# Using Netflows for slow portscan detection

Bjarte Malmedal

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2005

Institutt 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

# ABSTRACT

Most organizations that have a defined security strategy implement some kind of detection capability. These systems often focus on real-time analysis of security related events in the information systems. Signature-based systems need to inspect every byte of the network traffic. Capturing, storing and analyzing all this traffic for future analysis is very resource consuming. There is thus a need for alternative ways of detecting misuses that span long periods of time.

One alternative to inspecting each byte of the packet content is to analyze the metadata about each logical connection; i.e. source, destination, port numbers and packet length combined with the timestamp. The metadata for one logical connection is called a Netflow.
By limiting the scope of data collection, it becomes possible to search through the traffic data for longer timespans, and to discover trends that a traditional intrusion detection system cannot do. One type of misuse is particularly difficult to detect for the traditional "real-time" intrusion detection systems, namely slow portscans which are performed by introducing latency between each individual packet.

This thesis aims to investigate if Netflow analysis is more suitable for detecting slow portscans than two traditional systems for intrusion detection, Snort and Network Flight Recorder.

**Sammendrag**
**(Abstract in Norwegian)**

Organisasjoner som har en definert sikkerhetsstrategi har ofte implementert systemer for inntrengningsdeteksjon. Slike løsninger fokuserer som regel på sann-tids analyse av sikkerhetstruende hendelser i informasjonssystemene. Signaturbaserte inntrengnings systemer må inspisere hver eneste byte som sendes gjennom nettverket. Å samle inn, lagre og håndtere store mengder trafikkdata for fremtidig analyse er svært ressurskrevende. Man trenger derfor alternative metoder for deteksjon av misbruk i datanettverk som pågår over lang tid.

Et alternativ til å inspisere innholdet i hver nettverkspakke er å analysere metadata om hver logiske forbindelse; det vil si avsender og mottaker adresse, port nummer, pakkelengde og tidsstempel. Slike metadata kalles en Netflow.
Ved å begrense omfanget av datainnsamlingen blir det mulig å lagre data over lengre tid. Dette muliggjør deteksjon av trender som vanlige systemer for inntrengningsdeteksjon ikke kan oppdage. En type hendelse som er svært vanskelig å oppdage for sanntidssystemer er sakte portscan. Sakte portscan utføres ved å introdusere forsinkelse mellom hver individuelle pakke.

Denne masteroppgaven har som mål å undersøke om Netflow analyse er bedre egnet til å detektere sakte portscan enn to tradisjonelle systemer for inntrengningsdeteksjon, Snort og Network Flight Recorder.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# SUMMARY OF RESULTS

This thesis has demonstrated that one can build a Netflow analysis system based on Argus and PostgreSQL, and that this system can be used for intrusion detection.

Several misuse scenarios where we believe that a Netflow analysis system could be used for misuse detection are described. One of the scenarios, slow portscan detection, is particularly difficult to detect for ordinary intrusion detection systems. Slow portscan detection was selected for further study.

By analyzing how the Netflow system, Snort and Network Flight Recorder detect slow portscanning, we were able to make a ranking according to the true positive/false positive ratio at detecting slow portscans. The analysis concludes that the Netflow analysis system tops this ranking. Thus, the research hypothesis was confirmed.

An experiment was designed and conducted to answer the question :

*Has Netflow analysis a higher or lower true positive/false positive ratio for detecting slow portscanning than Snort and Network Flight Recorder?*

The hypothesis was that Netflow analysis would have a lower true positive/false positive ratio. The experiment performed in this thesis confirmed this hypothesis.

# PREFACE

The author is employed by the Norwegian Defense Security Agency/Critical Infrastructure Protection Center (FSA/FSKI), and works as senior instructor at the Norwegian Army's University College of engineering.

Over the last years, the Norwegian Defence Security Agency has prepared a strategy for protecting critical infrastructure. Security monitoring is a vital part of this strategy.
The Norwegian Defense faces challenges that require accurate and cost-effective solutions. Our assets have consequences for national security. The threat agents are real, with capacity for advanced Computer Network Operations. Our military units are deployed over a large geographic area, and they re-locate several times during exercises and operations. The communication lines vary from narrow-band to broad-band, and local personnel are not necessarily information security specialists. We need cost-effective solutions that provide strong indications of misuse in our critical infrastructure. The Norwegian Defense Security Agency focuses on deploying overlapping and complimentary detection regimes to consolidate indications of misuse.
A proactive approach to this challenge is to investigate what we can accomplish with existing infrastructure and low-cost security monitoring solutions. Session data can be generated by systems similar to the one used in this thesis, or by common network equipment. If session data analysis can provide sufficient indications of misuses, such systems may very well have a natural place in our strategy for security monitoring.

This thesis is a part of our work to investigate what we can detect with a low-cost and high-granularity data source. A system similar to the one described in this thesis has been deployed during national military exercises, and the results have been very positive.

**Acknowledgements**

I have been fortunate enough to receive help and support in all phases of the thesis. I would like to thank Prof. Slobodan Petrović for supervising the work on this thesis, and Prof. Einar Snekkenes for supervising the planning. My good friend Oddbjorn Steffensen is always a good source for interesting discussions about information security. A special thanks to Oddbjorn for introducing me to Argus and Netflow analysis. I would also like to thank Anton Chuvakin, Ph.D for his great help on various topics in this thesis. Last, I would like to thank my employer, the Norwegian Defense Security Agency for giving me time and equipment to use in this research.

## About the author

Bjarte Malmedal has a bachelor degree in computer science from the University College of Bergen, Norway. He lectures information security at the Norwegian Army's University College, and participates in projects for securing Norwegian Defense information infrastructure. He is also a member of a team of security specialists that carry out information security inspections at Norwegian Military Units.

Bjarte Malmedal
Lillehammer, 1 June 2005

# 1 INTRODUCTION

## 1.1 Topic

This thesis investigates if Netflow[1] analysis can be used for slow portscan detection, and describes some scenarios of misuses that can be detected based on Netflow data.
Key words are technical information security, network security, and intrusion detection.

## 1.2 Problem description

Most commercially available Intrusion Detection Systems are signature based. As a result, these systems can only detect the misuses they already know. Unfortunately, new types of misuse emerge almost every day. It may be necessary to check past traffic against new signatures to find out if we were attacked *before* we learned about the new attack. This is not an easy task. Even the most modest network can produce enormous amounts of data [4, 14], that must be stored and handled for as long as detecting past misuses still has a value to the security analyst. For example, if some new attack emerges, and it is clear that it has been active for several weeks, one might want to know if one was attacked before the new IDS signature was developed. To do this, one needs to capture and store all network traffic for as long as it is historically interesting to find out whether one was attacked or not.

To solve this problem, we must look at ways to reduce the amount of data we need to handle. One such reduction is to collect Netflow records only. A Netflow record is minuscule in comparison to a network packet, and only one netflow-record is recorded for each TCP-session or UDP-stream. But of course, all content and most information regarding the individual packet is lost in the reduction.

The problem treated in this thesis may be stated as follows:
*Is it possible to detect slow portscanning in computer network based on Netflow data, and if so, can Netflow analysis do this better than other methods?*

## 1.3 Justification, motivation and benefits

### The need for intrusion detection

Detecting intrusions and misuses has become a necessity for many organizations. It is generally accepted that preventing the unwanted is either too difficult and/or too expensive. In a cost-effectiveness perspective, it can be a viable solution to let some misuses slip past the preventive measures, and instead detect and react to them as they occur. Though many intrusion detection systems are focusing on detecting events in real time, there is also a need for detecting events in historic data.

[1]A Netflow is a record of which IP-addresses communicate, what ports they use, how much data they exchange and so forth. Netflow data can be collected from various sources like Cisco products [43], ARGUS (Audit Record Generation and Usage System) [34], SANCP (Security Analyst Network Connection Profiler) [29] and other systems. Please note that this thesis deals with Argus Netflows, not Cisco NetFlows. The Netflow record is described further in Section 5.2

**Using Netflow data in misuse detection**

One should be able to search trough historic activities for signs of misuse. Storing all network traffic is usually not a viable option, but when the data volume is reduced to session meta-data it becomes a manageable amount of data. By doing this, it is possible to keep data for a longer period of time. This enables us to apply data-mining techniques and discover trends one otherwise could not discover.
By knowing what types of misuses this approach can detect, it is possible to design a detection strategy that has the most cost-effective mix of traditional systems and Netflow analysis systems.

The reduced data volume is one obvious reason for using Netflow analysis. Another is that Netflow data can easily be stored in databases, making it easier and more efficient to study the material and apply data mining techniques.

**Intrusion detection benefits from using the Netflow analysis approach**

Signature based detection systems are usually very good at detecting what is already known. They can usually tell the operator exactly what type of intrusion it has detected, its parameters/context, its impact and so on. (Of course, the quality of the alert depends heavily on the quality of signatures). But what will the system do when it faces a completely new attack or a new version of a worm? In these cases, anomaly detection is expected to give the solution. Anomaly detection is a wide class of methods and techniques where we somehow know or learn the nature of the normal, non-attacked state. By knowing this we can detect behavior that deviates from the normal state. When using these techniques, it does not matter if a new version of a worm is released, as long as it makes the system behave differently. Encryption, which is a show-stopper for most signature based systems, does not necessarily affect the anomaly detection techniques at all. By studying the Netflow data for anomalies, it may be possible to detect events that are not yet known to traditional intrusion detection systems.

**The importance of slow portscan detection**

In general, portscanning is a minor event. When monitoring the outside of the network perimeter, one usually sees lots of portscanning activity. Worms and other malware do automatic scanning, searching for services and vulnerabilities. So called *'scriptkiddies'* have nothing better to do than to scan the entire address-space on the internet. One is tempted to just filter away this activity, and forget about it. However, some scanning has characteristics that deserve our attention. One of them is so-called slow portscanning, where the scanner pauses for a long time between the individual packets it sends out. The reason for doing this is obviously to evade potential intrusion detection measures. Such scanning can take weeks or months to complete per scanned host, and is thus more costly for the scanner. If someone is willing to pay this cost to evade our detection measures, we should do what we can to detect him.

**The Netflow approach helps privacy issues**

Storing and managing information about the users actions in a computer network is a concern to the government, the organization and the individual. By removing the specifics of the network session, an abstraction layer between the user and the transactions he is involved in is added. Even though one can still establish who communicates with whom, when and how much, one cannot reveal passwords, credit card numbers or other sensitive content.

**The Netflow approach can provide higher return on investment**

Economy is often a limiting factor when implementing a detection strategy. If one is certain that a Netflow analysis system can detect the misuses that are of interest, then it may provide a higher return on investment than other intrusion detection systems. Existing infrastructure may be able to generate Netflows, and the analysis system is made of inexpensive parts compared to commercial IDS systems.

**Benefactors**

Stakeholders are IDS operators, security managers and other people responsible for computer network security. If it is possible to show that Netflow analysis can detect slow portscanning and other types of misuse, this can aid the stakeholders in choosing the right products at the right places. This can lead to increased security and more cost-effective detection capabilities.

## 1.4 Research questions

To be able to say whether Netflow analysis can be used for misuse detection, the characteristics of misuse need to be defined. And to be able to say what types of misuse Netflow analysis can detect, one needs to use a classification of misuses.
This leads to the following research questions:

1. To what degree can Netflow analysis be used for misuse detection?
2. Analyze how Netflow analysis, Snort[41] and Network Flight Recorder[42] (NFR) detect slow portscanning. Find limitations in the three methods, and do a ranking.

   *Hypothesis: Netflow analysis will top this ranking.*
3. Has Netflow analysis a higher or lower true positive/false positive [3] ratio for detecting slow portscanning than Snort and Network Flight Recorder?

   *Hypothesis: Netflow Analysis has a lower true positive/false positive ratio than the two other systems.*

## 1.5 Characteristics of misuse in computer networks

Most definitions that are presented in Section 2.1 seem to encompass what is known as *attacks* or *intrusions*. These terms may imply that an attacker has a predetermined goal, and violates mechanisms that otherwise would prevent him from achieving this goal.
*Misuse* on the other hand, is a somewhat broader term. It embraces *Attacks* and *Intrusions* but it also includes other unwanted activities that do not necessarily violate mechanisms. For example, introducing a private computer onto a corporate network is not necessarily an attack. But it may be a violation of security policy. Users who are engaged in network gaming that exhaust network resources are not attacking the network, but they may be violating the security policy.

As it can be seen in Section 1.7, a Netflow analysis system can probably detect security policy violations as well as some *attacks* and *intrusions*.

## 1.6 Misuse taxonomy in computer networks

### 1.6.1 The need for classification

Classification refers to recognizing, naming, and describing units or elements to be classified. The objective of all classifications is the orderly arrangement of a large array of objects so that their differences and similarities can be better understood.
Classifying the events detected by the IDS is necessary in order to aid the operator to quickly assess the situation. Usually different classes of events are given different priorities in order to further aid the operator by deciding in which order the events should be handled.

### 1.6.2 Misuse taxonomies

Quite a lot of work has been done in the field of classifying misuses. A number of these are presented in Section 2.2, State of the art.

Attack-centric taxonomies describe the misuse from the attackers point of view. This may seem as a good idea, because the event may include the attacker's intentions and goals, as well as the methods he uses. The DARPA Intrusion Detection System Evaluation [9] uses this type of taxonomy. The events are categorized according to the attacker's objectives:

1. Denial of service.
2. Remote to user.
3. User to super-user.
4. Surveillance/probing.

The events are further classified by the methods used:

**m:** masquerading (stolen password or forged IP address)

**a:** abuse of a feature

**b:** implementation bug

**c:** system misconfiguration

**s:** social engineering.

In [10, 11] McHugh discusses some weaknesses in taxonomies of this type. One of the problems is that the target (or the IDS that protects the target) will often see only parts of the attack. This can make it very difficult to categorize misuses. Let us use a buffer overflow attack as an example:
Assume that some service has a buffer overflow vulnerability. The vulnerability enables an attacker to run an arbitrary piece of code on the target. It is not uncommon that one can use many different payloads in an attack like this. The Metasploit [31] tool enables the operator to use almost any payload with any buffer overflow exploit. Payload examples includes "Install NetCat[2] and listen on port x", "open a connection back to the attacker on port y", "mail the password-file to address z", "provide a super-user shell at port u", "shut down service or host (DoS)" and so on. The problem here is that the target may observe the buffer overflow exploitation only. Because of this, all of the above attack scenarios may trigger the same alert, despite the fact that the goals and consequences are quite different.

The taxonomies that take the defenders point of view are built around the assets. Web-attack taxonomies, mail-attack taxonomies and DNS-attack taxonomies are some examples. By relating the taxonomy to the assets, there is a possibility that it will aid the security operator to assess the criticality and consequences of the event more quickly. Network Flight Recorder, NetForensics [40] and Cisco IDS [43] are security products that mostly categorize events from the defenders point of view.

### 1.6.3 Netflow taxonomy

The Netflow record contains information from the Network and Transport layer in the communication stack. In [10, 11], McHugh proposes that one should develop taxonomies that categorize events according to the protocol layers. All network activities leave traces in the Network and/or Transport layers. Some misuses will leave strong indicators there, for example the Land-attack[3]. Other misuses may leave weaker indicators, for example a successful connection to a port commonly used by a backdoor program. Some misuses will not leave indicators at all, for example a buffer overflow attack on a web-server. The taxonomy should include all misuse categories that leave strong indicators in the Netflow records. It would also be interesting to investigate if it is possible to strengthen weak indicators by applying data-mining and other techniques to the Netflow records. Scenarios of misuse that may be detected with Netflow analysis are discussed further in Section 1.7.

In Chapter 7 we argue that one of the issues that must be taken into consideration when developing a detection strategy is the nature of available logs and data sources. Economy is usually a limiting factor, so it would be of great importance to know if the kind of misuse we are interested in can be detected with the data sources we already have. For example: If we are certain that Netflow analysis is sufficient for detecting all misuses we care about, and our existing infrastructure can generate such flows, it may not be necessary to purchase an expensive IDS. This requires that there are taxonomies

[2]Netcat is a networking utility which reads and writes data across network connections, using the TCP/IP protocol. http://netcat.sourceforge.net/

[3]A Land Attack is a specially crafted IP packet, where the source address is the same as the destination address. Some TCP-stack implementations cannot handle these packets

that categorize misuse in respect to the data source in which they can be detected.

## 1.7 Introduction to misuse detection with Netflow analysis

The aim of this section is to describe scenarios or types of misuses that can probably be detected by Netflow analysis. It neither results in a complete taxonomy, nor in a complete list of misuses. It is important to stress that while some of the misuse detection has been demonstrated in previous work and actual security systems, some has not undergone studies. One should therefore treat the list of misuses in this section as a suggestion for future study.

