301
302   if ( ! established &&
303     # not established, service not expressly allowed
304
305     # not known peer set
306     (orig !in distinct_peers || resp !in distinct_peers[orig]) &&
307
308     # want to consider service for scan detection
309     (analyze_all_services || service in analyze_services) )
310     {
311     if ( reverse && rev_service in backscatter_ports &&
312          # reverse, non-priv backscatter port
313          service >= 1024/tcp )
314       {
315       if ( orig !in distinct_backscatter_peers )
316         {
317         local empty_bs_table:
318           table[addr] of count &default=0;
319         distinct_backscatter_peers[orig] =
320           empty_bs_table;
321         }
322
323       if ( ++distinct_backscatter_peers[orig][resp] <= 2 &&
324            # The test is <= 2 because we get two check_scan()
325            # calls, once on connection attempt and once on
326            # tear-down.
327
328            distinct_backscatter_peers[orig][resp] == 1 &&
329
330            # Looks like backscatter, and it's not scanning
331            # a privileged port.
332
333            thresh_check(report_backscatter, rb_idx, orig,
```

```
334           |distinct_backscatter_peers[orig]|)
335          )
336         {
337         NOTICE([$note=BackscatterSeen, $src=orig,
338           $p=rev_service,
339           $identifier=fmt("%s", orig),
340           $msg=fmt("backscatter seen from %s (%d hosts; %s)",
341             orig, |distinct_backscatter_peers[orig]|, rev_service)]);
342         }
343
344       if ( ignore_scanners_threshold > 0 &&
345            |distinct_backscatter_peers[orig]| >
346           ignore_scanners_threshold )
347         ignore_addr(orig);
348       }
349
350     else
351       { # done with backscatter check
352       local ignore = F;
353
354       if ( orig !in distinct_peers && addr_scan_trigger > 0 )
355         {
356         if ( orig !in pre_distinct_peers )
357           pre_distinct_peers[orig] = set();
358
359         add pre_distinct_peers[orig][resp];
360         if ( |pre_distinct_peers[orig]| < addr_scan_trigger )
361           ignore = T;
362         }
363
364       if ( ! ignore )
365         { # XXXXX
366
367         if ( orig !in distinct_peers )
368           distinct_peers[orig] = set() &mergeable;
369
370         if ( resp !in distinct_peers[orig] )
371           add distinct_peers[orig][resp];
372
373         local n = |distinct_peers[orig]|;
374
375         # Check for threshold if not outbound.
376         if ( ! shut_down_thresh_reached[orig] &&
377              n >= shut_down_thresh &&
378              ! outbound && orig !in Site::neighbor_nets )
379           {
380           shut_down_thresh_reached[orig] = T;
381           local msg = fmt("shutdown threshold reached for %s", orig);
382           NOTICE([$note=ShutdownThresh, $src=orig,
383             $identifier=fmt("%s", orig),
384             $p=service, $msg=msg]);
385           }
386
387         else
388           {
389           local address_scan = F;
390           if ( outbound &&
391                # inside host scanning out?
392                thresh_check(report_outbound_peer_scan, rops_idx, orig, n) )
393             address_scan = T;
394
395           if ( ! outbound &&
396                thresh_check(report_peer_scan, rps_idx, orig, n) )
397             address_scan = T;
398
399           if ( address_scan )
400             NOTICE([$note=AddressScan,
```

```
401                 $src=orig, $p=service,
402                 $n=n,
403                 $identifier=fmt("%s-%d", orig, n),
404                 $msg=fmt("%s has scanned %d hosts (%s)",
405                   orig, n, service)]);
406
407             if ( address_scan &&
408                  ignore_scanners_threshold > 0 &&
409                  n > ignore_scanners_threshold )
410               ignore_addr(orig);
411             }
412           }
413       } # XXXX
414     }
415
416   if ( established )
417     # Don't consider established connections for port scanning,
418     # it's too easy to be mislead by FTP-like applications that
419     # legitimately gobble their way through the port space.
420     return F;
421
422   # Coarse search for port-scanning candidates: those that have made
423   # connections (attempts) to possible_port_scan_thresh or more
424   # distinct ports.
425   if ( orig !in distinct_ports || service !in distinct_ports[orig] )
426     {
427     if ( orig !in distinct_ports )
428       distinct_ports[orig] = set() &mergeable;
429
430     if ( service !in distinct_ports[orig] )
431       add distinct_ports[orig][service];
432
433     if ( |distinct_ports[orig]| >= possible_port_scan_thresh &&
434       orig !in scan_triples )
435       {
436       scan_triples[orig] = table() &mergeable;
437       add possible_scan_sources[orig];
438       }
439     }
440
441   # Check for low ports.
442   if ( activate_priv_port_check && ! outbound && service < 1024/tcp &&
443        service !in troll_skip_service )
444     {
445     if ( orig !in distinct_low_ports ||
446          service !in distinct_low_ports[orig] )
447       {
448       if ( orig !in distinct_low_ports )
449         distinct_low_ports[orig] = set() &mergeable;
450
451       add distinct_low_ports[orig][service];
452
453       if ( |distinct_low_ports[orig]| == priv_scan_trigger &&
454            orig !in Site::neighbor_nets )
455         {
456         local svrc_msg = fmt("low port trolling %s %s", orig, service);
457         NOTICE([$note=LowPortTrolling, $src=orig,
458           $identifier=fmt("%s", orig),
459           $p=service, $msg=svrc_msg]);
460         }
461
462       if ( ignore_scanners_threshold > 0 &&
463            |distinct_low_ports[orig]| >
464          ignore_scanners_threshold )
465         ignore_addr(orig);
466       }
467     }
```

```
468
469   # For sources that have been identified as possible scan sources,
470   # keep track of per-host scanning.
471   if ( orig in possible_scan_sources )
472     {
473     if ( orig !in scan_triples )
474       scan_triples[orig] = table() &mergeable;
475
476     if ( resp !in scan_triples[orig] )
477       scan_triples[orig][resp] = set() &mergeable;
478
479     if ( service !in scan_triples[orig][resp] )
480       {
481       add scan_triples[orig][resp][service];
482
483       if ( thresh_check_2(report_port_scan, rpts_idx,
484               orig, resp,
485               |scan_triples[orig][resp]|) )
486         {
487         local m = |scan_triples[orig][resp]|;
488         NOTICE([$note=PortScan, $n=m, $src=orig,
489           $p=service,
490           $identifier=fmt("%s-%d", orig, n),
491           $msg=fmt("%s has scanned %d ports of %s",
492           orig, m, resp)]);
493         }
494       }
495     }
496
497   return T;
498   }
499
500
501 # Hook into the catch&release dropping. When an address gets restored, we reset
502 # the source to allow dropping it again.
503 event Drop::address_restored(a: addr)
504   {
505   clear_addr(a);
506   }
507
508 event Drop::address_cleared(a: addr)
509   {
510   clear_addr(a);
511   }
512
513 # When removing a possible scan source, we automatically delete its scanned
514 # hosts and ports.  But we do not want the deletion propagated, because every
515 # peer calls the expire_function on its own (and thus applies the delete
516 # operation on its own table).
517 function remove_possible_source(s: set[addr], idx: addr): interval
518   {
519   suspend_state_updates();
520   delete scan_triples[idx];
521   resume_state_updates();
522
523   return 0 secs;
524   }
525
526 # To recognize whether a certain threshhold vector (e.g. report_peer_scans)
527 # has been transgressed, a global variable containing the next vector index
528 # (idx) must be incremented.  This cumbersome mechanism is necessary because
529 # values naturally don't increment by one (e.g. replayed table merges).
530 function thresh_check(v: vector of count, idx: table[addr] of count,
531       orig: addr, n: count): bool
532   {
533   if ( ignore_scanners_threshold > 0 && n > ignore_scanners_threshold )
534     {
```

```
535     ignore_addr(orig);
536     return F;
537     }
538
539   if ( idx[orig] <= |v| && n >= v[idx[orig]] )
540     {
541     ++idx[orig];
542     return T;
543     }
544   else
545     return F;
546   }
547
548 # Same as above, except the index has a different type signature.
549 function thresh_check_2(v: vector of count, idx: table[addr, addr] of count,
550       orig: addr, resp: addr, n: count): bool
551   {
552   if ( ignore_scanners_threshold > 0 && n > ignore_scanners_threshold )
553     {
554     ignore_addr(orig);
555     return F;
556     }
557
558   if ( idx[orig,resp] <= |v| && n >= v[idx[orig, resp]] )
559     {
560     ++idx[orig,resp];
561     return T;
562     }
563   else
564     return F;
565   }
566
567 event connection_established(c: connection)
568   {
569   local is_reverse_scan = (c$orig$state == TCP_INACTIVE);
570   Scan::check_scan(c, T, is_reverse_scan);
571   }
572
573 event partial_connection(c: connection)
574   {
575   Scan::check_scan(c, T, F);
576   }
577
578 event connection_attempt(c: connection)
579   {
580   Scan::check_scan(c, F, c$orig$state == TCP_INACTIVE);
581   }
582
583 event connection_half_finished(c: connection)
584   {
585   # Half connections never were "established", so do scan-checking here.
586   Scan::check_scan(c, F, F);
587   }
588
589 event connection_rejected(c: connection)
590   {
591   local is_reverse_scan = c$orig$state == TCP_RESET;
592
593   Scan::check_scan(c, F, is_reverse_scan);
594   }
595
596 event connection_reset(c: connection)
597   {
598   if ( c$orig$state == TCP_INACTIVE || c$resp$state == TCP_INACTIVE )
599     # We never heard from one side - that looks like a scan.
600     Scan::check_scan(c, c$orig$size + c$resp$size > 0,
601         c$orig$state == TCP_INACTIVE);
```

```
602   }
603
604 event connection_pending(c: connection)
605   {
606   if ( c$orig$state == TCP_PARTIAL && c$resp$state == TCP_INACTIVE )
607     Scan::check_scan(c, F, F);
608   }
609
610 # Report the remaining entries in the tables.
611 event bro_done()
612   {
613   for ( orig in distinct_peers )
614     scan_summary(distinct_peers, orig);
615
616   for ( orig in distinct_ports )
617     port_summary(distinct_ports, orig);
618
619   for ( orig in distinct_low_ports )
620     lowport_summary(distinct_low_ports, orig);
621   }
```

C.5 Bro Script Scan.bro - Improved Version

```
##! Scan detector ported from Bro 1.x.
##!
##! This script has evolved over many years
##! by several contributors in Bro community.
##! Robin Sommer did most likely the main job porting it from 1.x to 2.x
##!
##! 2013-11-07 - Roger Larsen added FIN & XMAS scan detection capabilities
##! and tuned variables for better slow port scan detection.

@load base/frameworks/notice/main

module Scan;

export {
        redef enum Notice::Type += {
            ## The source has scanned a number of ports.
            PortScan,
            ## The source has scanned a number of addresses.
            AddressScan,
            ## Apparent flooding backscatter seen from source.
            BackscatterSeen,
            ## Summary of scanning activity.
            ScanSummary,
            ## Summary of distinct ports per scanner.
            PortScanSummary,
            ##
            #    SlowPortScanSummary,
            ## Summary of distinct low ports per scanner.
            LowPortScanSummary,
            ## Source reached :bro:id:`Scan::shut_down_thresh`
            ShutdownThresh,
            ## Source touched privileged ports.
            LowPortTrolling,
            ## RL_> added the following lines
            #    ConnectionPartial,
            #    ConnectionAttempt,
            #    ConnectionHalfFinished,
            #    ConnectionRejected,
            #    ConnectionReset,
            #    ConnectionPending,
        };

    # Whether to consider UDP "connections" for scan detection.
    # Can lead to false positives due to UDP fanout from some P2P apps.
    const suppress_UDP_scan_checks = T &redef;

    const activate_priv_port_check = T &redef;
    const activate_landmine_check = F &redef;
    const landmine_thresh_trigger = 5 &redef;

    const landmine_address: set[addr] &redef;

    const scan_summary_trigger = 25 &redef;
    const port_summary_trigger = 20 &redef;
    const lowport_summary_trigger = 10 &redef;

    # Raise ShutdownThresh after this many failed attempts
    const shut_down_thresh = 100 &redef;

    # Which services should be analyzed when detecting scanning
    # (not consulted if analyze_all_services is set).
    const analyze_services: set[port] &redef;
    const analyze_all_services = T &redef;

    # Track address scanners only if at least these many hosts contacted.
```

```
66      const addr_scan_trigger = 0 &redef;
67
68      # Ignore address scanners for further scan detection after
69      # scanning this many hosts.
70      # 0 disables.
71      const ignore_scanners_threshold = 0 &redef;
72
73      # Report a scan of peers at each of these points.
74      const report_peer_scan: vector of count = {
75          10, 20, 100, 1000, 10000, 50000, 100000, 250000, 500000, 1000000,
76      } &redef;
77
78      #
79      const report_outbound_peer_scan: vector of count = {
80          10, 100, 1000, 10000,
81      } &redef;
82
83
84      # Report a scan of ports at each of these points.
85      const report_port_scan: vector of count = {
86          10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 250, 1000, 5000, 10000, 25000, 65000,
87      } &redef;
88
89      # Once a source has scanned this many different ports (to however many
90      # different remote hosts), start tracking its per-destination access.
91      const possible_port_scan_thresh = 10 &redef;
92
93      # Threshold for scanning privileged ports.
94      const priv_scan_trigger = 5 &redef;
95      const troll_skip_service = {
96          25/tcp, 21/tcp, 22/tcp, 20/tcp, 80/tcp,
97      } &redef;
98
99      const report_accounts_tried: vector of count = {
100         10, 20, 100, 1000, 10000, 100000, 1000000,
101     } &redef;
102
103     const report_remote_accounts_tried: vector of count = {
104         10, 100, 500,
105     } &redef;
106
107     # Report a successful password guessing if the source attempted
108     # at least this many.
109     const password_guessing_success_threshhold = 20 &redef;
110
111     const skip_accounts_tried: set[addr] &redef;
112
113     const addl_web = {
114         81/tcp, 443/tcp, 8000/tcp, 8001/tcp, 8080/tcp, }
115     &redef;
116
117     const skip_services = { 113/tcp, } &redef;
118     const skip_outbound_services = { 21/tcp, addl_web, }
119         &redef;
120
121     const skip_scan_sources = {
122         255.255.255.255,    # who knows why we see these, but we do
123     } &redef;
124
125     const skip_scan_nets: set[subnet] = {} &redef;
126
127     # List of well known local server/ports to exclude for scanning
128     # purposes.
129     const skip_dest_server_ports: set[addr, port] = {} &redef;
130
131     # Reverse (SYN-ack) scans seen from these ports are considered
132     # to reflect possible SYN-flooding backscatter, and not true
```

```
133     # (stealth) scans.
134     const backscatter_ports = {
135         80/tcp, 8080/tcp, 53/tcp, 53/udp, 179/tcp, 6666/tcp, 6667/tcp,
136     } &redef;
137
138     const report_backscatter: vector of count = {
139         20,
140     } &redef;
141
142     global check_scan:
143         function(c: connection, established: bool, reverse: bool): bool;
144
145     # The following tables are defined here so that we can redef
146     # the expire timeouts.
147     # FIXME: should we allow redef of attributes on IDs which
148     # are not exported?
149
150     # How many different hosts connected to with a possible
151     # backscatter signature.
152     global distinct_backscatter_peers: table[addr] of table[addr] of count
153         &read_expire = 15 min;
154
155     # Expire functions that trigger summaries.
156     global scan_summary:
157         function(t: table[addr] of set[addr], orig: addr): interval;
158     global port_summary:
159         function(t: table[addr] of set[port], orig: addr): interval;
160     #     global slow_port_summary:
161     #         function(t: table[addr] of set[port], orig: addr): interval;
162     global lowport_summary:
163         function(t: table[addr] of set[port], orig: addr): interval;
164
165     # Indexed by scanner address, yields # distinct peers scanned.
166     # pre_distinct_peers tracks until addr_scan_trigger hosts first.
167     global pre_distinct_peers: table[addr] of set[addr]
168         &read_expire = 15 mins &redef;
169
170     global distinct_peers: table[addr] of set[addr]
171         &read_expire = 15 mins &expire_func=scan_summary &redef;
172
173     global distinct_ports: table[addr] of set[port]
174         &read_expire = 99 mins &expire_func=port_summary &redef;
175
176     #     # RL_> slow port scanning in focus
177     #     global slow_distinct_ports: table[addr] of set[port]
178     #         &read_expire = 99 mins &expire_func=slow_port_summary &redef;
179
180     global distinct_low_ports: table[addr] of set[port]
181         &read_expire = 99 mins &expire_func=lowport_summary &redef;
182
183     # Indexed by scanner address, yields a table with scanned hosts
184     # (and ports).
185     global scan_triples: table[addr] of table[addr] of set[port];
186
187     global remove_possible_source:
188         function(s: set[addr], idx: addr): interval;
189     global possible_scan_sources: set[addr]
190         &expire_func=remove_possible_source &read_expire = 99 mins;
191
192     # Indexed by source address, yields user name & password tried.
193     global accounts_tried: table[addr] of set[string, string]
194         &read_expire = 1 days;
195
196     global ignored_scanners: set[addr] &create_expire = 1 day &redef;
197
198     # These tables track whether a threshold has been reached.
199     # More precisely, the counter is the next index of threshold vector.
```

```
200     global shut_down_thresh_reached: table[addr] of bool &default=F;
201     # report backscatter idx
202     global rb_idx: table[addr] of count
203             &default=1 &read_expire = 1 days &redef;
204     # report peer scan idx
205     global rps_idx: table[addr] of count
206             &default=1 &read_expire = 1 days &redef;
207     # report outbound peer scan idx
208     global rops_idx: table[addr] of count
209             &default=1 &read_expire = 1 days &redef;
210     # report port scan idx
211     global rpts_idx: table[addr,addr] of count
212             &default=1 &read_expire = 1 days &redef;
213     # report ???
214     global rat_idx: table[addr] of count
215             &default=1 &read_expire = 1 days &redef;
216     # report ???
217     global rrat_idx: table[addr] of count
218             &default=1 &read_expire = 1 days &redef;
219 }
220
221 global thresh_check: function(v: vector of count, idx: table[addr] of count,
222                 orig: addr, n: count): bool;
223 global thresh_check_2: function(v: vector of count,
224                 idx: table[addr,addr] of count, orig: addr,
225                 resp: addr, n: count): bool;
226
227 function scan_summary(t: table[addr] of set[addr], orig: addr): interval
228     {
229     local num_distinct_peers = orig in t ? |t[orig]| : 0;
230
231     if ( num_distinct_peers >= scan_summary_trigger )
232         {
233         NOTICE([$note=ScanSummary, $src=orig, $n=num_distinct_peers,
234             $identifier=fmt("%s", orig),
235             $msg=fmt("%s scanned a total of %d hosts",
236                     orig, num_distinct_peers)]);
237         # RL_> Added print line
238         print fmt("Notice ScanSummary: %s scanned a total of %d hosts",
239                     orig, num_distinct_peers);
240         }
241     return 0 secs;
242     }
243
244 function port_summary(t: table[addr] of set[port], orig: addr): interval
245     {
246     local num_distinct_ports = orig in t ? |t[orig]| : 0;
247
248     # RL_>
249     #    slow_distinct_ports[orig] = t[orig];
250
251     if ( num_distinct_ports >= port_summary_trigger )
252         {
253         NOTICE([$note=PortScanSummary, $src=orig, $n=num_distinct_ports,
254             $identifier=fmt("%s", orig),
255             $msg=fmt("%s scanned a total of %d ports",
256                     orig, num_distinct_ports)]);
257         # RL_> Added print line
258         print fmt("Notice PortScanSummary: %s scanned a total of %d ports",
259                     orig, num_distinct_ports);
260         }
261     return 0 secs;
262     }
263
264 #    function slow_port_summary(t: table[addr] of set[port], orig: addr): interval
265 #       {
266 #       local num_distinct_ports = orig in t ? |t[orig]| : 0;
```

```
267
268 #       # RL_>
269 #       #   slow_distinct_ports[orig] = t[orig];
270
271 #       if ( num_distinct_ports >= port_summary_trigger )
272 #           {
273 #           NOTICE([$note=SlowPortScanSummary, $src=orig, $n=num_distinct_ports,
274 #               $identifier=fmt("%s", orig),
275 #               $msg=fmt("%s scanned a total of %d ports slowly",
276 #                       orig, num_distinct_ports)]);
277 #           # RL_> Added print line
278 #           print fmt("Notice slow_port_summary: %s scanned a total of %d ports",
279 #                       orig, num_distinct_ports);
280 #           }
281 #       return 0 secs;
282 #       }
283
284 function lowport_summary(t: table[addr] of set[port], orig: addr): interval
285     {
286     local num_distinct_lowports = orig in t ? |t[orig]| : 0;
287
288     if ( num_distinct_lowports >= lowport_summary_trigger )
289         {
290         NOTICE([$note=LowPortScanSummary, $src=orig,
291             $n=num_distinct_lowports,
292             $identifier=fmt("%s", orig),
293             $msg=fmt("%s scanned a total of %d low ports",
294                     orig, num_distinct_lowports)]);
295         # RL_> Added print line
296         print fmt("Notice LowPortScanSummary: %s scanned a total of %d low ports",
297                     orig, num_distinct_lowports);
298         }
299     return 0 secs;
300     }
301
302 function clear_addr(a: addr)
303     {
304     delete distinct_peers[a];
305     delete distinct_ports[a];
306     #    delete slow_distinct_ports[a];
307     delete distinct_low_ports[a];
308     delete scan_triples[a];
309     delete possible_scan_sources[a];
310     delete distinct_backscatter_peers[a];
311     delete pre_distinct_peers[a];
312     delete rb_idx[a];
313     delete rps_idx[a];
314     delete rops_idx[a];
315     delete rat_idx[a];
316     delete rrat_idx[a];
317     delete shut_down_thresh_reached[a];
318     delete ignored_scanners[a];
319     # RL_> Added print line
320     print fmt("Function clear_addr");
321     }
322 function ignore_addr(a: addr)
323     {
324     clear_addr(a);
325     add ignored_scanners[a];
326     # RL_> Added print line
327     print fmt("Function ignore_addr");
328     }
329
330 function check_scan(c: connection, established: bool, reverse: bool): bool
331     {
332     local id = c$id;
333
```

```
334     local service = "ftp-data" in c$service ? 20/tcp
335             : (reverse ? id$orig_p : id$resp_p);
336     local rev_service = reverse ? id$resp_p : id$orig_p;
337     local orig = reverse ? id$resp_h : id$orig_h;
338     local resp = reverse ? id$orig_h : id$resp_h;
339     local outbound = Site::is_local_addr(orig);
340
341     # The following works better than using get_conn_transport_proto()
342     # because c might not correspond to an active connection (which
343     # causes the function to fail).
344     if ( suppress_UDP_scan_checks &&
345          service >= 0/udp && service <= 65535/udp )
346         return F;
347
348     if ( service in skip_services && ! outbound )
349         return F;
350
351     if ( outbound && service in skip_outbound_services )
352         return F;
353
354     if ( orig in skip_scan_sources )
355         return F;
356
357     if ( orig in skip_scan_nets )
358         return F;
359
360     # Don't include well known server/ports for scanning purposes.
361     if ( ! outbound && [resp, service] in skip_dest_server_ports )
362         return F;
363
364     if ( orig in ignored_scanners)
365         return F;
366
367     if ( ! established &&
368         # not established, service not expressly allowed
369
370         # not known peer set
371         (orig !in distinct_peers || resp !in distinct_peers[orig]) &&
372
373         # want to consider service for scan detection
374         (analyze_all_services || service in analyze_services) )
375         {
376         # RL_> Check if connections are from us, known and/or defined backscattered
377         if ( reverse && rev_service in backscatter_ports &&
378              # reverse, non-priv backscatter port
379              service >= 1024/tcp )
380             {
381             if ( orig !in distinct_backscatter_peers )
382                 {
383                 # RL_> This code empty bs table for actual orig_h
384                 local empty_bs_table:
385                     table[addr] of count &default=0;
386                 distinct_backscatter_peers[orig] =
387                     empty_bs_table;
388                 }
389
390             if ( ++distinct_backscatter_peers[orig][resp] <= 2 &&
391                  # The test is <= 2 because we get two check_scan()
392                  # calls, once on connection attempt and once on
393                  # tear-down.
394
395                  distinct_backscatter_peers[orig][resp] == 1 &&
396
397                  # Looks like backscatter, and it's not scanning
398                  # a privileged port.
399
400                  thresh_check(report_backscatter, rb_idx, orig,
```