Using Netflows for misuse detection is still a new and immature field. As argued in Section 1.6 there is a need for taxonomies that categorize misuses that can be detected by Netflow analysis systems. This section will not present a taxonomy, but the scenarios of misuse are grouped into two categories. One category where one may find strong indicators of misuse in the Netflow records, and one where one may find weaker indicators. There is not a defined line between the two categories. One one end, one may find misuses that Netflow analysis will detect, e.g. the previously mentioned Land attack. The entire attack exists in the Netflow tuples, and can thus be detected by Netflow analysis. On the other end, one may find misuses that leave almost no trace in the Netflow records at all, e.g. a buffer overflow in some network service. In this case the attack may not be seen at all, only the fact that the two hosts are involved. Other misuses may be found somewhere in between.
In this thesis we use the following definitions:

**Strong indicators.**

Indicators found in Netflow records that alone or in combination with other indicators give a warning of misuse with a high probability.

**Weak indicators.**

Indicators found in Netflow records that can be used in combination with other indicators to increase the probability of a warning, but cannot alone provide sufficient indication of misuse.

### 1.7.1 Strong indicators

We begin by presenting some misuses that are believed to leave strong indicators in the Netflow records.

#### Configuration changes

One should be able to detect if someone introduces or removes a unit on the network. One should also be able to detect if someone introduces a new service or makes use of an unusual protocol. In [23], Fullmer and Romig propose a system that collects Cisco NetFlows for this purpose.
In [22], Peter Troxell et al. describe The Network Security Analysis Tool (NSAT) for the same purpose. In [5, 6, 13] William Yurcik et al. describe a prototype tool also used for this purpose.

**Reconnaissance**

One should be able to detect both horizontal and vertical portscanning, and all scanning techniques (ICMP, UDP, TCP SYN, FIN, NULL and so on). In [5, 6, 13] William Yurcik et al. describe a prototype tool that can detect network probing. Since one should be able to store Netflow records for a long period of time, we believe that it is possible to detect slow portscanning as well. The aim of our experiment was to answer this hypothesis.

**Malicious code**

There has been an increasing number of worms that attack network services directly, instead of using for example mail as a carrier. By monitoring activity at "known worm ports" one can discover if (1) There was a successful connection to these ports (the host got infected) and (2) There are outgoing activities at these ports (the host is infecting others). Another important feature with the Netflow system is that once the security operator learns about a new type of worm, he can check historic data for signs of this worm on his network.
In [5, 6, 13] William Yurcik et al. describe a prototype tool for tracing worm propagation.

**Denial of Service (DoS)/Distributed DoS**

Some Denial of Service attacks can probably be detected with Netflow analysis. Some DoS attacks are executed by filling the communication line, or exhausting resources in the Network or Transport layer. These types of DoS could possibly be detected in the Netflow records.The kind where the DoS happens in the higher layers (e.g. a buffer overflow DoS attack) will probably not be detected with Netflow analysis.

**Compromise**

Even if the actual compromise cannot be detected (for example a buffer overflow attack) there is still the possibility that the compromise will result in strong indicators in the Netflow records.

Examples indications of compromised hosts:

- There should be no outgoing connections from our webserver. If such connections are observed, there is a good chance that the webserver is compromised.
- There should be no connections to ports other that port 80 on our webserver. If such connections are observed, there is a good chance that the webserver is compromised.

The security analyst should have a clear picture of the legitimate services on his network. All other successful connections should be treated as indications of misuse.

In [23] Fullmer and Romig monitor changes in network activity levels. If a normally quiet FTP server suddenly explodes with activity, this can mean that it is compromised and used for unwanted activities such as "Warez" distribution.
In [5, 6, 13], William Yurcik et al. describe a prototype tool for visualization of network traffic for the same purpose.

**Security policy violations**

There may be activities that one would not classify as an attack or intrusion, but that are unwanted either as a whole, or at designated times. Some examples are network gaming, Peer-to-peer file sharing, massive ftp downloads that exhaust the bandwidth and Internet

Relay Chat (IRC) and other Instant Messaging (IM) services.

### 1.7.2 Weak indicators

There is also a weaker set of indicators in the Netflow records. These are indicators that are sometimes traces of misuse, and sometimes not. It is possible that they cannot be used for misuse detection on their own, but combined with other indicators they can strengthen the suspicion of misuse.

**Covert channels**

In [44], a covert channel is defined to be: (1) A transmission channel that may be used to transfer data in a manner that violates security policy. (2) An unintended and/or unauthorized communications path that can be used to transfer information in a manner that violates an information system security policy.

Examples of misuses that include the use of covert channels:

- Stacheldraht[4] is a DDoS tool that uses ICMP for communication and control messages.
- Tribal Flood Network[5] uses ICMP echo reply packets for control messages.
- httptunnel[6] channel arbitrary protocols through a legitimate web channel (used for evading firewalls).

The Netflow records include the number of packets and bytes sent by the source and destination host. Stacheldraht commands in ICMP packets may deviate in size and number from what one would expect to see in our network, but it is possible that the indicator is too weak by itself. There is also a possibility that Instant Messaging channelled through a legitimate web channel has properties that are different from normal web-usage. We suggest that this undergoes future studies.

**Malicious code**

There is a possibility that one can use packet and byte count in the Netflow records to strengthen the suspicion that a given session is in fact a worm and not legitimate traffic.

[4]http://staff.washington.edu/dittrich/misc/stacheldraht.analysis
[5]http://staff.washington.edu/dittrich/misc/tfn.analysis
[6]http://www.gnu.org/software/httptunnel/httptunnel.html

# 2 REVIEW OF THE STATE OF THE ART

## 2.1 Characteristics of misuse in computer networks

In [1], D. Denning presents a model for real-time intrusion detection that is based on the hypothesis that exploitation of a system's vulnerabilities involves abnormal use of the system and that security violations can be detected from abnormal patterns in system usage. The article outlines metrics that can be used to measure the state, and thus to determine if the state is normal or abnormal. The problem is of course that there may be very many states in computer network or computer system. It may be difficult to know if a particular state is normal or abnormal. One particular state, or set of states, may be normal in one information system while abnormal in another.

In [12], Bierman et al. define a computer intrusion to be *any set of actions that attempt to compromise the Confidentiality, Integrity or Availability (abbreviated CIA) of a resource*. In [24], R. Heady et al. give the same definition. This may seem to be a high granularity definition. The 'CIA' triangle of security attributes may be the fundament for all other attributes and mechanisms, but one would expect to find non-repudiation and authenticity in the definition as well. Policy violations should also be classified as a security intrusion/misuse.

In [32], the following definition of an intrusion is found:
A security event or a combination of multiple security events that constitutes a security incident in which an intruder gains, or attempts to gain access to a system (or system resource) without having authorization to do so.

In [21], Ulf Lindquist defines an intrusion to be "... a successful event from the attacker's point of view and consists of:

1. an attack in which a vulnerability is exploited, resulting in
2. a breach which is a violation of the explicit or implicit security policy of the system."

This definition should be extended to encompass all violations of the security policies, not just the ones where a vulnerability is exploited (i.e. a policy may state that browsing racist or pornographic web-sites is a violation of the security policy. Per se it is not an exploited vulnerability.)

## 2.2 Classification of misuse in computer networks

There are many taxonomies for intrusions and misuse [1,9,15-20]. Some are classified from the attacker's point of view [1, 9, 16, 18, 20] (attack type or vector), others from the defenders point of view (consequences of the attack). Some try to encompass the entire attack: the tool used, the vulnerability exploited, the action taken, the target that is attacked and the result of the attack[19, 20], even broadening the classification including the attacker and his objective [15, 17].

A Netflow record contains very limited information about the events that take place on the network. It seems that most existing taxonomies require the security analyst or intrusion detection system to possess more information than can be found in the Netflow records. Sometimes, aggregated or in some way limited information are all there is. It may be of interest to know what types of intrusions/misuse are possible to detect given a data source (network traffic, process-calls, Netflows, web-logs and so forth).

## 2.3 Netflow analysis used in intrusion detection

In [23], Fullmer et al. present a tool set for collecting and analyzing Cisco Netflow data. The tools are capable of detecting anomalies in the network such as new network hosts or new services appearing. They can also detect changes in the usage profiles of the hosts/services. One particular type of misuse mentioned in the article is scanning and flooding. The article does not mention storing data in a relational database, thus benefiting from powerful SQL-queries. Nor does it mention visualization as a method for enhancing the usability of Netflow analysis.

In [22], Peter Troxell et al. describe the Network Security Analysis Tool (NSAT), developed for the US Air Force. The tool does not collect Cisco or Argus [34] Netflows, but builds a similar record from listening to the TCP/IP traffic. In traffic analysis mode, the tool can determine traffic patterns that can be matched against known patterns for misuse/intrusions detection.

In [5, 6, 13], William Yurcik et al. describe a prototype tool for visual data mining. The concept is to enhance the intrusion detection process by visualizing the state of the computer network. The source data are Cisco Netflow audit logs. Usage examples include identifying compromised machines involved in:

1. uploading/downloading unapproved software (high traffic levels)
2. hosting Internet relay chat (IRC) servers
3. worm/virus propagation
4. network and host probing
5. hosting remotely installed 'bots' that are remote controlled (traffic patterns on unusual ports).

Network and host probing are mentioned, but no comparison to other detection methods is made.

Security Analyst Network Connection Profiler (SANCP) [29] is a network security tool designed to collect statistical information regarding network traffic, as well as collect the traffic itself in pcap[1] format. This is similar to Argus which is used in this thesis. When SANCP is paired with Sguil [30], the session records are stored in a MySQL database. In this thesis, we use a similar approach. SANCP is used to keep a long-time situational awareness of the activity in the network. This means that when the security analyst becomes aware of some new attack, he can query the database to see if this attack has

[1]Pcap is a library for recording link-layer frames in a promiscuous fashion from network interface drivers.

happened *before* he (and signature-based IDSs) learned about it.

In [25], Axelsson use Netflow data for network security auditing. [25] focuses on collecting network audit logs that can be used in computer security forensics. Axelsson uses Argus to build the audit logs, and shows that it is possible to detect certain cases of misuse in a computer network. [25] is clearly related to this thesis and shows that an Argus Netflow system can be used in a high-speed network.

Flowscan [28] analyzes and reports on Internet Protocol (IP) flow data exported by routers and produces graph images that provide a continuous, near real-time view of the network border traffic. It visualizes time-series data in 48 hour windows and uses the Round Robin Database[2] for storing the NetFlows. Compared to the Netflow system used in this thesis, Flowscan has a much smaller time window and does not support the ability to do SQL-queries in the data source.

## 2.4 Detecting slow portscanning by means of Netflow analysis and other methods.

*Hypothesis: Netflow analysis will top this ranking.*

The comparison of various slow portscanning detection methods has not been treated exclusively in the literature. In [8], Juan M. Estevez-Tapiador et al. describe a case study where Snort is used to detect portscanning. But no comparison to Netflow analysis is made.

## 2.5 A comparison of true positive/false positive rate obtained by Netflow analysis, Snort and Network Flight Recorder.

*Hypothesis: Netflow Analysis has a lower true positive/false positive ratio than the two other methods.*

The comparison of false positive ratios obtained by Netflow analysis, Snort and Network Flight Recorder has not been treated exclusively in the literature.

[2]http://people.ee.ethz.ch/˜oetiker/webtools/rrdtool/index.en.html

# 3 SUMMARY OF CLAIMED CONTRIBUTIONS

For this thesis, a Netflow analysis system was set up from open source tools like Argus [34], PostgreSQL [35], php [37] and jpgraph [36]. Some scenarios where Netflow analysis may be suitable for misuse detection were presented. From these, slow portscan detection was selected for further study.
Two systems were chosen to be tested against the Netflow analysis system. Snort [41], a popular open source tool for network intrusion detection, and Network Flight Recorder [42], a highly advanced commercial tool for network intrusion detection. The slow portscan capabilities for each system were analyzed, and an experiment was conducted to validate the result from this analysis.

Main results in this thesis are:

1. Results from the analysis of how Netflow analysis is compared with Snort and Network Flight Recorder on slow portscan detection.
2. Results from the experiment where Netflow analysis is compared with Snort and Network Flight Recorder on slow portscan detection.
3. An overview of scenarios where Netflow analysis can probably be used for misuse detection.

# 4 CHOICE OF METHODS

A mixed approach is used in this thesis: Literature studies, laboratory experiments and field experiments. The choice of methods is described in relation to each research question.

## 4.1 Netflow analysis used in intrusion detection

By analyzing the attributes of Netflow data, it should be possible to describe a number of scenarios of misuse that Netflow analysis can probably detect. The amount of information in Netflow data is limited, thus the ability to detect intrusions/misuse is probably also limited. An example could be buffer overflow detection. Usually this is detected by inspecting packets for a known sequence of bytes. This information is not present in Netflow data, and cannot be detected in Netflow analysis. A Land Attack on the other hand *can* be detected by Netflow analysis as both source and destination address are present in Netflow data.

## 4.2 Detecting slow portscanning by means of Netflow analysis and other methods.

*Hypothesis: Netflow analysis will top this ranking.*

Both Snort and Network Flight Recorder have mechanisms for port detecting scanning activities. These mechanisms are implemented as signatures/algorithms. By analyzing these, one can identify what limitations they have for detecting slow portscanning. The Netflow analysis system was exposed to the same study. The idea was that by comparing these findings, it should be possible to rank Netflow analysis in respect to the two other methods.

## 4.3 A comparison of true positive/false positive rate obtained by Netflow analysis, Snort and Network Flight Recorder.

*Hypothesis: Netflow Analysis has a lower false positive ratio than the other methods.*

To answer the former research question, a theoretical approach was used to analyze the mechanisms that implement slow portscan detection in the Netflow analysis system, Snort and Network Flight Recorder. To validate the analysis, an experiment will be designed and performed to provide answer to how well a Netflow analysis system, Snort and Network Flight Recorder detect a slow vertical portscan.

The results are presented visually as ROC curves later in this thesis. ROC curves are explained in Section 6.3.

A schematic overview of the experiment setup is presented in Section 6.2.

1. ***Lab experiment***
   The scanner was implemented on a personal computer running FreeBSD 4.9 [39], and using *nmap* as the portscanning application. nmap has several configurable scanning options. One is the time-interval between individual packets in the scan. This time-interval was set to 60 seconds in this experiment. Scanning all TCP ports with this setting would take approximately 11 days. The experiment consisted of multiple iterations. To be able to conduct the entire experiment within the time available, the scan was limited to the first 1000 TCP ports.
   As an effort to ensure reliability in the experiment, a known data set from *The 1999 DARPA off-line intrusion detection evaluation* was used as background noise during the scan.
   The TopLayer 3531 IDS Load balancer ensured that all three systems were exposed to the same network traffic.
   The purpose of the experiment was to find out if the different detection mechanisms can detect slow portscans, and to determine the true positive/false positive ratio.

2. ***Field experiment***
   The scanner was implemented as in the lab experiment. As an effort to ensure validity for the experiment, a data set from a production network with 200 users was used as background noise in the field experiment.
   One should be aware that one does not control all the variables in the field experiment. There may be real portscanning activity during the experiment, and this may affect the true positive/false positive ratio. This can be mitigated as the IP address for the scanner is known. Any "real" portscans can thus be removed from the experiment, unless the real scanner is scanning a very small set of ports. In this case it may be very difficult to distinguish false positives from true positives.
   The purpose of the experiment is to determine if the results from the lab experiment still apply when using a different data-set for background noise.

# 5 SLOW PORTSCAN DETECTION

## 5.1 Slow portscan detection with Netflow analysis, Snort and Network Flight Recorder

In this section, we analyze the portscan detection capabilities and limitations in the Netflow analysis system, Snort and Network Flight Recorder. This results in a ranking of how well these systems perform in the experiment.

In [2], Axelsson presents a taxonomy of intrusion detection systems. When using Axelsson's taxonomy, the systems used in the experiment in this thesis are categorized as follows[1]:

| System | Type | Comment |
|---|---|---|
| Snort | Real-time, Network IDS, Programmed, Threshold | Snort enumerates targeted ports during a time-interval. If the threshold is exceeded, snort will produce an alarm. |
| Network Flight Recorder | Real-time, Network IDS, Programmed, Threshold | NFR enumerates targeted ports during a time-interval. NFR can be configured to give designated ports a higher weight. If the threshold is exceeded, NFR will produce an alarm. The ability to assign weight to different classes of destination ports causes NFR to be ranked above Snort. |
| Netflow system | Non Real-time, Network IDS, Programmed, Threshold | The Netflow system stores all Netflows in a database. It enumerates targeted ports by querying the database. Effective time-interval equals time-span of records in the database. If the threshold is exceeded, the Netflow system will produce an alarm. Additional techniques can be used to strengthen the threshold indicator. |

Table 1: Categorization of systems used in this thesis

When comparing the three systems, it is clear that they are not that different when it comes to portscan detection. The main difference is that the Netflow system is non real-time, and that it operates with a much wider timeout-window. As shown in the analysis that follows, the wider timeout-window gives the Netflow analysis system an advantage over the other systems.