```
401                     |distinct_backscatter_peers[orig]|)
402                 )
403                  {
404                  NOTICE([$note=BackscatterSeen, $src=orig,
405                      $p=rev_service,
406                      $identifier=fmt("%s", orig),
407                      $msg=fmt("backscatter seen from %s (%d hosts; %s)",
408                          orig, |distinct_backscatter_peers[orig]|, rev_service)]);
409                  # RL_> Added print line
410                  print fmt("%s - Notice BackscatterSeen: backscatter seen from %s (%d
                         hosts; %s)",
411                      strftime("%y-%m-%d_%H.%M.%S",c$start_time), orig, |
                             distinct_backscatter_peers[orig]|, rev_service);
412                  }
413
414              if ( ignore_scanners_threshold > 0 &&
415                   |distinct_backscatter_peers[orig]| >
416                      ignore_scanners_threshold )
417                  {
418                  ignore_addr(orig);
419                  # RL_> Added print line
420                  print fmt("%s - ignore_scanners_threshold && distinct_backscatter_peers
                         > ignore_scanners_threshold",
421                      strftime("%y-%m-%d_%H.%M.%S",c$start_time));
422                  }
423              }
424
425          else
426              { # done with backscatter check
427              local ignore = F;
428              # RL_> check if addr is new && higher than addr_scan_trigger
429              if ( orig !in distinct_peers && addr_scan_trigger > 0 )
430                  {
431                  if ( orig !in pre_distinct_peers )
432                      #   why this extra definition here?
433                      #   pre_distinct_peers[orig] = set();
434                      add pre_distinct_peers[orig][resp];
435
436                  if ( |pre_distinct_peers[orig]| < addr_scan_trigger )
437                      ignore = T;
438                  }
439
440              if ( ! ignore )
441                  { # XXXXX
442                  if ( orig !in distinct_peers )
443                      distinct_peers[orig] = set() &mergeable;
444
445                  if ( resp !in distinct_peers[orig] )
446                      add distinct_peers[orig][resp];
447
448                  local n = |distinct_peers[orig]|;
449
450                  # Check for shut_down_thresh_reached if not outbound.
451                  if ( ! shut_down_thresh_reached[orig] &&
452                       n >= shut_down_thresh &&
453                       ! outbound &&
454                       orig !in Site::neighbor_nets )
455                      {
456                      shut_down_thresh_reached[orig] = T;
457                      local msg = fmt("shutdown threshold reached for %s", orig);
458                      NOTICE([$note=ShutdownThresh, $src=orig,
459                          $identifier=fmt("%s", orig),
460                          $p=service, $msg=msg]);
461                      # RL_> Added print line
462                      print fmt("%s ShutdownThresh: for host %s", c$start_time, orig);
463                      }
464
```

```
465                 else
466                     {
467                     local address_scan = F;
468                     if ( outbound &&
469                          # inside host scanning out?
470                         thresh_check(report_outbound_peer_scan, rops_idx, orig, n) )
471                             address_scan = T;
472
473                     if ( ! outbound &&
474                         thresh_check(report_peer_scan, rps_idx, orig, n) )
475                             address_scan = T;
476
477                     if ( address_scan )
478                         {
479                         NOTICE([$note=AddressScan,
480                             $src=orig, $p=service,
481                             $n=n,
482                             $identifier=fmt("%s-%d", orig, n),
483                             $msg=fmt("%s has scanned %d hosts (%s)",
484                                 orig, n, service)]);
485                         # RL-> Added print line
486                         print fmt("%s Notice AddressScan: %s has scanned %d hosts (%s)",
487                             strftime("%y-%m-%d_%H.%M.%S",c$start_time), orig, n, service
                                );
488                         }
489                         # RL-> Ignore this scanner address after X hosts scanned?
490                         # RL-> Perhaps enough alarms sendt/rised.
491                     if ( address_scan &&
492                          ignore_scanners_threshold > 0 &&
493                          n > ignore_scanners_threshold )
494                         ignore_addr(orig);
495                     }
496                 }
497             } # XXXX
498         }
499
500     if ( established )
501         # Don't consider established connections for port scanning,
502         # it's too easy to be mislead by FTP-like applications that
503         # legitimately gobble their way through the port space.
504         return F;
505
506     # Coarse search for port-scanning candidates: those that have made
507     # connections (attempts) to possible_port_scan_thresh or more
508     # distinct ports.
509     if ( orig !in distinct_ports || service !in distinct_ports[orig] )
510         #    orig !in slow_distinct_ports || service !in slow_distinct_ports[orig]
511         {
512         if ( orig !in distinct_ports )
513             distinct_ports[orig] = set() &mergeable;
514
515         if ( service !in distinct_ports[orig] )
516             add distinct_ports[orig][service];
517
518         #    if ( orig !in slow_distinct_ports )
519         #        slow_distinct_ports[orig] = set() &mergeable;
520
521         #    if ( service !in slow_distinct_ports[orig] )
522         #        add slow_distinct_ports[orig][service];
523
524         if ( |distinct_ports[orig]| >= possible_port_scan_thresh &&
525             orig !in scan_triples )
526             {
527             scan_triples[orig] = table() &mergeable;
528             add possible_scan_sources[orig];
529             }
530         #    if ( |slow_distinct_ports[orig]| >= possible_port_scan_thresh &&
```

```
531         #       orig !in scan_triples )
532         #       {
533         #       scan_triples[orig] = table() &mergeable;
534         #       add possible_scan_sources[orig];
535         #       }
536
537         }
538
539     # Check for low ports.
540     if ( activate_priv_port_check && ! outbound && service < 1024/tcp &&
541          service !in troll_skip_service )
542         {
543         if ( orig !in distinct_low_ports ||
544              service !in distinct_low_ports[orig] )
545             {
546             if ( orig !in distinct_low_ports )
547                 distinct_low_ports[orig] = set() &mergeable;
548
549             add distinct_low_ports[orig][service];
550
551             if ( |distinct_low_ports[orig]| == priv_scan_trigger &&
552                  orig !in Site::neighbor_nets )
553                 {
554                 local svrc_msg = fmt("low port trolling %s %s", orig, service);
555                 NOTICE([$note=LowPortTrolling, $src=orig,
556                     $identifier=fmt("%s", orig),
557                     $p=service, $msg=svrc_msg]);
558                 # RL_> Added print line
559                 print fmt("%s - Notice LowPortTrolling: %s",
560                     strftime("%y-%m-%d_%H.%M.%S",c$start_time), orig);
561                 }
562
563             if ( ignore_scanners_threshold > 0 &&
564                  |distinct_low_ports[orig]| >
565                     ignore_scanners_threshold )
566                 ignore_addr(orig);
567             }
568         }
569
570     # For sources that have been identified as possible scan sources,
571     # keep track of per-host scanning.
572     if ( orig in possible_scan_sources )
573         {
574         if ( orig !in scan_triples )
575             scan_triples[orig] = table() &mergeable;
576
577         if ( resp !in scan_triples[orig] )
578             scan_triples[orig][resp] = set() &mergeable;
579
580         if ( service !in scan_triples[orig][resp] )
581             {
582             add scan_triples[orig][resp][service];
583
584             if ( thresh_check_2(report_port_scan, rpts_idx,
585                         orig, resp,
586                         |scan_triples[orig][resp]|) )
587                 {
588                 local m = |scan_triples[orig][resp]|;
589                 local rpts_idx_number = rpts_idx;
590                 NOTICE([$note=PortScan, $n=m, $src=orig,
591                     $p=service,
592                     $identifier=fmt("%s-%d", orig, m),
593                     $msg=fmt("%s has scanned %d ports of %s",
594                     orig, m, resp)]);
595                 # RL_> Added print line
596                 #    print fmt("%s - Notice PortScan: %s has scanned %d ports of %s.
                        rpts_idx = %s",
```

```
597                 #       strftime("%y-%m-%d_%H.%M.%S",c$start_time), orig, m, resp,
                        rpts_idx_number);
598                 }
599             }
600         }
601
602     return T;
603     }
604
605
606 # Hook into the catch&release dropping. When an address gets restored, we reset
607 # the source to allow dropping it again.
608 event Drop::address_restored(a: addr)
609     {
610     clear_addr(a);
611     # RL_> Added print line
612     print fmt("Event: drop address_restored");
613     }
614
615 event Drop::address_cleared(a: addr)
616     {
617     clear_addr(a);
618     # RL_> Added print line
619     print fmt("Event: drop address_cleared");
620     }
621
622 # When removing a possible scan source, we automatically delete its scanned
623 # hosts and ports.  But we do not want the deletion propagated, because every
624 # peer calls the expire_function on its own (and thus applies the delete
625 # operation on its own table).
626 function remove_possible_source(s: set[addr], idx: addr): interval
627     {
628     suspend_state_updates();
629     delete scan_triples[idx];
630     resume_state_updates();
631     # RL_> Added print line
632     print fmt("Function: remove_possible_source");
633     return 0 secs;
634     }
635
636 # To recognize whether a certain threshhold vector (e.g. report_peer_scans)
637 # has been transgressed, a global variable containing the next vector index
638 # (idx) must be incremented.  This cumbersome mechanism is necessary because
639 # values naturally don't increment by one (e.g. replayed table merges).
640 function thresh_check(v: vector of count, idx: table[addr] of count,
641     orig: addr, n: count): bool
642     {
643
644     # RL_> Should we ignore this scanner because of enough alarms rised?
645     if ( ignore_scanners_threshold > 0 && n > ignore_scanners_threshold )
646         {
647         ignore_addr(orig);
648         return F;
649         }
650
651     # RL_> Check if we will rise an scan alarm vs report_thesholds.
652     if ( idx[orig] <= |v| && n >= v[idx[orig]] )
653         {
654         # RL_> Increment number of count for this address
655         ++idx[orig];
656         return T;
657         }
658     else
659         return F;
660     }
661
662 # Same as above, except the index has a different type signature.
```

```
663 function thresh_check_2(v: vector of count, idx: table[addr,addr] of count,
664     orig: addr, resp: addr, n: count): bool
665     {
666     if ( ignore_scanners_threshold > 0 && n > ignore_scanners_threshold )
667         {
668         ignore_addr(orig);
669         return F;
670         }
671
672     if ( idx[orig,resp] <= |v| && n >= v[idx[orig, resp]] )
673         {
674         ++idx[orig,resp];
675         return T;
676         }
677     else
678         return F;
679     }
680
681 event connection_established(c: connection)
682     {
683     local is_reverse_scan = (c$orig$state == TCP_INACTIVE);
684
685     Scan::check_scan(c, T, is_reverse_scan);
686     }
687     # RL_> Added notice
688  #       NOTICE([$note=ConnectionPartial, $src=c$id$orig_h,
689  #           $p=c$id$resp_p,
690  #           $msg=fmt("Event: %s have establiskjed a connection on host %s, port %d",
691  #               c$id$orig_h, c$id$resp_h, c$id$resp_p)]);
692     #     print fmt("%s - Event: %s have establiskjed a connection on host %s, port %d",
693  #           strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
      c$id$resp_p);
694     #     }
695
696 event new_connection_contents(c: connection)
697     {
698     local is_reverse_scan = (c$resp$state == TCP_CLOSED );
699     if (( c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED ) &&
700             ( "f" in c$history || "F" in c$history ))
701         {
702         Scan::check_scan(c, F, is_reverse_scan);
703         }
704         # RL_> Added notice
705  #          NOTICE([$note=ConnectionPartial, $src=c$id$orig_h,
706  #              $p=c$id$resp_p,
707  #              $msg=fmt("Event: %s have FIN scanned host %s, port %d",
708  #              c$id$orig_h, c$id$resp_h, c$id$resp_p)]);
709         #     print fmt("%s - Event: %s have FIN scanned host %s, port %d",
710  #           strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
      c$id$resp_p);
711         #     }
712     else if (( c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED ) &&
713                 ( "i" in c$history || "I" in c$history ))
714         {
715         Scan::check_scan(c, F, is_reverse_scan);
716         }
717     }
718         # RL_> Added notice
719  #          NOTICE([$note=ConnectionPartial, $src=c$id$orig_h,
720  #              $p=c$id$resp_p,
721  #              $msg=fmt("Event: %s have FIN scanned host %s, port %d",
722  #              c$id$orig_h, c$id$resp_h, c$id$resp_p)]);
723         #     print fmt("%s - Event: %s have FIN scanned host %s, port %d",
724  #           strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
      c$id$resp_p);
725     #       }
726
```

```
727     #     }
728
729
730
731 event partial_connection(c: connection)
732     {
733     #RL_> line added
734     local is_reverse_scan = (c$resp$state == TCP_PARTIAL );
735
736     if (c$orig$state == TCP_PARTIAL || c$resp$state == TCP_PARTIAL )
737         {
738         Scan::check_scan(c, F, is_reverse_scan);
739         }
740     else
741         {
742         Scan::check_scan(c, T, is_reverse_scan);
743         }
744     }
745         # RL_> orginal line under
746         #   Scan::check_scan(c, T, F);
747         # RL_> Added notice
748  #          NOTICE([$note=ConnectionPartial, $src=c$id$orig_h,
749  #              $p=c$id$resp_p,
750  #              $msg=fmt("%s have performed an partial connection on port %s",
751  #              c$id$orig_h, c$id$resp_p)]);
752  #          # RL_> Added print line
753         #    print fmt("Event: partial_connection");
754     #       }
755     #    # RL_> Roger's ##############################################
756     #    if ( c$orig$state == TCP_RESET || c$resp$state == TCP_RESET )
757     #       {
758     #       #   local is_reverse_scan = (c$resp$state == TCP_RESET);
759     #       Scan::check_scan(c, T, F);
760     #       # RL_> Added notice
761 #          NOTICE([$note=ConnectionPartial, $src=c$id$orig_h,
762 #              $p=c$id$resp_p,
763 #              $msg=fmt("Event: %s may be performing BLABLABLABLA scan to host %s, port
        %d",
764 #                  c$id$orig_h, c$id$resp_h, c$id$resp_p)]);
765     #       print fmt("%s - Event: %s may be performing BLABLABLABLA scan to host %s,
          port %d",
766 #              strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
       c$id$resp_p);
767     #       }
768     #    }
769 event connection_attempt(c: connection)
770     {
771     local is_conn_attempt = (c$orig$state == TCP_INACTIVE);
772     Scan::check_scan(c, F, is_conn_attempt);
773     }
774     # RL_> Added notice
775 #       NOTICE([$note=ConnectionAttempt, $src=c$id$orig_h,
776 #           $p=c$id$resp_p,
777 #           $msg=fmt("%s have attempted connection on by port %s",
778 #           c$id$orig_h, c$id$resp_p)]);
779
780 #       # RL_> Added print line
781     #    print fmt("Event: connection_attempt");
782     #    }
783
784 event connection_half_finished(c: connection)
785     {
786     # Half connections never were "established", so do scan-checking here.
787     Scan::check_scan(c, F, F);
788     }
789     # RL_> Added notice
790 #       NOTICE([$note=ConnectionHalfFinished, $src=c$id$orig_h,
```

```
791 #           $p=c$id$resp_p,
792 #           $msg=fmt("%s have been half finished on by port %s",
793 #           c$id$orig_h, c$id$resp_p)]);
794
795 #       # RL_> Added print line
796     #     print fmt("Event: connection_half_finished");
797     #     }
798
799 event connection_rejected(c: connection)
800     {
801     local is_reverse_scan = c$orig$state == TCP_RESET;
802
803     if (c$orig$state == TCP_CLOSED || c$resp$state == TCP_CLOSED)
804         {
805         Scan::check_scan(c, F, F);
806         }
807
808     else
809         {
810         Scan::check_scan(c, F, is_reverse_scan);
811         }
812     }
813         # RL_> Added notice
814     #     NOTICE([$note=ConnectionRejected, $src=c$id$orig_h,
815     #       $p=c$id$resp_p,
816     #       $msg=fmt("%s have been rejected on by port %s",
817     #       c$id$orig_h, c$id$resp_p)]);
818
819     #     # RL_> Added print line
820     #     print fmt("%s - Event: connection_REJECTED: %s have been rejected by %s port %s
        ",
821     #         strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
        c$id$resp_p);
822     #     }
823
824
825     #     }
826         # RL_> Added notice
827   #         NOTICE([$note=ConnectionReset, $src=c$id$orig_h,
828   #             $p=c$id$resp_p,
829   #             $msg=fmt("%s have too many TCP flags set - possible FIN scan on port %s
      ",
830   #             c$id$orig_h, c$id$resp_p)]);
831
832   #         # RL_> Added print line
833         #     print fmt("%s - Event: %s possible FIN scan on port %s",
834         #         strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_p);
835     #         }
836     #     }
837
838 event connection_reset(c: connection)
839     {
840     if ( c$orig$state == TCP_INACTIVE || c$resp$state == TCP_INACTIVE )
841         {
842         # We never heard from one side - that looks like a scan.
843         # RL_> - should the expresion under with > have had paranthesis ????
844         Scan::check_scan(c, c$orig$size + c$resp$size > 0, c$orig$state == TCP_INACTIVE)
              ;
845         }
846     }
847         # RL_> Added notice
848   #         NOTICE([$note=ConnectionReset, $src=c$id$orig_h,
849   #             $p=c$id$resp_p,
850   #             $msg=fmt("%s have been reset on by port %s",
851   #             c$id$orig_h, c$id$resp_p)]);
852
853   #         # RL_> Added print line
```

```
854         #     print fmt("%s - Event: connection_RESET: Endpoint %s send RST on behalf of
                port %s",
855         #         strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_p);
856
857
858 event connection_pending(c: connection)
859     {
860     if ( c$orig$state == TCP_PARTIAL && c$resp$state == TCP_INACTIVE )
861         {
862         Scan::check_scan(c, F, F);
863         }
864     }
865         #   {
866         # RL_> Added notice
867   #         NOTICE([$note=ConnectionPending, $src=c$id$orig_h,
868   #             $p=c$id$resp_p,
869   #             $msg=fmt("%s are pending connection to %s port %s",
870   #             c$id$orig_h, c$id$resp_h, c$id$resp_p)]);
871
872   #         # RL_> Added print line
873         #     print fmt("%s - Event: connection_pending: %s are pending connection to %s
                port %s",
874         #         strftime("%y-%m-%d_%H.%M.%S",c$start_time), c$id$orig_h, c$id$resp_h,
                c$id$resp_p);
875     #         }
876     #     }
877
878 # Report the remaining entries in the tables.
879 event bro_done()
880     {
881     for ( orig in distinct_peers )
882         scan_summary(distinct_peers, orig);
883
884     for ( orig in distinct_ports )
885         port_summary(distinct_ports, orig);
886
887     #     for ( orig in distinct_ports )
888     #         slow_port_summary(distinct_ports, orig);
889
890     for ( orig in distinct_low_ports )
891         lowport_summary(distinct_low_ports, orig);
892     }
```

C.6 Bro TCP Events - Built In Functiones

```
1  # This file was automatically generated by bifcl from /da/www/git/modules/master/bro/src
       /analyzer/protocol/tcp/events.bif (plugin mode).
2
3
4  ## Generated when reassembly starts for a TCP connection. This event is raised
5  ## at the moment when Bro's TCP analyzer enables stream reassembly for a
6  ## connection.
7  ##
8  ## c: The connection.
9  ##
10 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
11 ##    connection_established connection_external connection_finished
12 ##    connection_first_ACK connection_half_finished connection_partial_close
13 ##    connection_pending connection_rejected connection_reset connection_reused
14 ##    connection_state_remove connection_status_update connection_timeout
15 ##    scheduled_analyzer_applied new_connection partial_connection
16 export {
17 global new_connection_contents: event(c: connection );
18
19
20 ## Generated for an unsuccessful connection attempt. This event is raised when
21 ## an originator unsuccessfully attempted to establish a connection.
22 ## "Unsuccessful" is defined as at least :bro:id:`tcp_attempt_delay` seconds
23 ## having elapsed since the originator first sent a connection establishment
24 ## packet to the destination without seeing a reply.
25 ##
26 ## c: The connection.
27 ##
28 ## .. bro:see:: connection_EOF connection_SYN_packet connection_established
29 ##    connection_external connection_finished connection_first_ACK
30 ##    connection_half_finished connection_partial_close connection_pending
31 ##    connection_rejected connection_reset connection_reused connection_state_remove
32 ##    connection_status_update connection_timeout scheduled_analyzer_applied
33 ##    new_connection new_connection_contents partial_connection
34 global connection_attempt: event(c: connection );
35
36
37 ## Generated when a SYN-ACK packet is seen in response to a SYN packet during
38 ## a TCP handshake.  The final ACK of the handshake in response to SYN-ACK may
39 ## or may not occur later, one way to tell is to check the *history* field of
40 ## :bro:type:`connection` to see if the originator sent an ACK, indicated by
41 ## 'A' in the history string.
42 ##
43 ## c: The connection.
44 ##
45 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
46 ##    connection_external connection_finished connection_first_ACK
47 ##    connection_half_finished connection_partial_close connection_pending
48 ##    connection_rejected connection_reset connection_reused connection_state_remove
49 ##    connection_status_update connection_timeout scheduled_analyzer_applied
50 ##    new_connection new_connection_contents partial_connection
51 global connection_established: event(c: connection );
52
53
54 ## Generated for a new active TCP connection if Bro did not see the initial
55 ## handshake. This event is raised when Bro has observed traffic from each
56 ## endpoint, but the activity did not begin with the usual connection
57 ## establishment.
58 ##
59 ## c: The connection.
60 ##
61 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
62 ##    connection_established connection_external connection_finished
63 ##    connection_first_ACK connection_half_finished connection_partial_close
64 ##    connection_pending connection_rejected connection_reset connection_reused
```

```
65 ##    connection_state_remove connection_status_update connection_timeout
66 ##    scheduled_analyzer_applied new_connection new_connection_contents
67 ##
68 global partial_connection: event(c: connection );
69
70
71 ## Generated when a previously inactive endpoint attempts to close a TCP
72 ## connection via a normal FIN handshake or an abort RST sequence. When the
73 ## endpoint sent one of these packets, Bro waits
74 ## :bro:id:`tcp_partial_close_delay` prior to generating the event, to give
75 ## the other endpoint a chance to close the connection normally.
76 ##
77 ## c: The connection.
78 ##
79 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
80 ##    connection_established connection_external connection_finished
81 ##    connection_first_ACK connection_half_finished connection_pending
82 ##    connection_rejected connection_reset connection_reused connection_state_remove
83 ##    connection_status_update connection_timeout scheduled_analyzer_applied
84 ##    new_connection new_connection_contents partial_connection
85 global connection_partial_close: event(c: connection );
86
87
88 ## Generated for a TCP connection that finished normally. The event is raised
89 ## when a regular FIN handshake from both endpoints was observed.
90 ##
91 ## c: The connection.
92 ##
93 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
94 ##    connection_established connection_external connection_first_ACK
95 ##    connection_half_finished connection_partial_close connection_pending
96 ##    connection_rejected connection_reset connection_reused connection_state_remove
97 ##    connection_status_update connection_timeout scheduled_analyzer_applied
98 ##    new_connection new_connection_contents partial_connection
99 global connection_finished: event(c: connection );
100
101
102 ## Generated when one endpoint of a TCP connection attempted to gracefully close
103 ## the connection, but the other endpoint is in the TCP_INACTIVE state. This can
104 ## happen due to split routing, in which Bro only sees one side of a connection.
105 ##
106 ## c: The connection.
107 ##
108 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
109 ##    connection_established connection_external connection_finished
110 ##    connection_first_ACK  connection_partial_close connection_pending
111 ##    connection_rejected connection_reset connection_reused connection_state_remove
112 ##    connection_status_update connection_timeout scheduled_analyzer_applied
113 ##    new_connection new_connection_contents partial_connection
114 global connection_half_finished: event(c: connection );
115
116
117 ## Generated for a rejected TCP connection. This event is raised when an
118 ## originator attempted to setup a TCP connection but the responder replied
119 ## with a RST packet denying it.
120 ##
121 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
122 ##    connection_established connection_external connection_finished
123 ##    connection_first_ACK connection_half_finished connection_partial_close
124 ##    connection_pending  connection_reset connection_reused connection_state_remove
125 ##    connection_status_update connection_timeout scheduled_analyzer_applied
126 ##    new_connection new_connection_contents partial_connection
127 ##
128 ## c: The connection.
129 ##
130 ## .. note::
131 ##
```