Another issue is the real-time vs. non real-time property. Snort and NFR are real-time

[1]Snort and Network Flight Recorder have capabilities far beyond portscan detection. They employ many different techniques, and may be categorized differently according to the different techniques they use. This categorization will only examine the portscan detection methods.

systems, and will produce an alarm as soon as the threshold is exceeded inside the time-out window. The real-time properties in the Netflow system are limited by latency for the insertion of the records into the database, and by the latency for the database queries. Pushing the records into the database and querying it more often will reduce this latency, but it cannot eliminate it. The size of the database and the speed of the server limits how much this latency can be reduced.

### 5.1.1 Ranking

Sections 5.2-5.4 look closer at the techniques the systems use for portscan detection and the limitations they have regarding slow portscan detection. The following is a short summary and a ranking of how well the three systems should detect slow portscans:

| Rank | System | Rationale |
|---|---|---|
| 1 | Netflow analysis system | The Netflow analysis system takes a step back and views a bigger picture. By only storing meta-data about network activity it is able to keep a very large time-window. This is essential for detecting slow portscans, and makes this system top this ranking |
| 2 | Network Flight Recorder | NFR enumerates targeted ports during a time-interval. NFR can be configured to give designated ports a higher weight. This means that a security analyst can use his expertise and intuition to give 'improbable' ports a higher weight, and thus reduce the chance of false positives. NFR has one serious limitation. The default time-window is far too small to detect slow portscans. When this window is increased, NFRs ability to detect other misuses is believed to decrease because of higher CPU cycles and memory consumption. |
| 3 | Snort/sfportscan | Snort/sfportscan enumerate RST-packets coming from a target during a time-interval. By doing this, Snort reduces the chance of false positives generated by legitimate traffic. Sfportscan's timeout-window vary from 60 to 600 seconds, and should thus be able to detect the portscanning in this thesis. |

Table 2: A ranking of the systems used in this thesis

## 5.2 The use of the Netflow system in slow portscan detection

The Netflow analysis system presented in this section is a prototype developed for this thesis. It uses Argus to build records of metadata about the logical connections in the computer network. The records are stored in a PostgreSQL database.

The system consists of three modules: one module for collecting netflows, one module for storing netflows and one module for presenting the findings. Figure 1 shows the internal structure of the Netflow analysis system

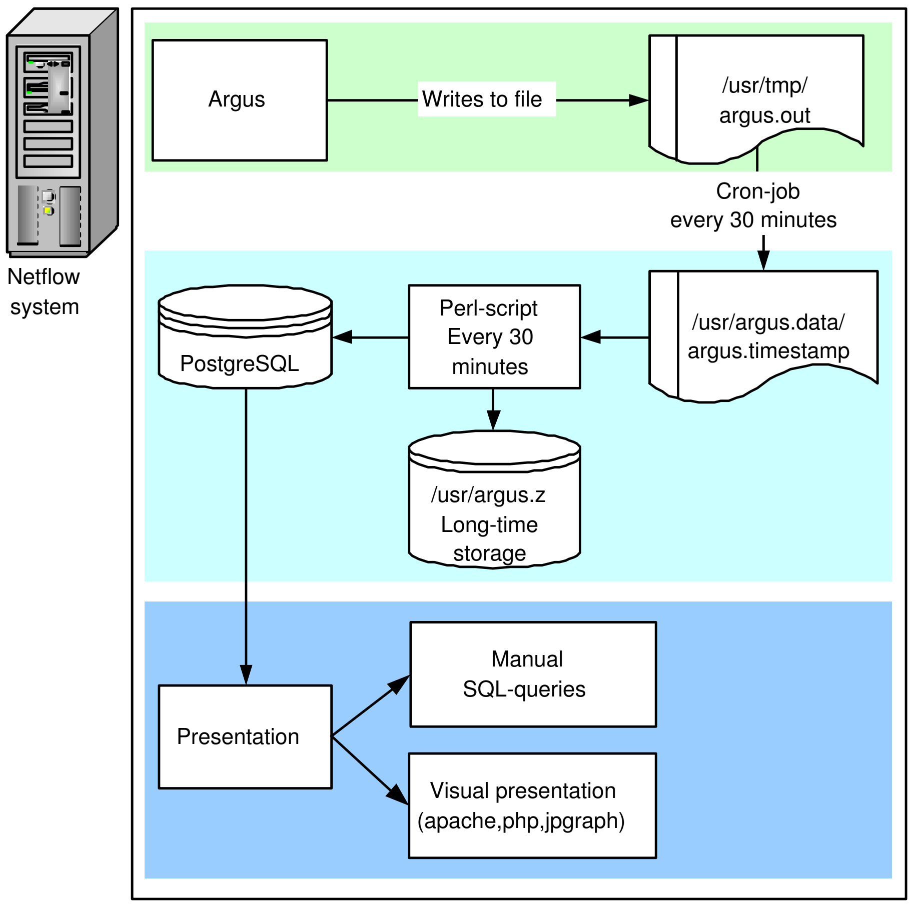
Netflow system
Argus
Writes to file
/usr/tmp/ argus.out
Cron-job every 30 minutes
PostgreSQL
Perl-script Every 30 minutes
/usr/argus.data/ argus.timestamp
/usr/argus.z Long-time storage
Presentation
Manual SQL-queries
Visual presentation (apache,php,jpgraph)

Figure 1: The Netflow system

The Netflow collecting module consists of Argus, which monitors the network interface and builds the netflows. By default, it collects netflows for 30 minutes before the capture-file is moved for further processing. This interval should be configured to the speed of the network to make sure the files are of a reasonable size.
The Netflow storing module is a perl-script [38] that uses *ra*[2] for reading the argus-file. The script then inserts the Netflow-records into a PostgreSQL database. The argus-file is also compressed and written to a long-time storage for future processing. Currently only TCP, UDP and ICMP sessions are stored in the database.
The presentation module allows the operator to query the database, either manually, or by presenting the query as text on a web-page or visually as images.

Table 3 describes the Netflow record used in this thesis.

| Column | Data type | Description |
|---|---|---|
| time | timestamp | Indicates the timestamp at the start of the session |
| proto | character(4) | Protocol used in the session. Currently, the Netflow system records ICMP, UDP and TCP sessions. |
| src_ip | inet | IP address that started the session. |
| src_port | integer | Source port (UDP or TCP). For ICMP this field is NULL. |
| dir | character(7) | Graphical symbols that indicate the direction of the traffic. |
| dst_ip | inet | IP address that was contacted by src_ip. |
| dst_port | integer | Destination port (UDP or TCP. For ICMP this field is NULL. |
| src_count | integer | Number of packets sent by src_ip during the session. |
| dst_count | integer | Number of packets sent by dst_ip during the session. |
| src_bytes | integer | Number of bytes sent by src_ip during the session. |
| dst_bytes | integer | Number of bytes sent by dst_ip during the session. |
| state | character(7) | Indicates session state. (TIM=Timeout, RST=Reset, CON=Connection) |

Table 3: Netflow record description

[2]*ra* reads argus data from either stdin, an argus-file, or from a remote argus-server, filters the records it encounters based on an optional filter-expression and either prints the contents of the argus records that it encounters to stdout or writes them out into an argus data-file.

### 5.2.1 Portscan detection in the Netflow analysis system

Argus records all sessions that pass through the network interface card in promiscuous mode. This means that one record is created for every port the scanner tries to access. By querying the database, the Netflow system should provide a list of how many ports each individual source address has tried to access. An example of a query is given below:

*SELECT src_ip,count(distinct dst_port) FROM flow GROUP BY src_ip ORDER BY count(distinct dst_port) DESC;*

The query produces the following output:

```
     src_ip      | count
-----------------+-------
 172.26.240.100  |   999
 172.16.114.148  |    11
 196.37.75.158   |     7
 194.7.248.153   |     7
 194.27.251.21   |     7
 197.182.91.233  |     7
 172.16.114.168  |     7
 172.16.113.84   |     6
```

The query lists each source IP, and how many individual ports it has tried to access. The query does not discriminate successful connection attempts from unsuccessful. All types of portscanning (UDP, TCP SYN, FIN, NULL, XMAS etc.) are detected. By modifying the query, one could also detect horizontal scanning as well as vertical. Vertical portscanning refers to scanning one host for all active services. Horizontal portscanning refers to scanning multiple hosts for one particular service

To reduce false positives from hosts with large numbers of services, all records that represent successful connections are removed from the query:

*SELECT src_ip,count(distinct dst_port) FROM flow WHERE state !='CON' GROUP BY src_ip ORDER BY count(distinct dst_port) DESC;*

This query produces the following output:

```
     src_ip      | count
-----------------+-------
 172.26.240.100  |   999
 172.16.114.148  |    10
 194.7.248.153   |     7
 172.16.114.168  |     7
 194.27.251.21   |     7
 196.37.75.158   |     7
 195.115.218.108 |     6
 135.13.216.191  |     6
```

To further reduce false positives one can exploit the assumption that a scan will have a regular delay between each individual packet, while normal network sessions will not. By calculating the standard deviation for the time delay between each individual session it may be possible to distinguish between an automated scan and normal use. The standard deviation $\sigma$ is given by

$$\sigma = \sqrt{\mu_2} = \sqrt{\sum (x - \mu_1)^2}$$

Where $x$ is the sample, $\mu_1$ is the mean of all samples and $\mu_2$is the sample variance.

Theoretically, a scan performed by for example *nmap* should have a low standard deviation while normal use should have a higher standard deviation. However, this approach is vulnerable. An intelligent scanner can easily evade this by introducing a random delay between the packets or similar that distorts the calculations. Normal network traffic can in some cases be very regular, and thus have a low standard deviation score. This indicator is probably weaker than counting the number of destination ports each source has tried to contact, but can be used to strengthen a suspicion of portscanning activity.

To further aid the security analyst in detecting slow portscanning in computer networks, one can use visualization methods. The Netflow record contains time-series data that can be used for visual trend-analysis.
For portscan detection, one can make an X-Y plot with *time* on the X-axis and *destination ports* on the Y-axis. By querying the Netflow database for all RST-records that originated from the outside of the perimeter, one should be able to immediately see if there are anomalies in network usage. By querying the Netflow database for all CON-records, one should be able to immediately see if unauthorized protocols or services are used.

### 5.2.2 Limitations

Even though the Netflow system performs very well in the experiments in this thesis, there are some limitations regarding its usage as a slow portscanning detector.

Like all other applications that record network traffic, the speed of the network can cause problems. If the network interface card or software applications that process incoming packets cannot keep up with the speed, they will begin to drop packets. Care must be taken in order to mitigate this when one builds a Netflow system. Fast processors, enough memory and fast disks should reduce these problems. Tuning the kernel for maximum performance and shutting down all unnecessary services should also help in this matter. In [25], Axelsson demonstrates the use of Argus in Gigabit speed networks, and provides implementation suggestions.

To make the argus-files manageable, they are divided into parts. Unfortunately, this may confuse *ra* when it reads the files and push the records into the database. Multiple records for the same session, can cause confusion about which party started the conversation. *ragator* is a tool in the Argus suite that reads argus data from an argus-file, and merges matching argus flow activity records.

One of the Netflow system advantages is that it can keep the state of the network for a very long time. This time-window is only limited by the size of the database. Large database means slower SQL-queries. The usable size of the database is highly dependent on the physical space on the disks and the CPU-speed and memory capacity. It is likely that the usability (responsiveness) of the system will decline at some point as the database grows.
To mitigate this, one can do batch-calculations instead of dynamic calculations (graphs and tables are generated dynamically in this thesis).
Another mitigation is to decide the amount of data that is needed in the database at all times. If one month of data is enough, older Netflows can be purged automatically. The information is still stored as argus-files, so they can be inserted later if needed.
A third way of mitigating this problem is to aggregate similar records. One should use caution when doing this, as information will get lost in the process.

## 5.3 The use of Snort in slow portscanning detection

Snort [41] is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more.

Figure 2 shows the internal structure of the Snort network intrusion detection system.

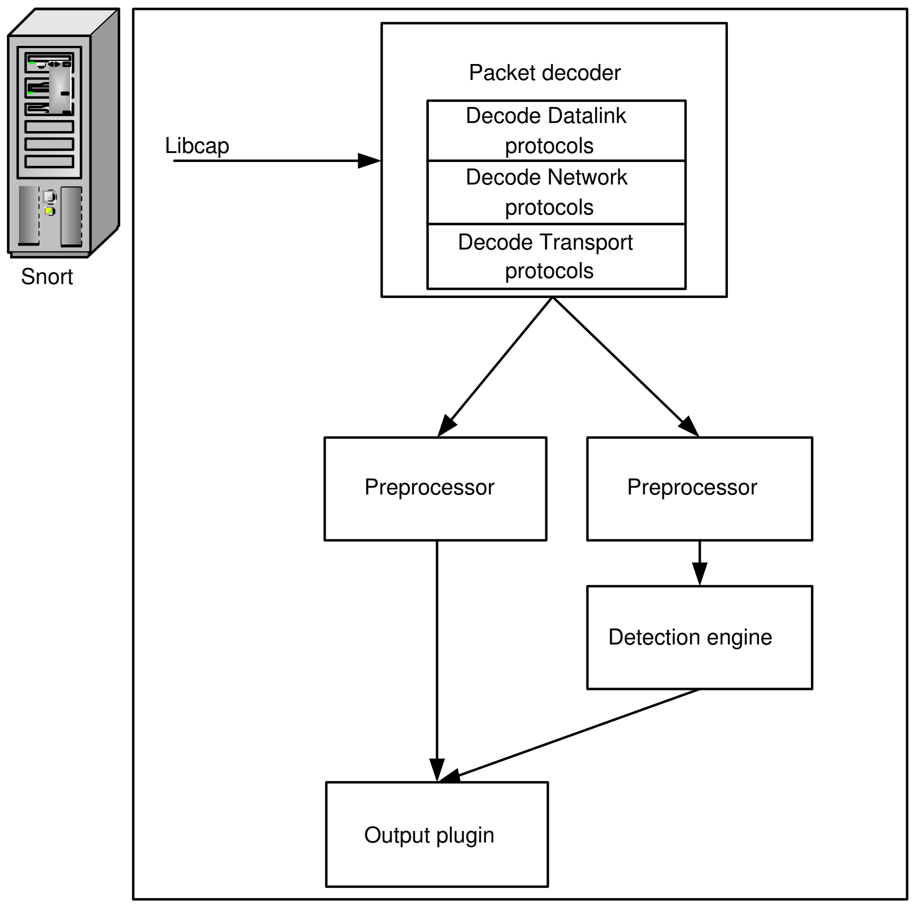

Figure 2: Snort's internal components

Snort does not pull network packets off the channel itself, but counts on libcap[3] to perform this. Snort then decodes the packets and feeds them to the preprocessors. There are basically two types of preprocessors. One category examines the packets for suspicious activity and directly notifies the output module. The other type modifies the packets in order to enable the decision engine to properly inspect the traffic. Snort cycles all packets through all preprocessors in the case where an attack needs more than one preprocessor to be detected.
Preprocessors and rules are configured in *snort.conf*.

[3]Libcap is a library for capturing network packets from the network interface card

### 5.3.1 Portscan detection in Snort

This section looks closer at the preprocessors and rules that are relevant to detecting portscans. Snort has a history of different preprocessors for detecting reconnaissance activity. This thesis focuses on the preprocessor that appeared in Snort version 2.3.0.

#### sfportscan

This rather new preprocessor appeared in snort version 2.3.0. Instead of detecting the number of SYNs, FINs or other creative combinations of TCP-flags hitting our target, it monitors the RSTs going *from* it over a period of time. When a target is portscanned, it will according to RFC 793 [33] send back a SYN+ACK if there is a listening service on that port and a RST if there are no listening services. This could result in fewer false positives, as these are often experienced in busy networks with lots of legitimate SYNs in a short period of time.
Sfportscan requires the "flow" preprocessor and its usage is advised in order to disable evasion alerts within the "stream4" preprocessor as some scan packets can cause these alerts to be generated.

The configurable parameters for sfportscan are:
proto { proto } [tcp,udp,icmp,ip_proto,all]
scan_type { scan_type } [portscan,portsweep,decoy_portscan,
distributed_portscan,all]
sense_level { level } [low,medium,high]

**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 never triggers 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 thus generate Filtered Scan alerts. This setting may generate false positives on active hosts (NATs, proxies, DNS caches, etc), so the user may need to deploy the use of Ignore directives in order to properly configure 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 detect some slow scans because of the continuous monitoring, but it is very sensitive to active hosts. This will most definitely require the user to configure sfPortscan.

See Appendix F for more details on configuring sfportscan.