```
132 ##    If the responder does not respond at all, :bro:id:`connection_attempt` is
133 ##    raised instead. If the responder initially accepts the connection but
134 ##    aborts it later, Bro first generates :bro:id:`connection_established`
135 ##    and then :bro:id:`connection_reset`.
136 global connection_rejected: event(c: connection );
137
138
139 ## Generated when an endpoint aborted a TCP connection. The event is raised
140 ## when one endpoint of an established TCP connection aborted by sending a RST
141 ## packet.
142 ##
143 ## c: The connection.
144 ##
145 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
146 ##    connection_established connection_external connection_finished
147 ##    connection_first_ACK connection_half_finished connection_partial_close
148 ##    connection_pending connection_rejected  connection_reused
149 ##    connection_state_remove connection_status_update connection_timeout
150 ##    scheduled_analyzer_applied new_connection new_connection_contents
151 ##    partial_connection
152 global connection_reset: event(c: connection );
153
154
155 ## Generated for each still-open TCP connection when Bro terminates.
156 ##
157 ## c: The connection.
158 ##
159 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
160 ##    connection_established connection_external connection_finished
161 ##    connection_first_ACK connection_half_finished connection_partial_close
162 ##    connection_rejected connection_reset connection_reused connection_state_remove
163 ##    connection_status_update connection_timeout scheduled_analyzer_applied
164 ##    new_connection new_connection_contents partial_connection bro_done
165 global connection_pending: event(c: connection );
166
167
168 ## Generated for a SYN packet. Bro raises this event for every SYN packet seen
169 ## by its TCP analyzer.
170 ##
171 ## c: The connection.
172 ##
173 ## pkt: Information extracted from the SYN packet.
174 ##
175 ## .. bro:see:: connection_EOF  connection_attempt connection_established
176 ##    connection_external connection_finished connection_first_ACK
177 ##    connection_half_finished connection_partial_close connection_pending
178 ##    connection_rejected connection_reset connection_reused connection_state_remove
179 ##    connection_status_update connection_timeout scheduled_analyzer_applied
180 ##    new_connection new_connection_contents partial_connection
181 ##
182 ## .. note::
183 ##
184 ##    This event has quite low-level semantics and can potentially be expensive
185 ##    to generate. It should only be used if one really needs the specific
186 ##    information passed into the handler via the ``pkt`` argument. If not,
187 ##    handling one of the other ``connection_*`` events is typically the
188 ##    better approach.
189 global connection_SYN_packet: event(c: connection , pkt: SYN_packet );
190
191
192 ## Generated for the first ACK packet seen for a TCP connection from
193 ## its *originator*.
194 ##
195 ## c: The connection.
196 ##
197 ## .. bro:see:: connection_EOF connection_SYN_packet connection_attempt
198 ##    connection_established connection_external connection_finished
```

```
199 ##    connection_half_finished connection_partial_close connection_pending
200 ##    connection_rejected connection_reset connection_reused connection_state_remove
201 ##    connection_status_update connection_timeout scheduled_analyzer_applied
202 ##    new_connection new_connection_contents partial_connection
203 ##
204 ## .. note::
205 ##
206 ##    This event has quite low-level semantics and should be used only rarely.
207 global connection_first_ACK: event(c: connection );
208
209
210 ## Generated at the end of reassembled TCP connections. The TCP reassembler
211 ## raised the event once for each endpoint of a connection when it finished
212 ## reassembling the corresponding side of the communication.
213 ##
214 ## c: The connection.
215 ##
216 ## is_orig: True if the event is raised for the originator side.
217 ##
218 ## .. bro:see::  connection_SYN_packet connection_attempt connection_established
219 ##    connection_external connection_finished connection_first_ACK
220 ##    connection_half_finished connection_partial_close connection_pending
221 ##    connection_rejected connection_reset connection_reused connection_state_remove
222 ##    connection_status_update connection_timeout scheduled_analyzer_applied
223 ##    new_connection new_connection_contents partial_connection
224 global connection_EOF: event(c: connection , is_orig: bool );
225
226
227 ## Generated for every TCP packet. This is a very low-level and expensive event
228 ## that should be avoided when at all possible. It's usually infeasible to
229 ## handle when processing even medium volumes of traffic in real-time.  It's
230 ## slightly better than :bro:id:`new_packet` because it affects only TCP, but
231 ## not much. That said, if you work from a trace and want to do some
232 ## packet-level analysis, it may come in handy.
233 ##
234 ## c: The connection the packet is part of.
235 ##
236 ## is_orig: True if the packet was sent by the connection's originator.
237 ##
238 ## flags: A string with the packet's TCP flags. In the string, each character
239 ##        corresponds to one set flag, as follows: ``S`` -> SYN; ``F`` -> FIN;
240 ##        ``R`` -> RST; ``A`` -> ACK; ``P`` -> PUSH.
241 ##
242 ## seq: The packet's TCP sequence number.
243 ##
244 ## ack: The packet's ACK number.
245 ##
246 ## len: The length of the TCP payload, as specified in the packet header.
247 ##
248 ## payload: The raw TCP payload. Note that this may be shorter than *len* if
249 ##          the packet was not fully captured.
250 ##
251 ## .. bro:see:: new_packet packet_contents tcp_option tcp_contents tcp_rexmit
252 global tcp_packet: event(c: connection , is_orig: bool , flags: string , seq: count ,
        ack: count , len: count , payload: string );
253
254
255 ## Generated for each option found in a TCP header. Like many of the ``tcp_*``
256 ## events, this is a very low-level event and potentially expensive as it may
257 ## be raised very often.
258 ##
259 ## c: The connection the packet is part of.
260 ##
261 ## is_orig: True if the packet was sent by the connection's originator.
262 ##
263 ## opt: The numerical option number, as found in the TCP header.
264 ##
```

```
265 ## optlen: The length of the options value.
266 ##
267 ## .. bro:see:: tcp_packet tcp_contents tcp_rexmit
268 ##
269 ## .. note:: There is currently no way to get the actual option value, if any.
270 global tcp_option: event(c: connection , is_orig: bool , opt: count , optlen: count );
271
272
273 ## Generated for each chunk of reassembled TCP payload. When content delivery is
274 ## enabled for a TCP connection (via :bro:id:`tcp_content_delivery_ports_orig`,
275 ## :bro:id:`tcp_content_delivery_ports_resp`,
276 ## :bro:id:`tcp_content_deliver_all_orig`,
277 ## :bro:id:`tcp_content_deliver_all_resp`), this event is raised for each chunk
278 ## of in-order payload reconstructed from the packet stream. Note that this
279 ## event is potentially expensive if many connections carry significant amounts
280 ## of data as then all that data needs to be passed on to the scripting layer.
281 ##
282 ## c: The connection the payload is part of.
283 ##
284 ## is_orig: True if the packet was sent by the connection's originator.
285 ##
286 ## seq: The sequence number corresponding to the first byte of the payload
287 ##      chunk.
288 ##
289 ## contents: The raw payload, which will be non-empty.
290 ##
291 ## .. bro:see:: tcp_packet tcp_option tcp_rexmit
292 ##    tcp_content_delivery_ports_orig tcp_content_delivery_ports_resp
293 ##    tcp_content_deliver_all_resp tcp_content_deliver_all_orig
294 ##
295 ## .. note::
296 ##
297 ##    The payload received by this event is the same that is also passed into
298 ##    application-layer protocol analyzers internally. Subsequent invocations of
299 ##    this event for the same connection receive non-overlapping in-order chunks
300 ##    of its TCP payload stream. It is however undefined what size each chunk
301 ##    has; while Bro passes the data on as soon as possible, specifics depend on
302 ##    network-level effects such as latency, acknowledgements, reordering, etc.
303 global tcp_contents: event(c: connection , is_orig: bool , seq: count , contents: string
      );
304
305
306 ## TODO.
307 global tcp_rexmit: event(c: connection , is_orig: bool , seq: count , len: count ,
     data_in_flight: count , window: count );
308
309
310 } # end of export section
311 module GLOBAL;
```

C.7 Bro - Content of Notice.log, Isolated Scan Session, Scan Detected

```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   notice
#open   2013-11-13-22-42-14
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
    proto   note    msg     sub     src     dst     p       n       peer_descr
    actions policy_items    suppress_for    dropperemote_location.country_code
    remote_location.region  remote_location.city    remote_location.latitude
    remote_location.longitude       metric_index.host       metric_index.str
    metric_index.network
#types  time    string  addr    port    addr    port    enum    enum    string  string
    addr    addr    port    count   string  table[enum]     table[count]    interval
        bool    string  string  string  doubledouble   addr    string  subnet
1384378934.244646       -       -       -       -       -       tcp     Scan::
    LowPortTrolling   low port trolling 192.168.1.200 23/tcp -       192.168.1.200   -
        23      -       bro     Notice::ACTION_LOG    3600.000000       F       -
        -       -       -       -       -       -
1384380497.798781       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 20 ports of 192.168.1.102     -       192.168.1.200   -
        880     20      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384381158.797887       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 30 ports of 192.168.1.102     -       192.168.1.200   -
        9       30      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384381819.897154       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 40 ports of 192.168.1.102     -       192.168.1.200   -
        81      40      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384382480.796665       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 50 ports of 192.168.1.102     -       192.168.1.200   -
        636     50      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384383141.523044       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 60 ports of 192.168.1.102     -       192.168.1.200   -
        19      60      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384383802.623654       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 70 ports of 192.168.1.102     -       192.168.1.200   -
        106     70      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384384463.724233       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 80 ports of 192.168.1.102     -       192.168.1.200   -
        515     80      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384385124.824800       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 90 ports of 192.168.1.102     -       192.168.1.200   -
        497     90      bro     Notice::ACTION_LOG    6       3600.000000       F
        -       -       -       -       -       -       -       -
1384385400.480270       -       -       -       -       -       -       Scan::
    PortScanSummary   192.168.1.200 scanned a total of 99 ports       -
    192.168.1.200   -       -       99      bro     Notice::ACTION_LOG    6
    3600.000000     F       -       -       -       -       -       -       -       -
1384385400.480270       -       -       -       -       -       -       Scan::
    LowPortScanSummary        192.168.1.200 scanned a total of 94 low ports   -
    192.168.1.200   -       -       94      bro     Notice::ACTION_LOG    6
    3600.000000     F       -       -       -       -       -       -       -       -
#close  2013-11-14-00-30-00
```

C.8 Bro - Content of Notice.log, Backscatter Scan Session, Scan Detected

```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   notice
#open   2013-11-24-06-27-13
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
    proto   note    msg     sub     src     dst     p       n       peer_descr
    actions policy_items    suppress_for    dropped remote_location.country_code
    remote_location.region  remote_location.city    remote_location.latitude
    remote_location.longitude       metric_index.host       metric_index.str
    metric_index.network
#types time    string  addr    port    addr    port    enum    enum    string  string
    addr    addr    port    count   string  table[enum]     table[count]    interval
        bool    string  string  string  double
  double addr    string  subnet
1385270833.858184       -       -       -       -       -       tcp     Scan::
    LowPortTrolling   low port trolling 192.168.1.200 993/tcp -       192.168.1.200   -
        993     -       bro     Notice::ACTION_LOG
      6       3600.000000     F       -       -       -       -       -       -       -
        -
1385272333.877256       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 20 ports of 192.168.1.102     -       192.168.1.200   -
        548     20      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385272993.885218       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 30 ports of 192.168.1.102     -       192.168.1.200   -
        280     30      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385273653.893155       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 40 ports of 192.168.1.102     -       192.168.1.200   -
        306     40      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385274313.901415       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 50 ports of 192.168.1.102     -       192.168.1.200   -
        497     50      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385274973.909370       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 60 ports of 192.168.1.102     -       192.168.1.200   -
        311     60      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385275633.917111       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 70 ports of 192.168.1.102     -       192.168.1.200   -
        82      70      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385276293.924452       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 80 ports of 192.168.1.102     -       192.168.1.200   -
        636     80      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385276585.194419       -       -       -       -       -       tcp     Scan::PortScan
    :: has scanned 20 ports of ::   -       ::      -       179     20      bro
    Notice::ACTION_LOG      6       3600.000000
     F      -       -       -       -       -       -       -       -
1385276953.931942       -       -       -       -       -       tcp     Scan::PortScan
    192.168.1.200 has scanned 90 ports of 192.168.1.102     -       192.168.1.200   -
        100     90      bro     Notice::ACTION_LOG      6       3600.000000     F
        -       -       -       -       -       -       -       -
1385277175.605353       -       -       -       -       -       tcp     Scan::PortScan
    :: has scanned 30 ports of ::   -       ::      -       60909   30      bro
    Notice::ACTION_LOG      6       3600.000000
     F      -       -       -       -       -       -       -       -
1385277300.635201       -       -       -       -       -       -       -       Scan::
    PortScanSummary   :: scanned a total of 41 ports -       ::      -       -       41
        bro     Notice::ACTION_LOG      6       3600.000000     F       -       -
        -       -       -       -       -       -
```

```
25 1385277300.635201        -       -       -       -       -       -       Scan::
       PortScanSummary    192.168.1.200 scanned a total of 98 ports       -
       192.168.1.200   -       -       98      bro     Notice::ACTION_LOG      6
       3600.000000     F       -       -       -       -       -       -       -
26 1385277300.635201        -       -       -       -       -       -       Scan::
       LowPortScanSummary          192.168.1.200 scanned a total of 94 low ports   -
       192.168.1.200   -       -       94      bro     Notice::ACTION_LOG      6
       3600.000000     F       -       -       -       -       -       -       -
27 #close  2013-11-24-08-15-00
```

C.9 Bro - Content of Notice.log, Backscatter Scan Session, Scan Not Detected

```
1  #separator \x09
2  #set_separator   ,
3  #empty_field     (empty)
4  #unset_field     -
5  #path    notice
6  #open    2013-11-24-09-58-04
7  #fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
       proto   note    msg     sub     src     dst     p       n       peer_descr
       actions policy_items    suppress_for    dropperemote_location.country_code
       remote_location.region  remote_location.city    remote_location.latitude
       remote_location.longitude       metric_index.host       metric_index.str
       metric_index.network
8  #types  time    string  addr    port    addr    port    enum    enum    string  string
       addr    addr    port    count   string  table[enum]     table[count]    interval
              bool    string  string  string  doubledouble   addr    string  subnet
9  1385283484.875938       -       -       -       -       -       tcp     Scan::PortScan
       :: has scanned 20 ports of ::   -       ::      -       179     20      bro
       Notice::ACTION_LOG      6       3600.000000     -
10 1385284075.140645       -       -       -       -       -       tcp     Scan::PortScan
       :: has scanned 30 ports of ::   -       ::      -       60909   30      bro
       Notice::ACTION_LOG      6       3600.000000     -
11 1385284200.638245       -       -       -       -       -       -       Scan::
       PortScanSummary    :: scanned a total of 41 ports -       ::      -       -       41
              bro     Notice::ACTION_LOG      6       3600.000000     F       -       -
              -       -       -       -       -       -
12 #close   2013-11-24-10-10-00
```

D CAIDA Dataset - About

D.1 The CAIDA description of dataset

The following is from the file README-2013 on CAIDA website. Our dataset did not suffer from packet loss. We have removed some information regarding other datasets having lost packets.

```
The CAIDA Anonymized Internet Traces 2013 Dataset

Overview
--------

This dataset contains anonymized passive traffic traces from CAIDA's passive
monitors in 2013. It contains traffic traces from the 'equinix-chicago' and
'equinix-sanjose' high-speed monitors.

Dataset Contents
----------------
    - trace files (*.pcap.gz): compressed pcap (tcpdump) format traces
    - time files (*.times.gz): contains original nanosecond-precision timestamps.
         The nanosecond timestamps in each *.times.gz line up exactly with the
         packets in the corresponding pcap file (containing timestamps truncated
         to microsecond precision).
    - stats files (*.pcap.stats): statistics on the trace, produced by crl_stats
         (part of the CoralReef suite of tools).
    - file md5.md5: contains md5 checksums for all files

Creation process
----------------

Raw traces were taken on Endace DAG cards with 'dagconvert' (part of Endace dagtools):

  dagconvert -s SNAPLEN -v -V -d DAG_DEVICE -t DURATION -r 2g -T dag:erf -o RAW_TRACE

Raw traces were stripped of payload with:
  crl_to_dag -l4 -Cipfilter='1=1' -o STRIPPED_TRACE RAW_TRACE

Payload-stripped traces were anonymized and split in 1-minute chunks with:
  crl_to_pcap -r -Canon=KEYFILE -Cai -Ci=60 -o ANON_TRACE STRIPPED_TRACE

This resulted in pcap files that only include layer 3 (IPv4 and IPv6) and layer 4
(eg. TCP,UDP,ICMP) headers, with no packet payload.

Traces are named using the following format: {monitor}.{direction}.{start-time}.anon.pcap.gz
```

```
     * monitor: equinix-chicago / equinix-sanjose
     * direction: dirA / dirB
     * start-time: time trace began, format: yyyymmdd-hhmmss.UTC

MD5 checksums were kept with the trace files, and files were checked against
these checksums whenever data was transferred between physical media.

At different stages of the traffic capturing process packet loss can occur, at the end of
this README we try to summarize the different types of losses we detected during various
stages.

Due to the way the monitoring equipment is set up to do time-synchronization
we don't know how well-aligned timestamps between directions of a single link are.

Acceptable Use Agreement
------------------------------

The AUA that you accepted when you were given access to these datas is included
in pdf format as a separate file in the same directory as this README file.

Attribution
-----------

When referencing this dataset (as required by the AUA), please use:
    The CAIDA UCSD Anonymized Internet Traces 2013 - [dates used],
    http://www.caida.org/data/passive/passive_2013_dataset.xml

Users are encouraged to include the following attribution in the
acknowledgments section of their document:
    Support for CAIDA's Internet Traces is provided by the National Science
    Foundation, the US Department of Homeland Security, and
    CAIDA Members.



More Information
----------------

The equinix-chicago OC192 monitor setup:
    http://www.caida.org/data/passive/monitors/equinix-chicago.xml

The Day in the Life of the Internet project:
    http://www.caida.org/projects/ditl/

The CoralReef Software Suite:
```

```
    http://www.caida.org/tools/measurement/coralreef/


Metadata
--------

This section provides some more detailed information on various traces, specifically
on packet loss.
pkts_captured:       Total number of packets that was captured by hardware
pkts_lost_hw:        Number of packets the capturing hardware reported to have
   lost. '>=' indicates that the loss between 2 packets in the traces was more
   then the loss counter can accomodate (65535 packets).
pkts_lost_stripping: Number of packets removed from traces as result of payload stripping step
pkts_lost_anon:      Number of packets lost from traces as result of anonymization step

Note that these numbers are the packet loss we can measure.

---

Event log:

---
```

D.2 CAIDA Dataset - Approved Access

```
-----Opprinnelig melding-----
Fra: Paul Hick [mailto:pphick@caida.org]
Sendt: 20. september 2013 22:36
Til: roger.larsen@hig.no
Kopi: passive-data-access@caida.org
Emne: Re: [passive-data-access] Passive Data Request: Roger Larsen (Gj?vik University College)

Hello Roger Larsen,

Thank you for requesting anonymized Internet trace data from CAIDA.

Your request has been approved

A username and temporary password has been assigned to you.

Your username is your email-address: roger.larsen@hig.no
Your temporary password is         : thasheij

The procedure for accessing the CAIDA data is as follows:

1.) You can use the temporary password to log in to
```

```
        https://data.caida.org/cgi-bin/chpw
    You will be prompted to change your password to a permanent one at that time.
2.) You can use your username and new, permanent password to download datasets from

    https://data.caida.org/datasets/passive-2013/         (Anonymized
2013 Internet Traces)
    https://data.caida.org/datasets/passive-2012/         (Anonymized
2012 Internet Traces)
    https://data.caida.org/datasets/passive-2011/         (Anonymized
2011 Internet Traces)

We've had several people experience difficulties retrieving the data using web browsers. Our gues

For more information on usage of CAIDA data see the CAIDA data usage
FAQ:
    http://www.caida.org/data/data-usage-faq.xml

We rely on you to comply with the Acceptable Use Policies for this data, and report all publicati
    http://www.caida.org/data/publications/bydataset/index.xml
unless you ask that your publication not be included on our website.

We have subscribed you to our passive-data-announce@caida.org mailinglist that we use for announc

=== Anonymized 2013 Internet Traces ===
More information about the 'Anonymized 2013 Internet Traces' can be found at:
    http://www.caida.org/data/passive/passive_2013_dataset.xml
    https://data.caida.org/datasets/passive-2013/README-2013

The README file includes information about packet loss and other notes on a per-trace basis.

=== Anonymized 2012 Internet Traces ===
More information about the 'Anonymized 2012 Internet Traces' can be found at:
    http://www.caida.org/data/passive/passive_2012_dataset.xml
    https://data.caida.org/datasets/passive-2012/README-2012

The README file includes information about packet loss and other notes on a per-trace basis.

=== Anonymized 2011 Internet Traces ===
More information about the 'Anonymized 2011 Internet Traces' can be found at:
    http://www.caida.org/data/passive/passive_2011_dataset.xml
    https://data.caida.org/datasets/passive-2011/README-2011

The README file includes information about packet loss and other notes on a per-trace basis.

===
```

We urge researchers to consider the data carefully and be sure that their use in research is cons

Best Regards,

Paul Hick
Data Administrator
Cooperative Association for Internet Data Analysis

San Diego Supercomputer Center
University of California San Diego
9500 Gilman Drive, La Jolla, CA 92093-0505

Phone: (858) 822-3674
URL : http://www.caida.org/~pphick
Email: pphick@caida.org

On Thu, 2013-09-19 at 12:06 -0700, roger.larsen@hig.no wrote:
> **
> This message was generated by the CAIDA formhandler in response to a
> submission, detailed below.
> **
>
> A Passive Data Request form was submitted by roger.larsen@hig.no on
> Thursday, September 19, 2013 at 12:06:25
>
> Here is a summary of the submission. Event logged as follows:
> ---
> ----- [1957] 12:06:25 09/19/2013 :: roger.larsen@hig.no submitted
> from Roger Larsen (roger.larsen@hig.no) from Gjvik University
> College
> ---
> -----
> DETAILS:
> Status: academic researcher/student,
> First Name: Roger
> Last Name: Larsen
> Institution: Gjvik University College
> Address: Gjvik University College
> Postboks 191
> 2802 Gjvik, Norway
> Phone Number: +47 95237640
> Email: roger.larsen@hig.no
> Position: student,
> Advisor's Name (if student): Slobodan Petrovic Project URL:
> Usage: I present writing a master thesis regarding Bro IDS and port
> scanning. I want to inject computer network traffic to test my

```
> scripts. Thanks
> Datasets: passive-2013,passive-2012,passive-2011
> PhD students:
> MS students: 1
> Other students:
> AUP: RL
> Temporary password: (removed)
> Already have password:
> Subscribe to data-announce: yes
> ------------------------------------------------------------------------
> -----
>
> Errors that occured during submission:
> + No errors detected.
>
>
```

E Snort Configuration

```
#--------------------------------------------------
#   VRT Rule Packages Snort.conf
#
#   For more information visit us at:
#     http://www.snort.org                    Snort Website
#     http://vrt-blog.snort.org/     Sourcefire VRT Blog
#
#     Mailing list Contact:       snort-sigs@lists.sourceforge.net
#     False Positive reports:     fp@sourcefire.com
#     Snort bugs:                 bugs@snort.org
#
#     Compatible with Snort Versions:
#     VERSIONS : 2.9.5.5
#
#     Snort build options:
#     OPTIONS : --enable-gre --enable-mpls --enable-targetbased --enable-ppm --enable-
    perfprofiling --enable-zlib --enable-active-response --enable-normalizer --enable-
    reload --enable-react --enable-flexresp3
#
#     Additional information:
#     This configuration file enables active response, to run snort in
#     test mode -T you are required to supply an interface -i <interface>
#     or test mode will fail to fully validate the configuration and
#     exit with a FATAL error
#--------------------------------------------------