#### SPADE

The Statistical Packet Anomaly Detection Engine (SPADE) is a Snort preprocessor that uses statistical methods to create a baseline of what types and flow rates of traffic exist on the network. From that baseline it can then alert the operator as to what is different. This is primarily useful for detecting outbreaks, bandwidth abuse (intentional or not), and unusual protocols.

SPADE may seem like a perfect candidate for the experiment. However, it turned out that SPADE has not been maintained for quite some time, and that it is still far too immature.

**Rules**

There are several rules for detecting scanning activities, but no rules designed to detect slow portscans.

### 5.3.2 Limitations

*Sfportscan* is a preprocessor with limited configurable options. Essentially, if one wants to increase the time-window, one must increase the sensitivity level.

Sensitivity level *Low* enumerates RST responses from the target. The threshold is 5 responses in 60 seconds. It is very unlikely that *sfportscan* will be able to detect the slow portscan with this setting.

Sensitivity level *Medium* enumerates RST responses from the target as well as connection counts. The threshold is 5 responses in 90 seconds. It is also very unlikely that *sfportscan* will be able to detect the slow portscan with this setting.

Sensitivity level *High* enumerates RST responses from the target and connection counts. It also uses portscan statistics for hosts on the network. It can detect some slow portscans, but it is also sensitive for active hosts on the network. The threshold is 5 responses in 600 seconds. This will be sufficient for detecting the slow portscan in the experiment.

The threshold and time-window can be changed, but one would have to alter the source code and recompile Snort. According to Andrew Mullican [45] this action is not recommended as it could cause *sfportscan* to be more prone to false positives. Mullican claims that configuring the time-window should not have negative effects regarding memory and CPU usage.

As *sfportscan* operates by means of time-windows, it will produce several individual alerts for the same event, if the event spans several time-windows. Based on these alerts, it is impossible for the security analyst to know if it is in fact one event that spans several time-windows, or it is several events that take place in every time-window. Thus, these redundant alerts will be treated as false alarms in this thesis.

## 5.4 The use of Network Flight Recorder in slow portscan detection

Network Flight Recorder [42] (NFR) is a realtime network intrusion detection tool. It emerged in 1997 and is now developed by Network Flight Recorder Inc.

Network Flight Recorder provides traffic analysis and monitoring tools that help the operator see how the network is being used. Its monitoring package defines a flexible, user-programmable system that enables:

- Recovering or monitoring online transaction records.
- Keeping historical statistics about how the network grows.
- Generating detailed breakdowns of how the network services are being used and by whom.
- Watching for patterns of abuse of network resources and identifying the culprit in real-time.
- Setting burglar alarms that alert the operator to security violations or unexpected changes in the network.
- Logging and monitoring who went where on the network.
- Replaying attackers' sessions and learning what they did.

In NFR, the intrusion detection signatures are grouped in so-called backends, which in turn are grouped in packages. The backends are made in the N-Code language and are configurable through variables.

Network Flight Recorder is scalable and can monitor small as well as enterprise networks. Figure 3 shows a typical NFR setup.

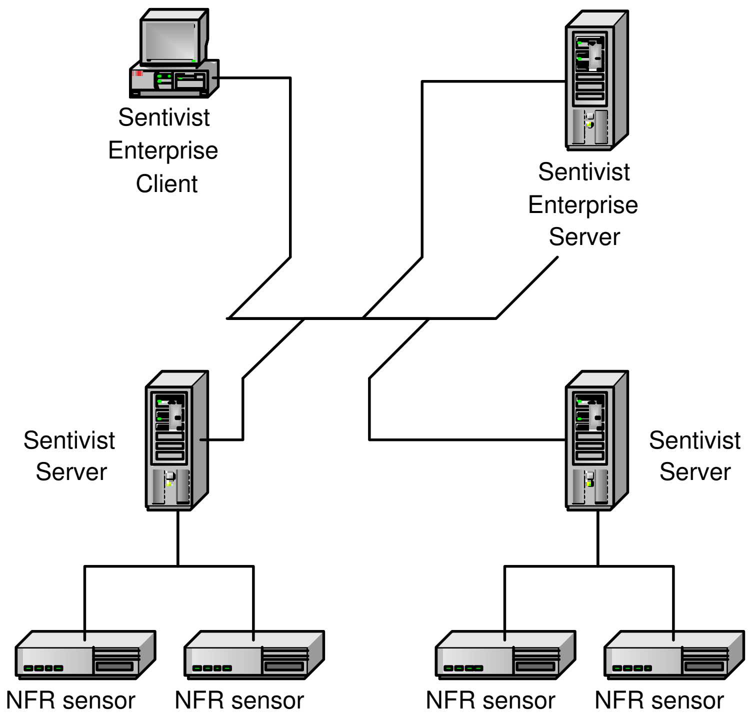

Figure 3: Typical NFR setup

### 5.4.1 Portscan detection in Network Flight Recorder

NFR has several backends for detecting reconnaissance activity. One of them is the TCP Portscan backend that will detect most types of TCP portscanning, e.g. TCP SYN-scans. As other backends, TCP Portscan is configurable through variables (See Table 4 for details).

TCP Portscan divides the destination ports into three classes. Privileged ports (ports below 1024) belong to one class, and are given the weight 3. Unprivileged ports (ports above 1024) belong to another class, and are given the weight 1. Trigger ports belong to a special class which includes ports that one normally does not see any traffic towards. This list[4] is configurable, and they are given the weight 6.
The backend also has an INTERVAL, which is the time-span NFR enumerates the weighted destination ports a source IP has tried to access. If the sum for one source IP address exceeds the value in the THRESHOLD variable, the TCP Portscan backend triggers a portscan alert.

[4]Default trigger ports are : 1,2,3,5,6,8,10,41,124,200,316,414,520,634,744,801,997 and 1000

TCP Portscan variables are presented in Table 4.

| TCP Portscan Variable | Value | Description |
|---|---|---|
| INTERVAL | 16 | Time interval (in seconds) to enumerate portscans |
| PRIV_WEIGHT | 3 | Weight of privileged ports |
| NONPRIV_WEIGHT | 1 | Weight of non-privileged ports |
| TRIGGER_PORTS | List | Unused TCP ports that will have a greater weight |
| TRIGGER_WEIGHT | 6 | Weight of Trigger ports |
| THRESHOLD | 64 | Threshold of weighted count to determine portscans |

Table 4: TCP Portscan backend variables and their default values

As Table 4 shows, the default INTERVAL value is 16 seconds. This is more than enough for detecting ordinary portscans. In a default *nmap* portscan, NFR enumerates approximately 200 ports in 16 seconds. Of course, a time-interval of 16 seconds is not enough for detecting slow portscans. In the experiment, one packet per minute is sent. If INTERVAL were to be increased in order to enable NFR to detect the scan, the value would have to be set at minimum 3840 seconds (Assume that only unprivileged ports are hit, and multiply 60 seconds with the THRESHOLD level, 64). The problem with this approach is that the NFR sensor will spend more memory and CPU cycles on this backend as the INTERVAL increases.

Another approach is to increase the WEIGHT. To increase the weight of the Trigger ports should not cause problems with CPU cycles and memory, but it is very important that the ports in the Trigger list are really unused ports. In this case, NFR could even trigger an alarm after the first packet hits one of the Trigger ports. However, maintaining such a list can be a time-consuming task and "random" connection attempts to the Trigger ports could set of portscanning alarms. If the list is not complete, one could experience false negatives. If the network usage profile changes, NFR could produce false positives.
Increasing the PRIV_WEIGHT or UNPRIV_WEIGHT is also possible, but can generate false positives in networks where the source IP addresses contact many services.

The third option is to decrease the THRESHOLD parameter. In this case, the TCP portscan backend becomes more sensitive. This is equivalent to increasing the WEIGHT of the three classes, thus there is a chance that normal traffic causes the backend to trigger false positives. In the experiment, the THRESHOLD parameter is varied.

### 5.4.2 Limitations

The TCP Portscan backend is written in N-code, a proprietary programming language for writing NFR packages and backends. One cannot ignore the fact that the code itself may have limitations. However, evaluating this code requires special training.

Table 4 shows the configurable variables for the TCP Portscan backend. The default value for the INTERVAL variable is enough for detecting ordinary portscans, but is a much to small window for slower portscans. To enable NFR to detect such slow scans, the value was increased to 600 seconds. The other values were not changed. By increasing NFRs "state-window" to 600 seconds, the sensor is forced to keep track of everything it sees during those ten minutes. During the lab experiment (1Mb/sec continuous network speed) the sensor reported 35% CPU usage. One should keep in mind that almost all backends were turned off. This gives reason to believe that a TIMEOUT value of 600 seconds will be too much when the sensor loads more backends and monitors a faster network.

As with Snort, NFR operates in time-windows. This causes it to produce several individual alerts for the same event, if the event spans several time-windows. Based on these alerts, it is impossible for the security analyst to know if it is in fact one event that spans several time-windows, or it is several events that take place in every time-window. Thus, these redundant alerts will be treated as false alarms in this thesis.

# 6 EXPERIMENTAL WORK

Section 1.7 presented some scenarios where Netflow analysis either has been used, or may be used for misuse detection. One such misuse is slow portscanning. Slow portscans are a type of reconnaissance that use stealthy techniques to evade the IDS. It may very well be one of he hardest events for an ordinary IDS to detect, because of the long intervals between each individual packet in the scan. The motivation for being interested in slow portscan detection is discussed in Section 1.3.

In this section we conduct an experiment to answer the research question:

*Has Netflow analysis a higher or lower true positive/false positive ratio for detecting slow portscanning than Snort and Network Flight Recorder?*

The hypothesis is that Netflow Analysis has a lower true positive/false positive ratio than the two other systems.

## 6.1 Strategy

When planning an experiment like this, one needs to consider several parameters.

**Reliability:** Ensuring reliability is a key factor for achieving a successful experiment. Reliability means that one should get the same results when reproducing or repeating the experiment. The reliability requirements are met in this thesis by

1. Using the data-set from [9], The 1999 DARPA off-line intrusion detection evaluation. The data-set from [9] was recorded during the first week of the DARPA evaluation, from the outside of the perimeter. As reported by [9] there are no known attacks in the data-set.
2. Providing a full description of the hardware, software and configuration settings used in the experiment
3. Providing a full description of how the scan is executed.

**Validity:** Ensuring validity is the other key factor for achieving a successful experiment. Validity means that the results actually say something about the IDS's ability to detect slow portscans. One of the major problems with IDS testing is the choice of the data-source. It may be difficult to ensure that the traffic that is actually sent towards the IDS's is representative for most environments. There are probably no ideal data-sets that are representative for all networks where IDS deployment is considered. One way to mitigate this is to run the tests with several different data-sets in order to investigate if the results are similar. In this experiment, two data sets are used. One as described above, and one from a production network with 200 users.

**The type of scan:** A target can be scanned in a multitude of ways. ICMP scans, UDP scans, TCP SYN/FIN/NULL/XMAS scans are some examples. One can do a vertical scan, where one enumerates all services on one host, or one can do a horizontal scan, where one enumerates one (or a few) services on a range of hosts. In this thesis one of the most common types of scans are used, the vertical TCP SYN scan.

**The speed of the scan:** To increase the time between each sent packet in a scan is a technique to evade the IDS. There is no clear definition of what a slow portscan is, but at some point (time-interval of 30 sec, one minute, one hour, one month) the IDS will no longer be able to correlate the individual packets and notice that it is in fact a scan. In an experiment like this, one would like this time-interval to be so large that there is no doubt that it is a slow portscan. However, as the interval increases, so does the length of the experiment. To be able to complete the experiment in a reasonable time, the intervals between each packet are set to 60 seconds. This serves two purposes. First, it is a slow portscan. Many IDSs will not be able to detect this with the default configuration. Second, it is not extremely slow. By tuning the IDSs, it is quite possible that they will detect the scan. This can provide useful knowledge on how the systems should be configured.

**Presentation:** How one should present IDS evaluations is a much debated topic. The main tool for presenting such comparisons are modified ROC curves. (See Section 6.3 for details.) Although these have weaknesses [10, 11], they are still the most frequently used method of presentation. The results from this experiment are presented by using these modified ROC curves.

**Change the scan or the IDS:** ROC curves are used for presenting the results and comparing the intrusion detection systems. This requires that one or more parameters must be changed during the execution of the experiment. There are two options: The first is to vary parameters in the scan, for example the interval between each individual packet. The second is to keep the scan fixed and vary parameters in the intrusion detection systems. For this experiment, parameters in the three systems are varied, while the scan is kept fixed for each iteration.

**Network speeds:** As discussed in Section 6.3, the modifications that are performed on the ROC curves render them sensitive to network speeds. There is also a possibility that the large time-windows that are used in the experiment, can cause resource exhaustion if the network speeds are too high. The lab experiment makes use of the data-set from [9]. Due to practical reasons, it was not possible to record the data-set for the field experiment until *after* the execution of the lab experiment. The production network uses a 2Mb/sec link to the internet. The mean network speed is assumed to be in the range up to 1Mb/sec during working hours.

The lab experiment data set was replayed at 1Mb/sec constant speed, and the field experiment data-set at its original speed. Mean network speed for the field experiment data-set has been measured at 0.782Mb/Sec. This is somewhat lower than the network speed of the lab experiment, but it is assumed to be close enough to validate the measurements in the lab experiment. Since the modified ROC curves are sensitive to network speeds, a lower false positive rate in the field experiment should be expected.

## 6.2 Overview

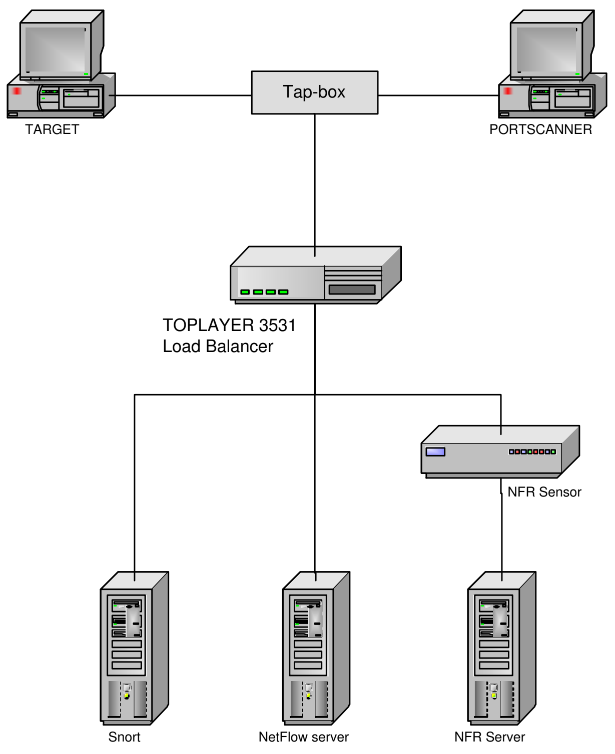

Figure 4: Experiment setup

### 6.2.1 The scan

The scanner host is an Intel-based personal computer running FreeBSD 4.9. *tcpreplay* generates the background traffic, and *nmap* generates the slow portscan.
The target host is an Intel-based personal computer running Microsoft Windows XP (SP1).

There are numerous ways to carry out network reconnaissance. A horizontal scan probes the same port (or a few ports) over multiple IP-addresses, in order to find the same services in several hosts. A vertical scan probes multiple ports in one hosts, in order to find active services in that particular host. The scanner executes a common SYN scan, where one expects to receive an ACK if the service is active and a RST if it is not. To make the portscan slow, there was a 60 seconds interval between each packet sent by *nmap*.
A full scan would take minimum 45 days to complete. The experiment calls for several iterations where parameters in the intrusion detection systems are varied. This would not be possible to do in the time-span available for the experiment. Thus, the number of ports has been limited to the first 1000 ports.

The following commands define the scanner settings for the lab experiment and for the field experiment, respectively.

1. Lab experiment:
   ```
   # tcpreplay -i xl0 -l 27 -r 1 tcpdump.outside
   # nmap -i xl0 –scandelay 60000 -p 1-1000 -P0 172.26.240.23
   ```
2. Field experiment:
   ```
   # tcpreplay -i xl0 -l 13 feltex.cap
   # nmap -i xl0 –scandelay 60000 -p 1-1000 -P0 172.26.240.23
   ```

*Lab experiment*

The slow portscan lasts for approximately 18 hours. The data-set from [9] is replayed with *tcpreplay* to provide background traffic for the entire duration of the portscan. It has been replayed 27 times at 1Mb/sec to last 18 hours.

*Field experiment*

Approximately one and a half hour of network traffic was recorded from a production network. This data-set was replayed with *tcpreplay* to provide background traffic for the entire duration of the portscan, and have been replayed 13 times at its original speed to last 18 hours. Mean network traffic speed in the data-set was measured to be 0.782Mb/sec (Measured with *tcpstat*[1]). This must be taken into consideration when comparing results from the lab and field experiment to each other. The ROC curves as used in this thesis are sensitive to the network speeds, so one should expect fewer false positives in the field experiment.

### 6.2.2 The Netflow system configuration

The system runs on an Intel Pentium-4 1.8GHz/512MB RAM/FreeBSD 4.9 platform. Argus captures the Netflows, and a perl-script inserts them into a PostgreSQL database. Results are queried directly in PostgreSQL or presented as graphs using php/jpgraph. (See Section 5.2 for details about the Netflow analysis system.)

The database is queried for the number of distinct destination ports each source IP address has tried to contact. A threshold determines if the sessions from the source IP address constitutes a portscan or not. To generate the ROC curve, this threshold is varied. Different techniques for strengthening the indicators from the query are also used.

[1]http://www.frenchfries.net/paul/tcpstat/

### 6.2.3 The Snort IDS configuration

Snort 2.3.0 runs on an Intel Pentium-4 1.8GHz/512MB RAM/FreeBSD 4.9 platform. The sfportscan preprocessor writes alerts to /var/log/snort/alert. After each iteration, this file is examined for true and false positives. (See Section 5.3 for details on Snort/sfportscan.)

### 6.2.4 The Network Flight Recorder IDS configuration

The sensor runs on a Sun V60X, and the server runs on a Sun Fire 210 with Solaris 9. Only the backends relevant to TCP Portscan detection are loaded.
The TCP Portscan backend has several configurable options. (See Table 4 for details.) Default values for all variables have been used, except for TIMEOUT that has been set to 600 seconds, and THRESHOLD that has been varied throughout the experiment.

## 6.3 ROC curves as a means of results presentation

Whenever intrusion detection systems are measured, one is interested in the detection rates. The decision made by the IDS can lead to four possibilities:

| Decision | Description |
|---|---|
| True Positive (TP) | There was an event, and the IDS decided that there was an event |
| False Positive (FP) | There was no event, but the IDS decided that there was an event |
| True Negative (TN) | There was no event, and the IDS decided that there was no event |
| False Negative (FN) | There was an event, but the IDS decided that there was no event |

Table 5: Possible IDS decisions

The total number of intrusions is given by

$$TP + FN$$

while the total numer of no-intrusions is given by

$$FP + TN$$

In order to generate the ROC curves[2], one must calculate the True Positive Ratio (TPR) and False Positive Ratio (FPR). The True Positive Ratio is given by

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

while the False Positive Ratio is given by

$$FPR = \frac{FP}{FP+TN}$$

An ideal IDS would have TPR=1 and FPR=0.

[2]ROC curves were developed in the 1950's as a by-product of research related to detection of radio signals in noise. They have also been extensively used in medical research

A Receiver Operating Characteristics Curve (ROC Curve) summarizes the relationship between two of the most important IDS characteristics: the probability of false alarms and the probability of detection measurements.

Figure 5 illustrates the principal of ROC curves.

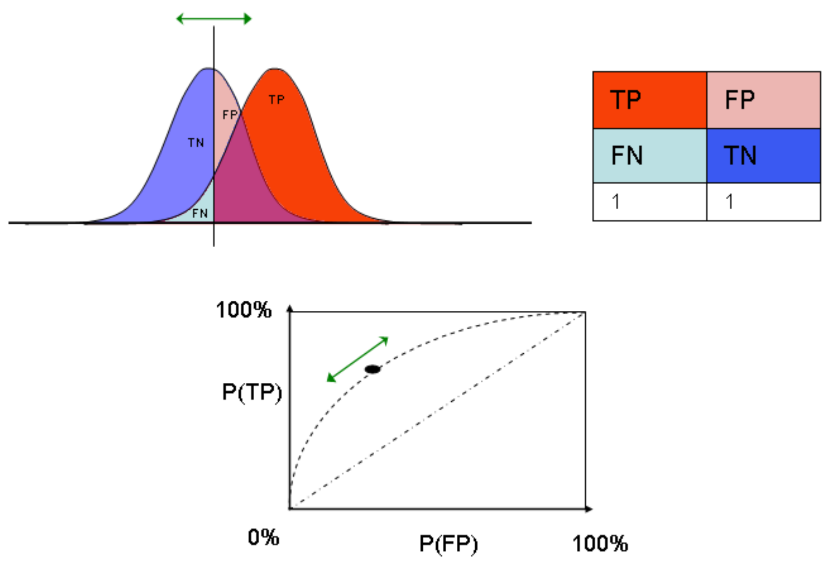

Figure 5: The principal of ROC curves

Starting from two distributions of positives (red) and negatives (blue) one can apply a threshold criterion (vertical line) to arbitrarily separate the two. For overlapping distributions, there is always a tradeoff between sensitivity (TP) and specificity (1-FP). TP and FN as well as TN and FP both add up to 1. Sliding the threshold line towards the distribution of positives will result in a decreased probability for true positive detection P(TP) and FPs, which is equivalent to moving the ROC curve (dashed) downwards. If the two distributions overlap completely, the ROC curve will be the diagonal shown as the dot-dashed curve. The IDS can be configured for operation anywhere along the curve.

In [7], Mell et al. discuss why one must modify the ROC curve so that it can be used for testing IDS. Basically, the modification is that instead of calculating the false positive factor on the X-axis, one uses false positive alarms per unit time. Unfortunately, this causes the ROC curve to be sensitive to the speed of the network. Higher speeds are likely to produce a higher false positive count per time unit. In [10, 11], McHugh comments this modification and points out its weaknesses, but one cannot ignore the fact that this modified version of the ROC curve is after all the best method of testing and comparing intrusion detection systems that is available at the present time.

One should keep this in mind when ROC curves are used for comparing IDSs and look for weaknesses in the experiment that may distort the presentation of the results. The experiment in this thesis is no exception. The network speed is indeed lower in the field experiment than in the lab experiment. One should therefore expect to detect less false positives in the field experiment for the IDS in question.

Figure 6 shows an example of a ROC curve that is used for IDS testing. The X-axis is modified to show the number of false positives per unit time. In this example, IDS 1 has a better mode of operation than IDS 2.

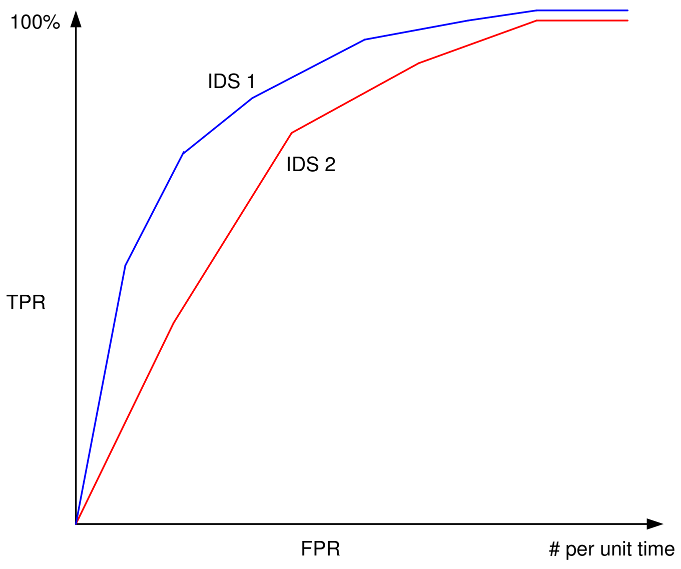

Figure 6: The modified ROC curves used for IDS testing

## 6.4 The schedule of the experiment

### Round 1 - Snort with sfportscan

The sfportscan has the following configurable options:
proto { proto } [tcp,udp,icmp,ip_proto,all]
scan_type { scan_type } [portscan,portsweep,decoy_portscan,
distributed_portscan,all]
sense_level { level } [low,medium,high]

The sense_level was varied from low to high. It would be desirable to have more points in the ROC curve, but this is the level of granularity in sfportscan. Results were recorded from the /var/log/snort/alert file.

### Round 2 - Network Flight Recorder

The TCP Portscan backend has the following configurable options:
INTERVAL, PRIV_WEIGHT, NONPRIV_WEIGHT, TRIGGER_PORTS, TRIGGER_WEIGHT and THRESHOLD. (See Table 4 for default values.) INTERVAL is increased to 600 seconds, and THRESHOLD is varied from 10 to 40.

### Round 3 - Netflow analysis system

Query the database for the number of distinct destination ports each source IP address has tried to contact. A threshold determines if the sessions from the source IP address are a portscan or not. The threshold vary from 2 to 2000.

### Round 4 - Netflow analysis system with additional calculations

The query from Round 3 was altered in a sense that only RST-records were included. This is an adaption of the technique that is used in the *sfportscan* preprocessor.
Additional calculations and techniques to strengthen the indicator from the query are applied. These are calculations of standard deviation for the time-intervals between sessions from the source IP address and the use of visualization techniques for slow portscan detection.

## 6.5 Experimental results

This section presents the results from each individual round in the experiment. There are two ROC curves for each round, one for the lab experiment and one for the field experiment. The scale is adjusted to facilitate easy comparison between lab and field experiment. There is a short comment to the results of each round. Details from the lab experiment are presented in Appendix B and from the field experiment in Appendix C.

The results are discussed more thoroughly in Chapter 7.

It would be preferable to plot the ROC curves in the same graph to illustrate how the results compare to each other. Snort/sfportscan produces more false positives than Network Flight Recorder and the Netflow analysis system. This causes the X-axis scale to be so highly granular that it is not possible to visually distinguish the Netflow analysis system and Network Flight Recorder in the graph.

**Round 1 - Snort with sfportscan**

The sfportscan preprocessor has three sensitivity levels. When set to *low* or *medium* sfportscan did not detect the portscan. As discussed in 5.3.2, this was to be expected. When set to *high* it detected the portscan in both the lab and the field experiment. As shown in Tables 7 and 11, there were still a pretty high number of false positives. Sfportscan also produced several identical alerts for one scan (93 alerts in the lab experiment and 91 alerts in the field experiment). sfportscan *can* be used for detecting slow portscans, but in this experiment the sensitivity level had to be *high*. False positives probably can be lowered by tuning the preprocessor further.

As shown in Figures 7 and 8, it is not possible to draw a proper ROC curve. Normally, one would want more points in the curve, but this is not possible when only altering the sensitivity level in sfportscan. If the x-axis were flipped in Figure 7 it might resemble a proper ROC curve.

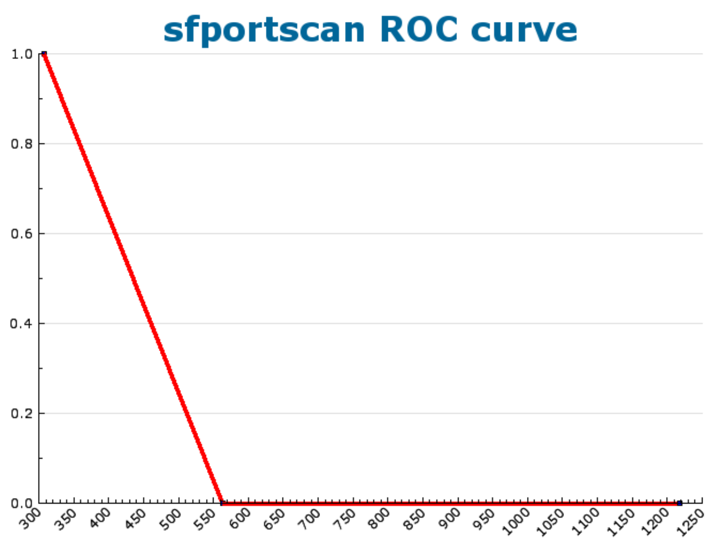

Figure 7: Results sfportscan - Lab experiment

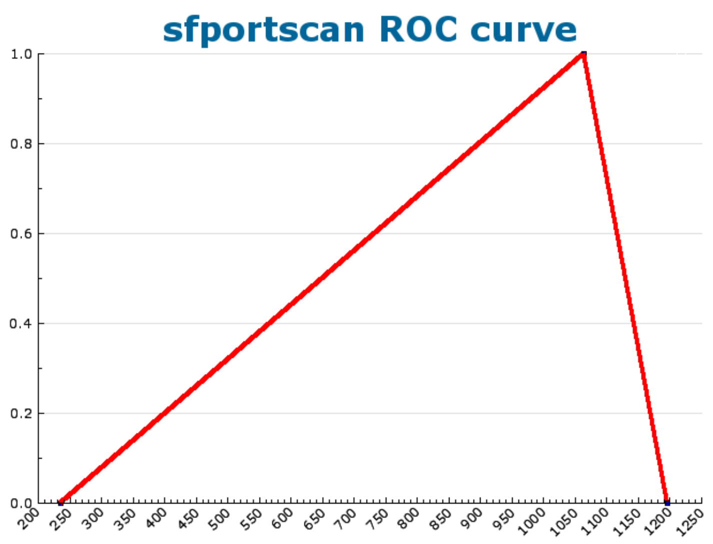

Figure 8: Results sfportscan - Field experiment

### Round 2 - Network Flight Recorder

In this round, the TIMEOUT value for the TCP Portscan backend is raised to 600 seconds. This is a *very* large value, probably too large in a real world scenario. It means that NFR must keep record of all traffic for a period of 600 seconds. During the lab-experiment the sensor's CPU utilization was approximately 35%. One should keep in mind that all unnecessary backends were turned off, and the traffic speed was only 1Mb/sec. During the field-experiment the CPU utilization was a bit lower, and so was the network traffic speed. Nevertheless, NFR performed well. As shown in Tables 8 and 12, it produced only 40 false positives in the lab-experiment when the THRESHOLD was set to 25. In the field-experiment it produced only one false positive when the threshold was set to 40.

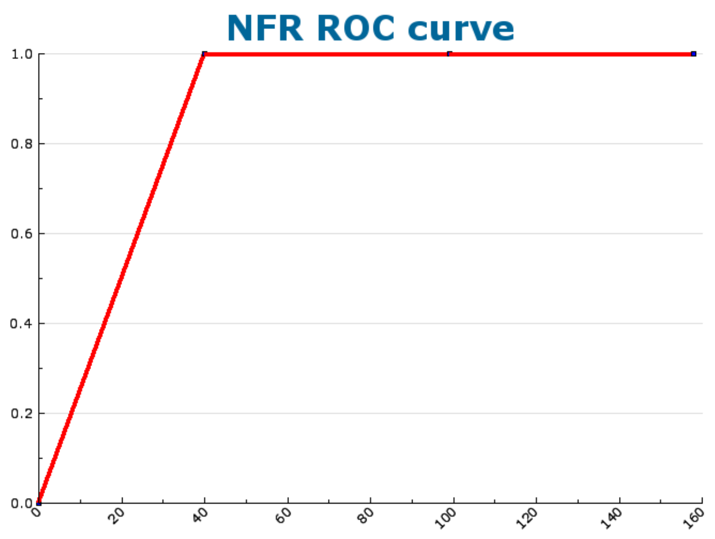

Figure 9: Results Network Flight Recorder - Lab experiment

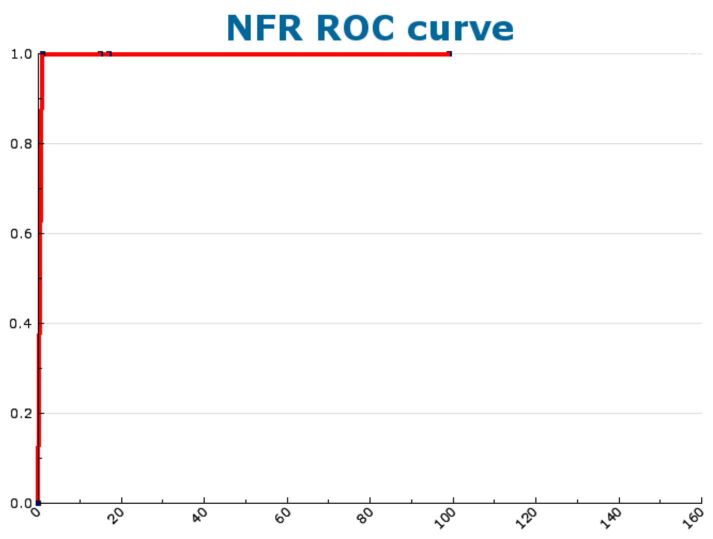

Figure 10: Results Network Flight Recorder - Field experiment

### Round 3 - Netflow analysis system

The Netflow system performed best in both the lab-experiment and the field-experiment. As shown in Tables 9 and 13 the Netflow system detected the portscan as long as the threshold was set to 1000 or below. This should come at no surprise, as the scan was limited to the first 1000 ports. In the lab-experiment, there were no false positives when the threshold was set between 10 and 1000. In the field-experiment there were no false positives when the threshold was set between 7 and 1000.