###################################################
# This file contains a sample snort configuration.
# You should take the following steps to create your own custom configuration:
#
#  1) Set the network variables.
#  2) Configure the decoder
#  3) Configure the base detection engine
#  4) Configure dynamic loaded libraries
#  5) Configure preprocessors
#  6) Configure output plugins
#  7) Customize your rule set
#  8) Customize preprocessor and decoder rule set
#  9) Customize shared object rule set
###################################################

###################################################
# Step #1: Set the network variables.  For more information, see README.variables
###################################################

# Setup the network addresses you are protecting
ipvar HOME_NET 192.168.1.0/25

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET
```

```
58
59 # List of sql servers on your network
60 ipvar SQL_SERVERS $HOME_NET
61
62 # List of telnet servers on your network
63 ipvar TELNET_SERVERS $HOME_NET
64
65 # List of ssh servers on your network
66 ipvar SSH_SERVERS $HOME_NET
67
68 # List of ftp servers on your network
69 ipvar FTP_SERVERS $HOME_NET
70
71 # List of sip servers on your network
72 ipvar SIP_SERVERS $HOME_NET
73
74 # List of ports you run web servers on
75 portvar HTTP_PORTS
       [36,80,81,82,83,84,85,86,87,88,89,90,311,383,591,593,631,801,818,901,972,1220,1414,1741,1830,2301,2381,
76
77 # List of ports you want to look for SHELLCODE on.
78 portvar SHELLCODE_PORTS !80
79
80 # List of ports you might see oracle attacks on
81 portvar ORACLE_PORTS 1024:
82
83 # List of ports you want to look for SSH connections on:
84 portvar SSH_PORTS 22
85
86 # List of ports you run ftp servers on
87 portvar FTP_PORTS [21,2100,3535]
88
89 # List of ports you run SIP servers on
90 portvar SIP_PORTS [5060,5061,5600]
91
92 # List of file data ports for file inspection
93 portvar FILE_DATA_PORTS [$HTTP_PORTS,110,143]
94
95 # List of GTP ports for GTP preprocessor
96 portvar GTP_PORTS [2123,2152,3386]
97
98 # other variables, these should not be modified
99 ipvar AIM_SERVERS
       [64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,64.12.163.0/24,64.12.200.0/24,205.188.3.0/24,205.188.5.0/2
100
101 # Path to your rules files (this can be a relative path)
102 # Note for Windows users:  You are advised to make this an absolute path,
103 # such as:  c:\snort\rules
104 var RULE_PATH ./rules
105 var SO_RULE_PATH ../so_rules
106 var PREPROC_RULE_PATH ../preproc_rules
107
108 # If you are using reputation preprocessor set these
109 # Currently there is a bug with relative paths, they are relative to where snort is
110 # not relative to snort.conf like the above variables
111 # This is completely inconsistent with how other vars work, BUG 89986
112 # Set the absolute path appropriately
113 var WHITE_LIST_PATH ./rules
114 var BLACK_LIST_PATH ./rules
115
116 ##################################################
117 # Step #2: Configure the decoder.  For more information, see README.decode
118 ##################################################
119
120 # Stop generic decode events:
```

```
121 config disable_decode_alerts
122
123 # Stop Alerts on experimental TCP options
124 config disable_tcpopt_experimental_alerts
125
126 # Stop Alerts on obsolete TCP options
127 config disable_tcpopt_obsolete_alerts
128
129 # Stop Alerts on T/TCP alerts
130 config disable_tcpopt_ttcp_alerts
131
132 # Stop Alerts on all other TCPOption type events:
133 config disable_tcpopt_alerts
134
135 # Stop Alerts on invalid ip options
136 config disable_ipopt_alerts
137
138 # Alert if value in length field (IP, TCP, UDP) is greater th elength of the packet
139 # config enable_decode_oversized_alerts
140
141 # Same as above, but drop packet if in Inline mode (requires
      enable_decode_oversized_alerts)
142 # config enable_decode_oversized_drops
143
144 # Configure IP / TCP checksum mode
145 config checksum_mode: all
146
147 # Configure maximum number of flowbit references.  For more information, see README.
      flowbits
148 # config flowbits_size: 64
149
150 # Configure ports to ignore
151 # config ignore_ports: tcp 21 6667:6671 1356
152 # config ignore_ports: udp 1:17 53
153
154 # Configure active response for non inline operation. For more information, see REAMDE.
      active
155 # config response: eth0 attempts 2
156
157 # Configure DAQ related options for inline operation. For more information, see README.
      daq
158 #
159 # config daq: <type>
160 # config daq_dir: <dir>
161 # config daq_mode: <mode>
162 # config daq_var: <var>
163 #
164 # <type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
165 # <mode> ::= read-file | passive | inline
166 # <var> ::= arbitrary <name>=<value passed to DAQ
167 # <dir> ::= path as to where to look for DAQ module so's
168
169 # Configure specific UID and GID to run snort as after dropping privs. For more
      information see snort -h command line options
170 #
171 # config set_gid:
172 # config set_uid:
173
174 # Configure default snaplen. Snort defaults to MTU of in use interface. For more
      information see README
175 #
176 # config snaplen:
177 #
178
179 # Configure default bpf_file to use for filtering what traffic reaches snort. For more
      information see snort -h command line options (-F)
180 #
```

```
181 # config bpf_file:
182 #
183
184 # Configure default log directory for snort to log to.  For more information see snort -
        h command line options (-l)
185 #
186 # config logdir:
187
188
189 ##################################################
190 # Step #3: Configure the base detection engine.  For more information, see  README.
        decode
191 ##################################################
192
193 # Configure PCRE match limitations
194 config pcre_match_limit: 3500
195 config pcre_match_limit_recursion: 1500
196
197 # Configure the detection engine  See the Snort Manual, Configuring Snort - Includes -
        Config
198 config detection: search-method ac-split search-optimize max-pattern-len 20
199
200 # Configure the event queue.  For more information, see README.event_queue
201 config event_queue: max_queue 8 log 5 order_events content_length
202
203 ##################################################
204 ## Configure GTP if it is to be used.
205 ## For more information, see README.GTP
206 ##################################################
207
208 # config enable_gtp
209
210 ##################################################
211 # Per packet and rule latency enforcement
212 # For more information see README.ppm
213 ##################################################
214
215 # Per Packet latency configuration
216 #config ppm: max-pkt-time 250, \
217 #   fastpath-expensive-packets, \
218 #   pkt-log
219
220 # Per Rule latency configuration
221 #config ppm: max-rule-time 200, \
222 #   threshold 3, \
223 #   suspend-expensive-rules, \
224 #   suspend-timeout 20, \
225 #   rule-log alert
226
227 ##################################################
228 # Configure Perf Profiling for debugging
229 # For more information see README.PerfProfiling
230 ##################################################
231
232 #config profile_rules: print all, sort avg_ticks
233 #config profile_preprocs: print all, sort avg_ticks
234
235 ##################################################
236 # Configure protocol aware flushing
237 # For more information see README.stream5
238 ##################################################
239 config paf_max: 16000
240
241 ##################################################
242 # Step #4: Configure dynamic loaded libraries.
243 # For more information, see Snort Manual, Configuring Snort - Dynamic Modules
244 ##################################################
```

```
245
246 # path to dynamic preprocessor libraries
247 dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/
248
249 # path to base preprocessor engine
250 dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so
251
252 # path to dynamic rules libraries
253 dynamicdetection directory /usr/local/lib/snort_dynamicrules
254
255 ##################################################
256 # Step #5: Configure preprocessors
257 # For more information, see the Snort Manual, Configuring Snort - Preprocessors
258 ##################################################
259
260 # GTP Control Channle Preprocessor. For more information, see README.GTP
261 # preprocessor gtp: ports { 2123 3386 2152 }
262
263 # Inline packet normalization. For more information, see README.normalize
264 # Does nothing in IDS mode
265 preprocessor normalize_ip4
266 preprocessor normalize_tcp: ips ecn stream
267 preprocessor normalize_icmp4
268 preprocessor normalize_ip6
269 preprocessor normalize_icmp6
270
271 # Target-based IP defragmentation.  For more inforation, see README.frag3
272 preprocessor frag3_global: max_frags 65536
273 preprocessor frag3_engine: policy windows detect_anomalies overlap_limit 10
        min_fragment_length 100 timeout 180
274
275 # Target-Based stateful inspection/stream reassembly.  For more inforation, see README.
        stream5
276 # stream5_globale RL tuned
277 preprocessor stream5_global: track_tcp yes, \
278    track_udp no, \
279    track_icmp no, \
280    max_tcp 262144, \
281    max_udp 131072, \
282    max_active_responses 2, \
283    min_response_seconds 5
284
285 preprocessor stream5_tcp: \
286     policy windows, \
287     detect_anomalies, \
288     require_3whs 180, \
289     overlap_limit 10, \
290     small_segments 3 bytes 150, \
291     timeout 180, \
292     ports client 21 22 23 25 42 53 70 79 109 110 111 113 119 135 136 137 139 143 \
293         161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668
                6669 \
294         7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
295     ports both 36 80 81 82 83 84 85 86 87 88 89 90 110 311 383 443 465 563 591 593 631
            636 801 818 901 972 989 992 993 994 995 1220 1414 1741 1830 2301 2381 2809 3037
            3057 3128 3443 3702 4000 4343 4848 5250 6080 6988 7907 7000 7001 7144 7145 7510
            7802 7777 7779 \
296         7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915
                7916 \
297         7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8181
                8222 8243 8280 8300 8500 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999
                10000 11371 34443 34444 41080 50000 50002 55555
298
299 preprocessor stream5_udp: timeout 180
300
301
302
```

```
303
304 # performance statistics.  For more information, see the Snort Manual, Configuring Snort
        - Preprocessors - Performance Monitor
305 #       RL tuned
306 preprocessor perfmonitor: \
307         time 60 \
308         file /var/log/snort/snort.stats \
309         pktcnt 10000
310
311 # HTTP normalization and anomaly detection.  For more information, see README.
        http_inspect
312 preprocessor http_inspect: global iis_unicode_map unicode.map 1252 compress_depth 65535
        decompress_depth 65535 max_gzip_mem 104857600
313 preprocessor http_inspect_server: server default \
314     http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY POLL BCOPY
          BDELETE BMOVE LINK UNLINK OPTIONS HEAD DELETE TRACE TRACK CONNECT SOURCE
          SUBSCRIBE UNSUBSCRIBE PROPFIND PROPPATCH BPROPFIND BPROPPATCH RPC_CONNECT
          PROXY_SUCCESS BITS_POST CCM_POST SMS_POST RPC_IN_DATA RPC_OUT_DATA RPC_ECHO_DATA
           } \
315     chunk_length 500000 \
316     server_flow_depth 0 \
317     client_flow_depth 0 \
318     post_depth 65495 \
319     oversize_dir_length 500 \
320     max_header_length 750 \
321     max_headers 100 \
322     max_spaces 200 \
323     small_chunk_length { 10 5 } \
324     ports { 36 80 81 82 83 84 85 86 87 88 89 90 311 383 591 593 631 801 818 901 972 1220
          1414 1741 1830 2301 2381 2809 3037 3057 3128 3443 3702 4000 4343 4848 5250 6080
          6988 7000 7001 7144 7145 7510 7777 7779 8000 8008 8014 8028 8080 8085 8088 8090
          8118 8123 8180 8181 8222 8243 8280 8300 8500 8800 8888 8899 9000 9060 9080 9090
          9091 9443 9999 10000 11371 34443 34444 41080 50000 50002 55555 } \
325     non_rfc_char { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 } \
326     enable_cookie \
327     extended_response_inspection \
328     inspect_gzip \
329     normalize_utf \
330     unlimited_decompress \
331     normalize_javascript \
332     apache_whitespace no \
333     ascii no \
334     bare_byte no \
335     directory no \
336     double_decode no \
337     iis_backslash no \
338     iis_delimiter no \
339     iis_unicode no \
340     multi_slash no \
341     utf_8 no \
342     u_encode yes \
343     webroot no
344
345 # ONC-RPC normalization and anomaly detection.  For more information, see the Snort
        Manual, Configuring Snort - Preprocessors - RPC Decode
346 preprocessor rpc_decode: 111 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779
       no_alert_multiple_requests no_alert_large_fragments no_alert_incomplete
347
348 # Back Orifice detection.
349 preprocessor bo
350
351 # FTP / Telnet normalization and anomaly detection.  For more information, see README.
        ftptelnet
352 preprocessor ftp_telnet: global inspection_type stateful encrypted_traffic no
        check_encrypted
353 preprocessor ftp_telnet_protocol: telnet \
354     ayt_attack_thresh 20 \
```