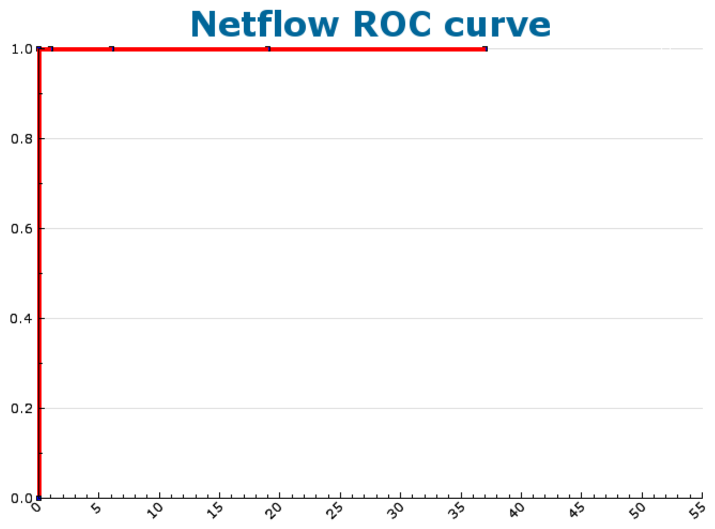

Figure 11: Results Netflow - Lab experiment

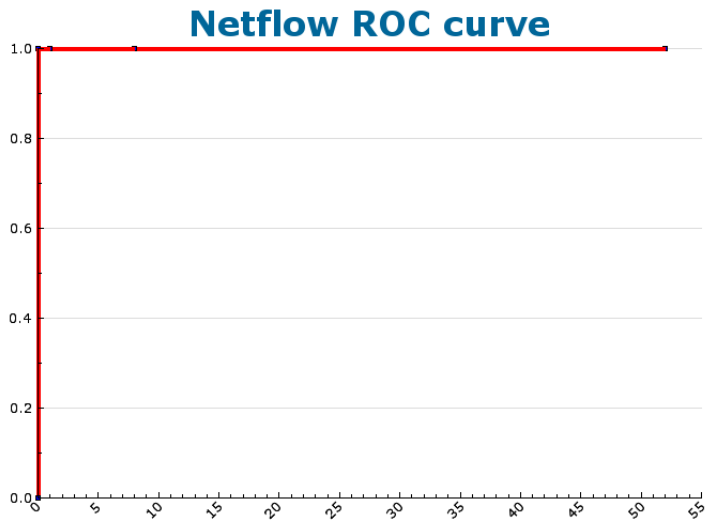

Figure 12: Results Netflow - Field experiment

### Round 4 - Netflow analysis system with additional calculations

Additional techniques to further improve the results from Round 3 were used. The goal was to reduce the false positive rate even further.

**Improving the query:**

First, the query was modified so that it would only look for sessions that got reset:

*SELECT src_ip,count(distinct dst_port) FROM flow WHERE state='RST' GROUP BY src_ip ORDER BY count(distinct dst_port) DESC;*

This was an adaptation of the technique used in Snort's sfportscan, and allowed the threshold to be lowered to 5 for the field-experiment and still get 0 false positives.

**Standard deviation calculations:**

By calculating the standard deviation for the intervals (in seconds) between the sessions, we have investigated if a portscan as performed in this thesis had a much smaller standard deviation than normal network traffic. Table 6 presents a sample of this calculation. The scanner has a much lower standard deviation than the sessions from the other source IP addresses.

| Source IP | Dest. port count | Number of sessions | Std. dev. |
|---|---|---|---|
| 172.26.240.100 | 998 | 998 | 2.68 |
| 158.112.96.108 | 7 | 5214 | 38.91 |
| 144.84.15.123 | 6 | 7473 | 13.19 |
| 158.112.96.181 | 6 | 2624 | 50.74 |
| 158.112.96.165 | 6 | 4011 | 18.72 |
| 144.84.15.117 | 5 | 1903 | 43.02 |

Table 6: Netflow - Standard deviation sample calculation

When plotting the standard deviation score for all source IP addresses, one can easily see if some of them have small values.

Figure 13 presents the standard deviation score for all source IP addresses in the lab experiment. The first bar represents the scanner in the experiment, and it clearly has the lowest standard deviation score in the plot. This indicates that the sessions coming from this source IP address are very regular, which is a property one would expect to see in scanning activities. Some of the other source IP addresses also seem to have a low standard deviation score. One should keep in mind that the background traffic is looped, and that this adds a regularity to the traffic that is probably not experienced in a real world scenario.

Figure 14 presents the standard deviation score for all source IP addresses in the field experiment. The first bar, which has the lowest standard deviation score, corresponds to the scanner in the experiment.

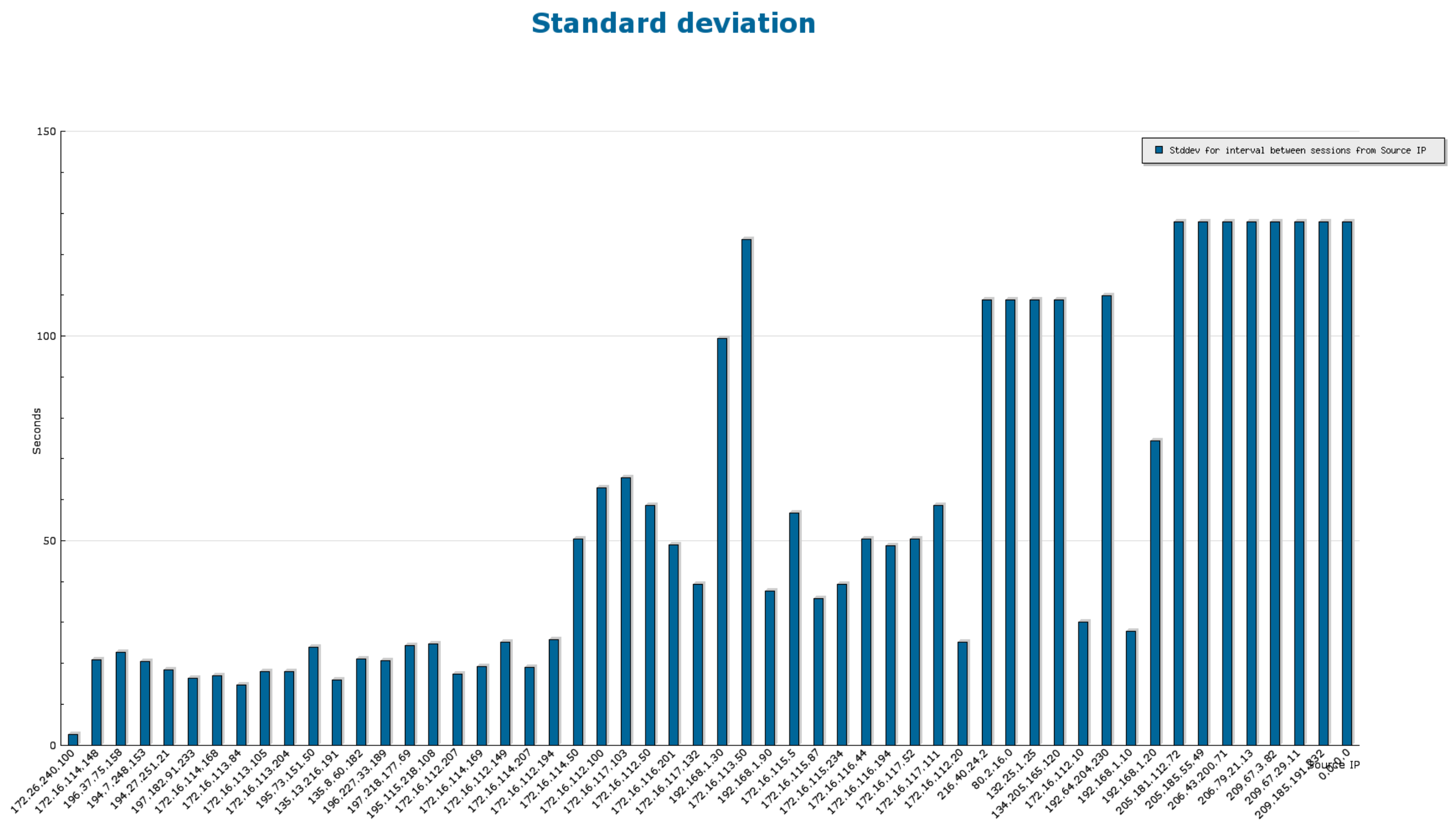

Figure 13: Standard deviation score in lab experiment

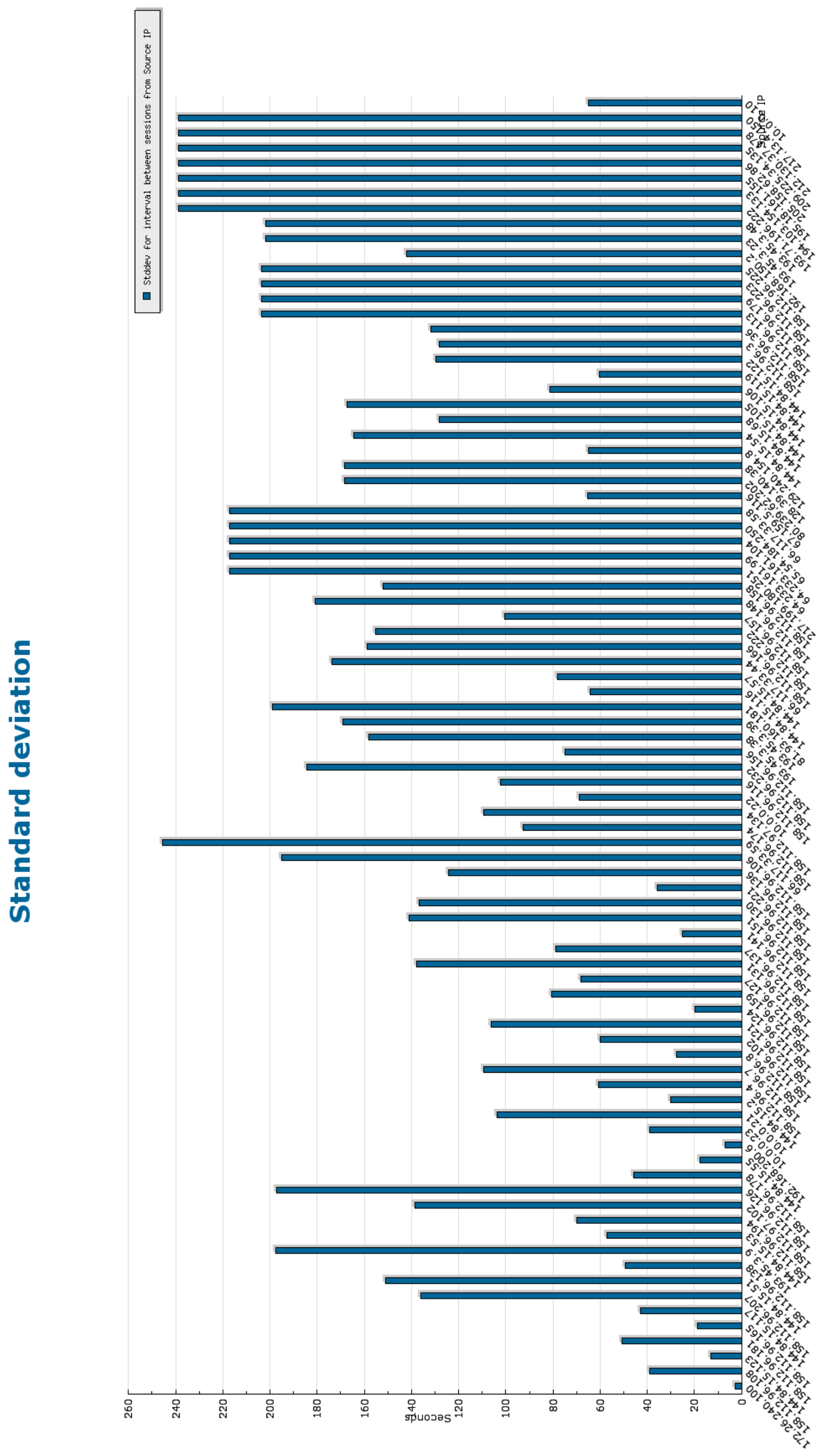

Figure 14: Standard deviation score in field experiment

**Visualization:**

Visualization was used as a tool for aiding the human operator to efficiently, clearly and continuously assess the security status of the entire network. It is well known that humans are especially good at recognizing visual patterns. By visualizing patterns of misuses, we believe that a human at the right place in the decision making process could excel an automated decision maker (IDS detection engine).

In [5, 6, 13] Yurcik et al introduce a prototype tool for visualizing Cisco NetFlows. The concept is to enhance the intrusion detection process by visualizing the state of the computer network.

Using a pre-determined threshold for deciding whether it is in fact a slow portscan or not, is not easy. If some of the hosts have many active services, this can cause false positives. If the decision is left to the human operator, one hopes that his knowledge of the normal state of the network and intuition will further reduce the number of false positives. One way of doing this is to plot time on the x-axis and destination ports on the y-axis. By querying the Netflow database for all connection attempts that result in a RST state, the operator should easily detect both 'quick' and 'slow' portscans.

Figure 15 illustrates how all the RST-records in a portscan can be visualized. A faster scan will be visualized as a column or spike, while slower scans are more 'cloudy'. Of course, this graph does not reveal who the scanner is, who he is scanning or the number of concurrent scans. But it may serve as a first warning and trigger the operator to investigate further.
Figure 16 illustrates the same traffic, but now the portscan has been removed. This resembles a normal graph, where one expect to see some RST records in relation to legitimate services (and probably a lot of records in relation to failed connections to known backdoor programs).

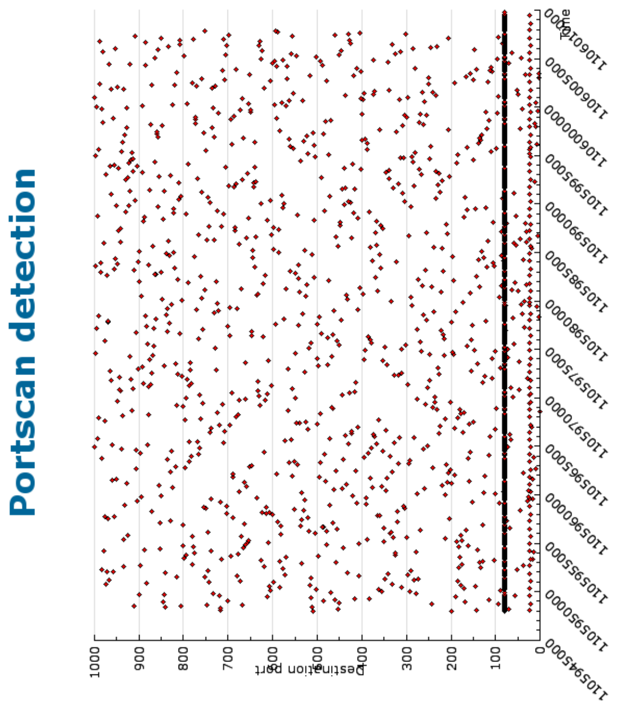

Figure 15: Visualizing a slow portscan

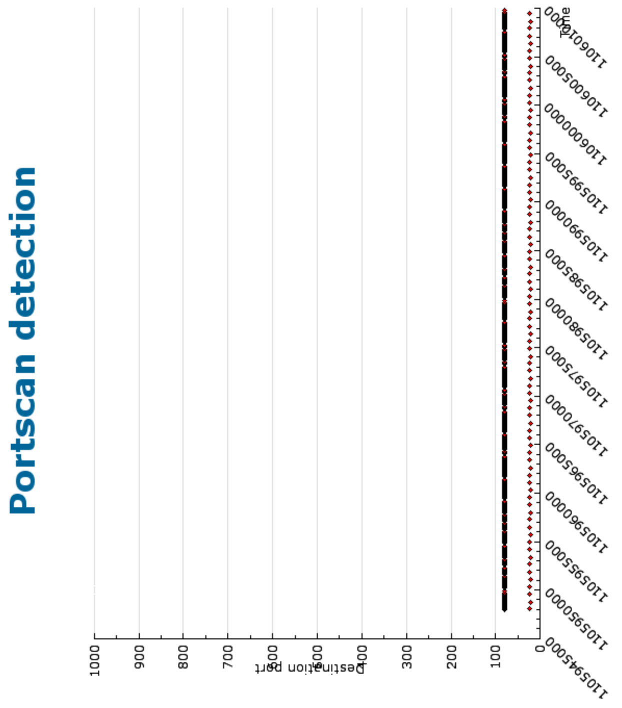

Figure 16: The portscan has been removed

# 7 DISCUSSION OF EXPERIMENTAL RESULTS

This experiment has shown that Netflow analysis can be used for detecting this type of slow portscans, and that Netflow analysis does this better than Snort and Network Flight Recorder. It is possible to further reduce the number of false positives through data-mining and visualization techniques. The problem with this approach lies in the fact that Snort and NFR are very complex systems that have many configuration options. This experiment has not explored all these possibilities. One can therefore not discard the possibility that there may exist a configuration that would make the system perform better than it did during the experiment in this thesis.
It should also be clear that Snort and NFR have the capability of detecting misuses that is beyond the Netflow analysis system's capabilities. Cases where one would need to examine the contents of a session would obviously not be detected by the Netflow system.