```
355     normalize ports { 23 } \
356     detect_anomalies
357 preprocessor ftp_telnet_protocol: ftp server default \
358     def_max_param_len 100 \
359     ports { 21 2100 3535 } \
360     telnet_cmds yes \
361     ignore_telnet_erase_cmds yes \
362     ftp_cmds { ABOR ACCT ADAT ALLO APPE AUTH CCC CDUP } \
363     ftp_cmds { CEL CLNT CMD CONF CWD DELE ENC EPRT } \
364     ftp_cmds { EPSV ESTA ESTP FEAT HELP LANG LIST LPRT } \
365     ftp_cmds { LPSV MACB MAIL MDTM MIC MKD MLSD MLST } \
366     ftp_cmds { MODE NLST NOOP OPTS PASS PASV PBSZ PORT } \
367     ftp_cmds { PROT PWD QUIT REIN REST RETR RMD RNFR } \
368     ftp_cmds { RNTO SDUP SITE SIZE SMNT STAT STOR STOU } \
369     ftp_cmds { STRU SYST TEST TYPE USER XCUP XCRC XCWD } \
370     ftp_cmds { XMAS XMD5 XMKD XPWD XRCP XRMD XRSQ XSEM } \
371     ftp_cmds { XSEN XSHA1 XSHA256 } \
372     alt_max_param_len 0 { ABOR CCC CDUP ESTA FEAT LPSV NOOP PASV PWD QUIT REIN STOU SYST
          XCUP XPWD } \
373     alt_max_param_len 200 { ALLO APPE CMD HELP NLST RETR RNFR STOR STOU XMKD } \
374     alt_max_param_len 256 { CWD RNTO } \
375     alt_max_param_len 400 { PORT } \
376     alt_max_param_len 512 { SIZE } \
377     chk_str_fmt { ACCT ADAT ALLO APPE AUTH CEL CLNT CMD } \
378     chk_str_fmt { CONF CWD DELE ENC EPRT EPSV ESTP HELP } \
379     chk_str_fmt { LANG LIST LPRT MACB MAIL MDTM MIC MKD } \
380     chk_str_fmt { MLSD MLST MODE NLST OPTS PASS PBSZ PORT } \
381     chk_str_fmt { PROT REST RETR RMD RNFR RNTO SDUP SITE } \
382     chk_str_fmt { SIZE SMNT STAT STOR STRU TEST TYPE USER } \
383     chk_str_fmt { XCRC XCWD XMAS XMD5 XMKD XRCP XRMD XRSQ } \
384     chk_str_fmt { XSEM XSEN XSHA1 XSHA256 } \
385     cmd_validity ALLO < int [ char R int ] > \
386     cmd_validity EPSV < [ { char 12 | char A char L char L } ] > \
387     cmd_validity MACB < string > \
388     cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string > \
389     cmd_validity MODE < char ASBCZ > \
390     cmd_validity PORT < host_port > \
391     cmd_validity PROT < char CSEP > \
392     cmd_validity STRU < char FRPO [ string ] > \
393     cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ] } >
394 preprocessor ftp_telnet_protocol: ftp client default \
395     max_resp_len 256 \
396     bounce yes \
397     ignore_telnet_erase_cmds yes \
398     telnet_cmds yes
399
400
401 # SMTP normalization and anomaly detection.  For more information, see README.SMTP
402 preprocessor smtp: ports { 25 465 587 691 } \
403     inspection_type stateful \
404     b64_decode_depth 0 \
405     qp_decode_depth 0 \
406     bitenc_decode_depth 0 \
407     uu_decode_depth 0 \
408     log_mailfrom \
409     log_rcptto \
410     log_filename \
411     log_email_hdrs \
412     normalize cmds \
413     normalize_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN
          EVFY } \
414     normalize_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND
          SOML } \
415     normalize_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-ERCP X-
          EXCH50 } \
416     normalize_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA
          XTRN XUSR } \
```

```
417     max_command_line_len 512 \
418     max_header_line_len 1000 \
419     max_response_line_len 512 \
420     alt_max_command_line_len 260 { MAIL } \
421     alt_max_command_line_len 300 { RCPT } \
422     alt_max_command_line_len 500 { HELP HELO ETRN EHLO } \
423     alt_max_command_line_len 255 { EXPN VRFY ATRN SIZE BDAT DEBUG EMAL ESAM ESND ESOM
            EVFY IDENT NOOP RSET } \
424     alt_max_command_line_len 246 { SEND SAML SOML AUTH TURN ETRN DATA RSET QUIT ONEX
            QUEU STARTTLS TICK TIME TURNME VERB X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50
            XGEN XLICENSE XQUE XSTA XTRN XUSR } \
425     valid_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY }
            \
426     valid_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SOML
            } \
427     valid_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-ERCP X-EXCH50
            } \
428     valid_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA
            XTRN XUSR } \
429     xlink2state { enabled }
430
431 # Portscan detection. For more information, see README.sfportscan
432 # preprocessor sfportscan: proto  { all } memcap { 10000000 } sense_level { low }
433 # RL tuned
434 preprocessor sfportscan:\
435         proto  { tcp } \
436         scan_type { all } \
437         memcap { 10000000 } \
438         logfile { /var/log/snort/sfportscan-alert.log } \
439         sense_level { high } \
440         detect_ack_scans
441
442 # ARP spoof detection. For more information, see the Snort Manual - Configuring Snort -
        Preprocessors - ARP Spoof Preprocessor
443 # preprocessor arpspoof
444 # preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00
445
446 # SSH anomaly detection. For more information, see README.ssh
447 preprocessor ssh: server_ports { 22 } \
448                   autodetect \
449                   max_client_bytes 19600 \
450                   max_encrypted_packets 20 \
451                   max_server_version_len 100 \
452                   enable_respoverflow enable_ssh1crc32 \
453                   enable_srvoverflow enable_protomismatch
454
455 # SMB / DCE-RPC normalization and anomaly detection. For more information, see README.
        dcerpc2
456 preprocessor dcerpc2: memcap 102400, events [co ]
457 preprocessor dcerpc2_server: default, policy WinXP, \
458     detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
459     autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], \
460     smb_max_chain 3, smb_invalid_shares ["C$", "D$", "ADMIN$"]
461
462 # DNS anomaly detection. For more information, see README.dns
463 preprocessor dns: ports { 53 } enable_rdata_overflow
464
465 # SSL anomaly detection and traffic bypass. For more information, see README.ssl
466 preprocessor ssl: ports { 443 465 563 636 989 992 993 994 995 7801 7802 7900 7901 7902
        7903 7904 7905 7906 7907 7908 7909 7910 7911 7912 7913 7914 7915 7916 7917 7918 7919
         7920 }, trustservers, noinspect_encrypted
467
468 # SDF sensitive data preprocessor. For more information see README.sensitive_data
469 preprocessor sensitive_data: alert_threshold 25
470
471 # SIP Session Initiation Protocol preprocessor. For more information see README.sip
472 preprocessor sip: max_sessions 40000, \
```

```
473     ports { 5060 5061 5600 }, \
474     methods { invite \
475               cancel \
476               ack \
477               bye \
478               register \
479               options \
480               refer \
481               subscribe \
482               update \
483               join \
484               info \
485               message \
486               notify \
487               benotify \
488               do \
489               qauth \
490               sprack \
491               publish \
492               service \
493               unsubscribe \
494               prack }, \
495     max_uri_len 512, \
496     max_call_id_len 80, \
497     max_requestName_len 20, \
498     max_from_len 256, \
499     max_to_len 256, \
500     max_via_len 1024, \
501     max_contact_len 512, \
502     max_content_len 2048
503
504 # IMAP preprocessor.  For more information see README.imap
505 preprocessor imap: \
506     ports { 143 } \
507     b64_decode_depth 0 \
508     qp_decode_depth 0 \
509     bitenc_decode_depth 0 \
510     uu_decode_depth 0
511
512 # POP preprocessor. For more information see README.pop
513 preprocessor pop: \
514     ports { 110 } \
515     b64_decode_depth 0 \
516     qp_decode_depth 0 \
517     bitenc_decode_depth 0 \
518     uu_decode_depth 0
519
520 # Modbus preprocessor. For more information see README.modbus
521 preprocessor modbus: ports { 502 }
522
523 # DNP3 preprocessor. For more information see README.dnp3
524 preprocessor dnp3: ports { 20000 } \
525     memcap 262144 \
526     check_crc
527
528 # Reputation preprocessor. For more information see README.reputation
529 preprocessor reputation: \
530     memcap 500, \
531     priority whitelist, \
532     nested_ip inner, \
533     whitelist $WHITE_LIST_PATH/white_list.rules, \
534     blacklist $BLACK_LIST_PATH/black_list.rules
535
536 ##################################################
537 # Step #6: Configure output plugins
538 # For more information, see Snort Manual, Configuring Snort - Output Modules
539 ##################################################
```

```
540
541 # unified2
542 # Recommended for most installs
543 # output unified2: filename merged.log, limit 128, nostamp, mpls_event_types,
      vlan_event_types
544 #       output unified2: filename snort.log, limit 128
545
546 # Additional configuration for specific types of installs
547 # output alert_unified2: filename snort.alert, limit 128, nostamp
548 # output log_unified2: filename snort.log, limit 128, nostamp
549
550 # syslog
551 output alert_syslog: LOG_AUTH LOG_ALERT
552
553 # pcap
554 # output log_tcpdump: tcpdump.log
555
556 # metadata reference data.  do not modify these lines
557 include classification.config
558 include reference.config
559
560
561 ##################################################
562 # Step #7: Customize your rule set
563 # For more information, see Snort Manual, Writing Snort Rules
564 #
565 # NOTE: All categories are enabled in this conf file
566 ##################################################
567
568 # site specific rules
569 include $RULE_PATH/local.rules
570
571
572 ##################################################
573 # Step #8: Customize your preprocessor and decoder alerts
574 # For more information, see README.decoder_preproc_rules
575 ##################################################
576
577 # decoder and preprocessor event rules
578 # include $PREPROC_RULE_PATH/preprocessor.rules
579 # include $PREPROC_RULE_PATH/decoder.rules
580 # include $PREPROC_RULE_PATH/sensitive-data.rules
581
582 ##################################################
583 # Step #9: Customize your Shared Object Snort Rules
584 # For more information, see http://vrt-blog.snort.org/2009/01/using-vrt-certified-shared
      -object-rules.html
585 ##################################################
586
587 # dynamic library rules
588 # include $SO_RULE_PATH/bad-traffic.rules
589 # include $SO_RULE_PATH/chat.rules
590 # include $SO_RULE_PATH/dos.rules
591 # include $SO_RULE_PATH/exploit.rules
592 # include $SO_RULE_PATH/icmp.rules
593 # include $SO_RULE_PATH/imap.rules
594 # include $SO_RULE_PATH/misc.rules
595 # include $SO_RULE_PATH/multimedia.rules
596 # include $SO_RULE_PATH/netbios.rules
597 # include $SO_RULE_PATH/nntp.rules
598 # include $SO_RULE_PATH/p2p.rules
599 # include $SO_RULE_PATH/smtp.rules
600 # include $SO_RULE_PATH/snmp.rules
601 # include $SO_RULE_PATH/specific-threats.rules
602 # include $SO_RULE_PATH/web-activex.rules
603 # include $SO_RULE_PATH/web-client.rules
604 # include $SO_RULE_PATH/web-iis.rules
```

```
605 # include $SO_RULE_PATH/web-misc.rules
606
607 # Event thresholding or suppression commands. See threshold.conf
608 include threshold.conf
609 2013-11-27 13:16:56 root@snorty:~ ] #
610 2013-11-27 13:16:58 root@snorty:~ ] #
611 2013-11-27 13:16:58 root@snorty:~ ] #
612 2013-11-27 13:16:58 root@snorty:~ ] #
613 2013-11-27 13:16:58 root@snorty:~ ] #
614 2013-11-27 13:16:59 root@snorty:~ ] #
615 2013-11-27 13:16:59 root@snorty:~ ] #
616 2013-11-27 13:16:59 root@snorty:~ ] #
```