**The Netflow analysis system.**

The Netflow record contains only a fraction of the total amount of data that is transmitted during a session. Because of this, it is possible to store session-data for a very long time. This increased time-window is the most important benefit of using this system for slow portscan detection.

In this experiment, the database was queried for all individual destination ports that each source IP has tried to contact (successful and unsuccessful connection attempts). A regular vertical portscan will probe thousands of ports on the target, while legitimate use will be limited to a few ports. The query listed all source IP addresses, and as Tables 9 and 13 show, one source IP address has clearly tried to connect to a very high number of destination ports. By applying a threshold, the activity from a source IP that exceeded the threshold was defined to be portscanning activity. It may be difficult to decide on a fixed threshold for all networks. In this experiment the threshold should be 20 for the lab experiment and 10 for the field experiment. In a real world scenario, the security analyst should know how many legitimate services he is defending, and set the threshold accordingly.

There is always the possibility of false positives. If there are many legitimate services on the defended network, legitimate traffic may raise over the threshold. Some protocols, such as FTP, may use several ports in a logical session. There is also the possibility of false negatives. If the scanner is targeting only a small set of destination ports, this number may be below the threshold. To mitigate this, one can turn to data-mining techniques. One such technique is to calculate the standard deviation for the time-interval between the sessions. If one assume that the scan sends out its packets with a regular interval, the standard deviation should be small. Normal network usage tends to have higher standard deviation between sessions from a given source IP address. When combining these calculations with the original query, one may strengthen the indicators from the query. If the attacker introduces irregular time-intervals between the individual packets in the scan, or introduces other network traffic besides the scan, he might be able to evade this technique. Therefore it is not believed that standard deviation calculations can be used to weaken an indicator from the query.

It is important to realize that the alerts from our intrusion detection systems are indicators and warnings. Sometimes one must correlate several indicators before one can turn them into an event. Another indicator that can be used in slow portscan detection is visualization. In the experiment, an X-Y plot with time on the X-axis and destination ports on the Y-axis was made. The graph was populated with a query for all RST-records (thereby excluding successful connection attempts). In the case of no portscan, one would see RST-records to some of our legitimate services, and probably to the most common backdoor program destination ports. In the case of a portscan on the other hand, one would see RST-records all over the destination port range. Depending on the scale on the X-axis, this activity might be represented as a thin spike for a normal portscan (takes little time to complete) and represented as a cloud for a slow portscan. (See Figure 15 for an example.)

**The Snort IDS**

The sfportscan preprocessor is the latest addition to Snort's portscan detection capabilities, and were introduced in version 2.3.0. It is meant to replace older portscan preprocessors, and was for that reason chosen in the experiment. Older preprocessors like portscan2 and SPADE was considered, but not included in this experiment.

The sfportscan preprocessor utilizes a new idea for portscan detection. A portscan will generate a high number of RST packets coming from the target. By enumerating these, instead of connection attempts, one hopes that sfportscan will reduce the number of false positives.

Sfportscan has three sensitivity levels. When this level is increased, sfportscan widens its time-window, and starts enumerating more than just RST packets. For *Low* the timeout is 60 seconds, for *Medium* it is 90 seconds and for *High* it is 600 seconds. These values are hard-coded into the source. If one wants to change these, Snort must be re-compiled. These values were not changed in the experiment. The threshold for each sensitivity level is 5 enumerated ports inside the time-interval.
This means that sensitivity levels *Low* and *Medium* will not be able to detect the scan in this experiment. Sensitivity level *High* should be able to enumerate 10 ports in 600 seconds, and given a threshold of 5 ports, it should be able to detect the scan in this experiment. The results presented in Tables 7 and 11 confirm this. There are some results that may be a bit confusing. In the lab experiment, sensitivity level *High* produced fewer false positives than *Low* and *Medium*. In the field experiment, sensitivity level *High* produced fewer false positives than *Medium*. This is not expected when examining how sfportscan works. Sfportscan is a rather new preprocessor. One should therefore investigate if this discrepancy can be explained as implementation problems.

The high number of false positives is also a concern. The network speeds were 1Mb/sec for the lab experiment and 0.782Mb/sec (mean network speed) for the field experiment.

These are pretty low speeds, and yet did sfportscan produce 308 false positives in the lab experiment and 1064 false positives in the field experiment. One would naturally be able to lower these by filtering noisy IP addresses, but they still seem like unacceptably high numbers.

Snort is a very popular Network IDS and is capable of detecting a wide range of misuses. As a low-cost intrusion detection system it may have a natural place in the detection strategy, but this experiment suggests that it should not be used for slow portscan detection.

**The Network Flight Recorder IDS**

The Network Flight Recorder is a commercial network IDS solution. It has a wide range of detection capabilities, and can be extended with its own Security Information Manager (SIM) and the Enterprise Console for alarm management.

Network Flight Recorder performed acceptably during the experiments in this thesis, but one should keep in mind that NFR was configured for portscan detection only. It is likely that other configuration schemes would have influenced the results. It is beyond the scope of this thesis to further investigate this.

In this experiment, the packages/backends that are absolutely necessary for portscan detection are activated. The TCP Portscan backend is configurable through variables. (See Table 4 for details.) NFR assigns weights to different classes of destination ports. Ports above 1023 belong to one class, 1023 and below belong to another class, and there is a special class to which trigger ports belong. When a port in the three classes is hit, NFR calculates the weight. If the sum reaches the threshold inside the time-window, a portscan alert is generated. The default threshold is 64, and the default time-window is 16 seconds.

The idea of assigning weights to classes of ports seems to be a good one. Normally, services will reside below port 1024, so these may be of special interest to a scanner. Giving these a higher weight than ports 1024 and above makes perfect sense. In addition, the TCP Portscan backend keeps a list of special ports that one wants to give a higher weight. These are ports where one should not see any traffic, for example port 0,1,2,3,5 and so on. A careless scanner may hit these ports, and by giving them a high weight, the threshold would be reached more quickly. One would of course need to maintain this list. Should there be a legitimate service on one of these ports, false positives are to be expected.

A time-window of 16 seconds is too narrow to detect the scan in this experiment. Therefore the time-window was changed to 600 seconds. The threshold varied from 10 to 40. With these settings, NFR was able to detect the slow portscan in this experiment. As shown in Tables 8 and 12, the optimal THRESHOLD setting was 25 for the lab experiment and 40 for the field experiment.

The results for the lab and field experiment are somewhat different. As stated in Section 5.4 the portscan in this thesis should be detected when the THRESHOLD holds the value 30. This is the maximum value where one would still be certain to detect the scan (10 privileged ports in 10 minutes equals a weighted score of 30). When the THRESHOLD variable is lowered, it becomes easier to detect the scan since NFR does not have to enumerate so many ports before generating a portscan alert. As seen in Table 8, NFR did not detect the portscan when THRESHOLD was set to 30. It had to be lowered to 25 before it would detect the scan. As Table 12 shows, NFR detected the scan with THRESHOLD set to 30 in the field experiment. One needs to examine the difference in the lab and field experiment to find the explanation behind these results, and one should be aware of the fact that the experiment is not entirely controlled, so there may be non-controllable factors that may have influenced the results.

The *nmap* settings was not altered for any of the iterations in the experiment. The *tcpreplay* settings (provides the background noise) was altered in two ways. One, different data sets were used in the lab and field experiment. Two, the lab experiment had a fixed network speed of 1Mb/sec while the field experiment used the natural speed of the data set.

If the scanner sent the traffic as it was supposed to, one must assume that NFR is responsible for the unexpected lab experiment results. However, it is not likely that the network speed overloaded the NFR sensor. It reported a CPU usage of 35% during the experiment. This is probably not a high enough load to cause the sensor to start dropping packets.
The other possibility was that the scanner did not send the packets as it was supposed to. The scanner was implemented on a computer with a rather old 10Mb/sec network card. It is possible that the buffers in either hardware or driver software got filled up by the continuous traffic that was sent through it, or that other events on the computer exhausted resources and that this caused packets to be dropped before they were sent. The computer and network card should be able to handle network traffic at 1Mb/s. Furthermore, if this was the case, there should be a number of missing records in the Netflow system as well. Table 15 shows that the Netflow system recorded all 1000 packets in the scan. One should note that the Netflow system did not collect the records in the same iteration where the THRESHOLD was set to 30. This means that there is a possibility that some event on the scanner caused these results. In the absence of any better explanations, one should assume that this is the case.

This gives reason to believe that the reliability of the lab experiment is weaker than expected. Even if there is no evidence that the scanner has caused similar results in other iterations, one cannot entirely exclude the possibility. This must be taken into consideration when reading the results.

This experiment cannot fully answer the question whether NFR should be used for slow portscan detection or not. It could detect it with the settings used in the experiment, but one does not know for certain if these settings can have negative effects on NFR's ability to detect other misuses.

When comparing NFR to the Netflow system, it produced more false positives. However, the amount of false positives is manageable for the security analyst. By filtering away noisy hosts, they can probably be reduced further.

If one already has deployed NFR, one should investigate further how it can be optimally configured for slow portscan detection.

### Closing remarks

Planning a strategy for intrusion detection is a very serious and complicated task. Intrusion detection solutions are often very expensive to purchase. The costs of manning a central for intrusion detection and response can also be high.
Before one decides which systems to buy, there are several questions that need to be answered:

**What are the assets?** Most production chains consist of many components and procedures. The assets in such a chain can be the infrastructure itself, including network components, hosts and services. The assets can also be the information that is produced and the procedures that makes use of that information. Information technology supports many of these assets. One needs to know what to protect in order to achieve the best security for the available resources.

**What are the threats?** Protecting ourselves from every possible threat may not be the most cost-effective thing to do. A carefully chosen intrusion detection regime could aid the security analyst in obtaining a better understanding of the actual threats to his assets. By using for example Andrew Jones' [27] approach for threat assessment, he could learn actual threat agents' intentions, capabilities and methods.

**How will the information be used?** Will the situational picture from the intrusion detection systems be used for coordinating countermeasures and to take legal actions against perpetrators? Or to study the attackers intentions, capabilities and methods?

**What data sources are available?** After deciding what assets to protect from which threats, the security analyst needs to decide the type of data source to search for misuses. He would have to know what kind of logs or other data sources are already available and what are the costs related to collecting and processing them. If no relevant data sources are available, he may have to purchase or develop a system that produces the logs he needs.

**What are the limitations?** There will most likely be limitations that may affect the detection strategy. They can be technical limitations like network speeds or storage capacities. They may also be legal limitations like privacy law etc. that limit the type of information one can record and store.

Other factors to consider are economy, competence/knowledge in the organization, requirements for interoperability with existing systems for intrusion detection, scalability and usability.

In [26], Richard Bejtlich introduces the concept of Network Security Monitoring (NSM). NSM is defined as *the collection, analysis and escalation of indicators and warnings* *to detect and respond to intrusions*.

By indicators and warnings, Bejtlich uses the definition found in a U.S. Army intelligence training document[1]. The document defines an indicator as *observable or discernable actions that confirm or deny enemy capabilities and intentions*.

Indications and warnings (I&W) are then defined as *the strategic monitoring of world military, economic and political events to ensure that they are not the precursor to hostile or other activities which are contrary to U.S. interests*.

Bejtlich then adapts this definition to NSM, defining digital I&W as *the strategic monitoring of network traffic to assist in the detection and validation of intrusions*.
In this context, *indicators* are information such as the alerts generated by security products, activity monitors in network units, application logs in hosts, manual reports from users and so on. Indicators from the Netflow system would be results from queries and data mining techniques, as well as visualization of patterns of misuse. *Warnings* are the results of a security analyst's interpretation of indicators.

How does this relate to the Netflow analysis system, Snort and Network Flight Recorder? If one is going to develop a holistic strategy for security monitoring and response, it is unlikely that one will be able to find a single product that will serve all needs. At some places, one may need a powerful IDS like NFR. At other places, economy will only allow us to use open source products like Snort. Yet at other places, limited bandwidth prevents us from collecting anything but Netflow sessions. The ultimate goal is to record everything and store it indefinitely for future analysis. When reality constraints prevent us from reaching this goal, one must do the next best thing: Decide what is absolutely needed to detect at different locations in the particular infrastructure, and deploy the best tool given the limitations in that environment. Sometimes the tools may be overlapping, sometimes they may be complementary and sometimes they may leave gaps in the detection capabilities.

One cannot use the experiment in this thesis as a rationale for replacing systems like Snort or NFR with a Netflow system. A reasonable strategy for intrusion detection should of course aim at giving the largest return of investment as possible. The information given by the systems should be sufficient to implement coordinated and effective countermeasures.
A Netflow analysis system has a natural place in this detection strategy, and may fill a niche that systems like Snort and NFR do not fill. By storing meta-data only, it becomes possible to store and process information that covers a large time-window. Signature based systems are limited to detecting misuses they already know. When the security analyst becomes aware of a new attack, it will usually not be possible to go back days/weeks/months to investigate if and when the network was hit with this new attack when using such signature based systems. By monitoring selected time-series data, it is possible that a Netflow analysis system could detect new attacks in an early stage.

[1]Document is titled "Indicators in Operations Other Than War" and can be found in the Federation of American Scientists' archive at http://www.fas.org/irp/doddir/army/miobc/shts4lbi.htm

# 8 CONCLUSIONS

This thesis has demonstrated that one can build a Netflow analysis system based on Argus, and that this system can be used for misuse detection. The research questions for this thesis are presented in Section 1.4. We now return to these to determine if the questions have been answered.

1. To what degree can Netflow analysis be used for intrusion detection?

   Some misuse scenarios where Netflow analysis probably can detect misuses in computer networks are presented in Section 1.7. They are grouped into two categories. Misuses with strong indicators in the Netflow record, and misuses with weaker indicators in the Netflow record. One of the misuses, slow portscanning, is a particular difficult type of misuse to detect with ordinary network intrusion detection systems and was selected for further study in this thesis.

2. Analyze how Netflow analysis, Snort and Network Flight Recorder detect slow portscanning. Find limitations in the three methods, and perform a ranking. The hypothesis was: *Netflow analysis will top this ranking.*
   By analyzing the three systems, one was able to estimate limits in the mechanisms that the systems use for slow portscan detection. The systems focus on different techniques. Snort/sfportscan looks for RST-packets coming from the targets. NFR assigns weight to ports that are unlikely to be hit (except from scanners). The Netflow system takes a step back and examines large time-windows of traffic. As a result, a ranking was made, and the Netflow analysis system tops this ranking.

3. Has Netflow analysis a higher or lower true positive/false positive ratio for detecting slow portscanning than Snort and Network Flight Recorder? The hypothesis was: *Netflow Analysis has a lower true positive/false positive ratio than the two other methods.*

   The experiment was conducted successfully. The experiment confirmed the hypothesis and the ranking is presented in Table 2.

In addition to answering the research questions, other techniques were adapted to improve the indicators from the Netflow analysis system:

1. We adopted the techniques used by Snort/sfportscan in such a way that we only look for connections that were reset. By doing this, the true positive/false positive ratio was improved.
2. We used timing analysis to strengthen the indicator. By examining the standard deviation for the time intervals between the sessions coming from each source we were able to identify sources where very regular sessions originated.
3. We used visualization to aid the operator in deciding if there are portscanning activities in the network.

# 9 FUTURE WORK

This thesis confirms that Netflow analysis systems can detect slow portscanning, and some effort has been made to strengthen the indicator from Round 3 in the experiment. It is possible that there are better ways to strengthen the indicator. By examining timing information or patterns in the scanned ports one could further reduce the false positives. This should also enable us to discern slow portscans from "normal" portscans.

One of the misuses from the scenarios presented in Section 1.7 was chosen for further study in this thesis. One needs to gain a deeper understanding of what other types of misuse that can be detected with Netflow analysis, and investigate how well Netflow analysis systems do this compared to other methods.

This thesis has shown that NFR can be used for detecting slow portscans given relatively low network speeds, few backends loaded and a very wide time-window. In the experiment, the only value that was varied was the TIMEOUT variable. It would be desirable to learn NFRs limitations more thoroughly. At which speeds can it still detect slow portscans? When will the large timeout-window cause it to start dropping packets, and when will this affect its capability to detect other intrusions? How will the alteration of other variables in the TCP Portscan backend affect its ability to detect slow portscans?

Netflow records contain information about events that take place in the network or transport layer. One can also find information on how much data the two hosts are sending and how many packets they send. There is a possibility that this information can indicate the type of traffic that takes place in the layers above the transport layer.
Here we present some suggestions of higher layer events that one perhaps could detect in Netflow data:

1. **Covert Channel activity.** Restrictive firewall policies can spur internal users to set up tunnelling of unwanted protocols inside legitimate protocols. By examining timing information, the number of packets and the amount of data transmitted, it might be possible to detect covert channel activities.
2. **New peer-to-peer activity.** Does peer-to-peer activity (morpheus, edonkey, bittorrent etc.) share certain properties that make it possible to detect new similar protocols by examining Netflow records?
3. **Strengthen other indicators.** Patterns in Netflow records may raise suspicion of misuses. Is it possible to find other indicators that can strengthen (or weaken) the suspicion, in order to reduce false positives?

# BIBLIOGRAPHY

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

[2] S.Axelson. *Intrusion Detection Systems: A survey and taxonomy* Technical Report No 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.
http://www.mnlab.cs.depaul.edu/seminar/spr2003/IDSSurvey.pdf

[3] Marcus Ranum. *False Positives: A User's Guide to Making Sense of IDS Alarms.* ICSA Labs IDSC, February 2003.
http://www.icsalabs.com/html/communities/ids/whitepaper/FalsePositives.pdf

[4] Paul Barford and David Plonka. *Characteristics of Network Traffic Flow Anomalies.*Proceedings of ACM SIGCOMM Internet Measurement Workshop, San Francisco, CA, November 2001.

[5] Xiaoxin Yin, William Yurcik, Yifan Li, Kiran Lakkaraju and Cristina Abad. *VisFlowConnect: Providing Security Situational Awareness by Visualizing Network Traffic Flows.* In Conference proceedings of the 2003 IEEE International Performance,Computing, and Communications Conference, Phoenix, Arizona, April 9-11 2003, pp. 601-608
http://www.ncassr.org/projects/sift/papers/wia2004ipccc.pdf

[6] William Yurcik, Kiran Lakkaraju, James Barlow and Jeff Rosendale. *A prototype Tool for Visual Data Mining of Network Traffic for Intrusion Detection.* In Workshop on Data Mining for Computer Security (DMSEC'03) Melbourne, Florida, November 19-22 2003
http://www.ncsa.uiuc.edu/People/jbarlow/publications/ICDM-DMSEC03.pdf

[7] Peter Mell, Vincent Hu, Richard Lippmann, Josh Haines, Marc Zissman *An overview of Issues in Testing Intrusion Detection Systems* NIST IR 7007.
http://csrc.nist.gov/publications/nistir/nistir-7007.pdf

[8] Juan M. Estevez-Tapiador, Pedro Garcia-Teodoro, Jesus E. Diaz-Verdejo. *Anomaly detection methods in wired networks: a survey and taxonomy.* Computer Communications, Volume 27, Issue 16, 15 October 2004, Pages 1569-1584

[9] Richard Lippmann *, Joshua W. Haines, David J. Fried, Jonathan Korba, Kumar Das *The 1999 DARPA off-line intrusion detection evaluation.* Computer Networks 34 (4) (1999) 579Ű595.

[10] John McHugh *The Lincoln Laboratories Intrusion Detection System Evaluation, A Critique.* Proceedings of RAID 2000, Toulouse, Fr. (Springer-Verlag LNCS 1907);

[11] John McHugh *Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory.* ACM TISSEC, 3(4) PP262-291 Nov. 2000 (longer version)

[12] E.Biermann, E.Cloete and L.M.Venter *A comparison of Intrusion Detection systems.* Computers Security 2001;20: 676Ű83.

[13] Cristina Abad and Yifan Li and Kiran Lakkaraju and Xiaoxin Yin and William Yurcik *Correlation between Netflow System and Network Views for Intrusion Detection.* In Workshop on Information Assurance (WIA04). Phoenix, Arizona, April 14-17, 2004 http://www.ncassr.org/projects/sift/papers/wia2004ipccc.pdf

[14] John McHugh *Sets, Bags, and Rock and Roll - Analyzing Large Data Sets of Network Data.* ESORICS 2004, LNCS 3193, pp.407-422, 2004.

[15] John D. Howard and Thomas A. Longstaff *A Common Language for Computer Security Incidents* Technical Report SAND98-8667, SNL, 1998.

[16] Martin Karresand *A proposed Taxonomy of Software Weapons.* Master's thesis in computer security, LITH December 2002.

[17] John Pinkston, Jeffrey Undercoffer, Anupam Joshi and Timothy Finin. *A Target-Centric Ontology for Intrusion Detection.* IJCAI'03, August, 2003, Acapulco MX. http://umbc.edu/ finin/papers/ijcai03ontologyWorkshop.pdf

[18] Gonzalo Álvarez and Slobodan Petrović *A taxonomy of Web Attacks.* LNCS, Vol. 2722, 2003, pp. 295-298

[19] Ulf Lindquist and Erland Jonsson *How to Systematically Classify Computer Security Intrusions.* In Proceedings of the 1997 IEEE Symposium on Security and Privacy, pages 154–163, Oakland, California, May 4–7, 1997. IEEE Computer Society Press, Los Alamitos, California.

[20] Jeffrey Undercoffer, Anupam Joshi and John Pinkston *Modeling Computer Attacks: An Ontology for Intrusion Detection.* In Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburg, PA, LNCS, 2820, p.113-135, 2003

[21] Ulf Lindquist *On the Fundamentals of Analysis and Detection of Computer Misuse.* Thesis for the degree of Doctor of Philosophy, Chalmers University of Technology, G  teborg, Sweden 1999

[22] Peter Troxell, Curry Bartlett, Nicholas Gill *Using network traffic analysis as a security tool.* Proc. of the 18th National Info. Systems Security Conference. Baltimore, MD. Oct 10-13 1995. P. 262-270

[23] Mark Fullmer, Suresh Ramachandran and Steve Romig *Cisco Flow Logs and Intrusion Detection at the Ohio State University.* Usenix ;login magazine. Special Issue:Intrusion Detection(Sept. 1999) http://www.usenix.org/publications/login/1999-9/osu.html

[24] R. Heady, G. Luger, A. Maccabe and M. Servilla. *The Architecture of a Network Level Intrusion Detection System.* Technical report, Department of Computer Science, University of New Mexico, August 1990

[25] Thorbjorn Axelsson *Network Security Auditing at Gigabit Speeds* Master Thesis, Chalmers University of Technology, Sweden, 22 july 2004

[26] Richard Bejtlich *The Tao of Network Security Monitoring - Beyond Intrusion Detection* Addison-Wesley
ISBN 0-321-24677-2

[27] Andrew Jones *Identification of a Method for the Calculation of Threat in an Information Environment* Internal publication, QinetiQ, Inc. April 2002.

[28] Flowscan (28 march 2005)
http://www.caida.org/tools/utilities/flowscan/

[29] *Security Analyst Network Connection Profiler* (20 march 2005)
http://www.metre.net/sancp.html

[30] *SGUIL - The Analyst Console for Network Security Monitoring* (20 march 2005)
http://sguil.sourceforge.net/

[31] *The Metasploit Project* (20 march 2005)
http://www.metasploit.com/

[32] RFC 2828 - Internet Security Glossary (1 nov 2004)
http://rfc.net/rfc2828.html

[33] RFC 793 - Transmission Control Protocol (29 april 2005)
http://rfc.net/rfc793.html

[34] *Argus - Audit record generation and utilization system.* (1 oct 2004)
http://www.qosient.com/argus/

[35] *PostgreSQL.* A free database system. (1 oct 2004)
http://www.postgresql.org/

[36] *JpGraph - An OO Graph Library for PHP* (1 oct 2004)
http://http://www.aditus.nu/jpgraph/

[37] PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. (1 oct 2004)
http://www.php.net/

[38] Perl is a stable, cross platform programming language.(1 oct 2004)
http://www.perl.org/

[39] *FreeBSD - An UNIXő like operating system for the Alpha/AXP, AMD64 and Intel EM64T, i386Ź IA-64, PC-98, and UltraSPARCő platforms based on U.C. Berkeley's "4.4BSD-Lite" release* (1 oct 2004)
http://www.freebsd.org/

[40] NetForensics - Security Information Management (1 nov 2004)
http://www.netforensics.com/

[41] *Snort - An open source network intrusion detection system.* (13 nov 2004)
http://www.snort.org/

[42] *Network Flight Recorder - A commersial Network Intrusion Detection System.* (13 nov 2004) http://www.nfr.com/

[43] *Cisco Intrusion Detection* (29 april 2005) http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/index.shtml

[44] *Alliance for Telecommunications Industry Solutions* (13 april 2005) http://www.atis.org/

[45] Andrew Mullican *Snort developer*. Personal communication.

# A EXPERIMENT SETUP

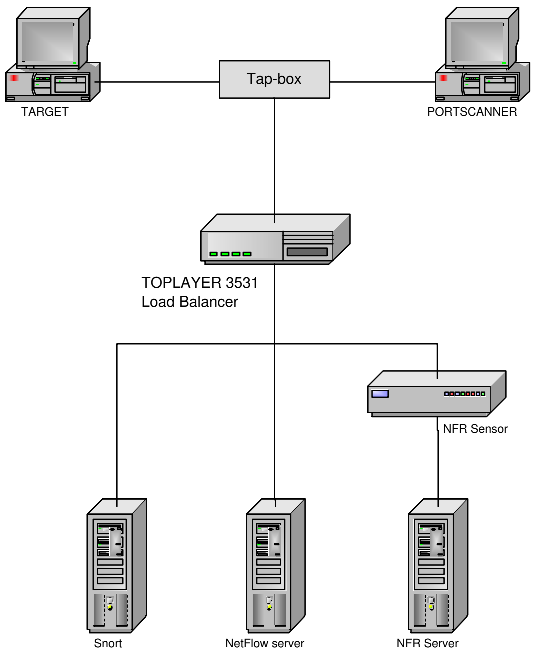

Figure 17: Experiment setup and component description

**Monitored network.**

The monitored system consists of a target and a scanner. A data set from [9] and a production network with approximately 200 users was replayed to give the illusion that the two mentioned hosts existed on an active computer network.

**NetOptics 10/100 tap-box**

The tap-box was used to ensure that the measurements did not influence the monitored network. No network packet was allowed to enter into the monitored network, and the tap-box did not cause packet delay in the monitored network. If power to the tap-box would fail, network traffic would still pass through it.

**TopLayer 3531 IDS Load balancer.**

An IDS Load balancer was used to feed multiple IDS systems the same network traffic. The IDS Load balancer receives network traffic at *input ports* and sends the traffic out on *monitor ports*. It is capable of sending 100Mb/sec traffic to any input port out on any monitor port. This ensures that all IDS systems 'see' the same traffic at the same time.

**Intrusion detection systems**

The Netflow analysis system and Snort were running on an Intel/FreeBSD platform. NFR was running on a Sun/Solaris system.

# B RESULTS OF THE LAB EXPERIMENT

*Experimental results - Round 1 - Snort with sfportscan.*

Snort is configured with sfportscan and no rules. The sensitivity level is varied from low to high.

| Iteration | Sensitivity | Snort true positives | Snort false positives |
|---|---|---|---|
| 1 | Low | 0 | 564 |
| 2 | Medium | 0 | 1218 |
| 3 | High | 1 | 308 |

Table 7: Results Snort sfportscan - Round 1 Lab experiment

*Experimental results - Round 2 - Network Flight Recorder.*

The TCP Portscan backend is enabled and all other backends are disabled. INTERVAL is set to 600 seconds, UNPRIV_WEIGHT=1, PRIV_WEIGHT=3, TRIGGER_PORTS=6. Trigger port list is not changed.

| Iteration | Threshold | NFR true positives | NFR false positives |
|---|---|---|---|
| 1 | 10 | 1 | 158 |
| 2 | 20 | 1 | 99 |
| 3 | 25 | 1 | 40 |
| 4 | 30 | 0 | 0 |
| 5 | 35 | 0 | 0 |
| 6 | 40 | 0 | 0 |

Table 8: Results Network Flight Recorder - Round 2 Lab experiment

*Experimental results - Round 3 - Netflow analysis system.*

Threshold varied from 2 to 2000 (distinct destination ports). The query includes all sessions (successful and unsuccessful).

| Iteration | Threshold | Netflow true positives | Netflow alse positives |
|---|---|---|---|
| 1 | 2 | 1 | 37 |
| 2 | 5 | 1 | 19 |
| 3 | 7 | 1 | 6 |
| 4 | 10 | 1 | 1 |
| 5 | 20 | 1 | 0 |
| 6 | 50 | 1 | 0 |
| 7 | 100 | 1 | 0 |
| 8 | 200 | 1 | 0 |
| 9 | 300 | 1 | 0 |
| 10 | 500 | 1 | 0 |
| 11 | 700 | 1 | 0 |
| 12 | 1000 | 1 | 0 |
| 13 | 1500 | 0 | 0 |
| 14 | 2000 | 0 | 0 |

Table 9: Results Netflow - Round 3 Lab experiment

*Experiment results - Round 4 - Netflow analysis system with additional calculations.*

Threshold varied from 2 to 2000 (distinct destination ports). The query includes only unsuccessful sessions.

| Iteration | Threshold | Netflow true positives | Netflow false positives |
|---|---|---|---|
| 1 | 2 | 1 | 34 |
| 2 | 5 | 1 | 19 |
| 3 | 7 | 1 | 6 |
| 4 | 10 | 1 | 1 |
| 5 | 20 | 1 | 0 |
| 6 | 50 | 1 | 0 |
| 7 | 100 | 1 | 0 |
| 8 | 200 | 1 | 0 |
| 9 | 300 | 1 | 0 |
| 10 | 500 | 1 | 0 |
| 11 | 700 | 1 | 0 |
| 12 | 1000 | 1 | 0 |
| 13 | 1500 | 0 | 0 |
| 14 | 2000 | 0 | 0 |

Table 10: Results Netflow Round 4 - Lab experiment

# C RESULTS OF THE FIELD EXPERIMENT

*Experimental results - Round 1 - Snort with sfportscan.*

Snort is configured with sfportscan and no rules. The sensitivity level is varied from low to high.

| Iteration | Sensitivity | Snort true positives | Snort false positives |
|---|---|---|---|
| 1 | Low | 0 | 235 |
| 2 | Medium | 0 | 1195 |
| 3 | High | 1 | 1064 |

Table 11: Results snort sfportscan - Round 1 Field experiment

*Experimental results - Round 2 - Network Flight Recorder.*

The TCP Portscan backend is enabled and all other backends are disabled. INTERVAL is set to 600 seconds, UNPRIV_WEIGHT=1, PRIV_WEIGHT=3, TRIGGER_PORTS=6. Trigger port list is not changed.

| Iteration | Threshold | NFR true positives | NFR false positives |
|---|---|---|---|
| 1 | 10 | 1 | 99 |
| 2 | 20 | 1 | 99 |
| 3 | 25 | 1 | 99 |
| 4 | 30 | 1 | 17 |
| 5 | 35 | 1 | 15 |
| 6 | 40 | 1 | 1 |
| 7 | 45 | 0 | 0 |

Table 12: Results Network Flight Recorder - Round 2 Field experiment

*Experimental results - Round 3 - Netflow analysis system.*

Threshold varied from 2 to 2000 (distinct destination ports). The query includes all sessions (successful and unsuccessful).

| Iteration | Threshold | Netflow true positives | Netflow false positives |
|---|---|---|---|
| 1 | 2 | 1 | 52 |
| 2 | 5 | 1 | 8 |
| 3 | 7 | 1 | 1 |
| 4 | 10 | 1 | 0 |
| 5 | 20 | 1 | 0 |
| 6 | 50 | 1 | 0 |
| 7 | 100 | 1 | 0 |
| 8 | 200 | 1 | 0 |
| 9 | 300 | 1 | 0 |
| 10 | 500 | 1 | 0 |
| 11 | 700 | 1 | 0 |
| 12 | 1000 | 1 | 0 |
| 13 | 1500 | 0 | 0 |
| 14 | 2000 | 0 | 0 |

Table 13: Results Netflow - Round 3 Field experiment

*Experimental results - Round 4 - Netflow analysis system with additional calculations*

Threshold varied from 2 to 2000 (distinct destination ports). The query includes only unsuccessful sessions.

| Iteration | Threshold | Netflow true positives | Netflow false positives |
|---|---|---|---|
| 1 | 2 | 1 | 37 |
| 2 | 5 | 1 | 5 |
| 3 | 7 | 1 | 0 |
| 4 | 10 | 1 | 0 |
| 5 | 20 | 1 | 0 |
| 6 | 50 | 1 | 0 |
| 7 | 100 | 1 | 0 |
| 8 | 200 | 1 | 0 |
| 9 | 300 | 1 | 0 |
| 10 | 500 | 1 | 0 |
| 11 | 700 | 1 | 0 |
| 12 | 1000 | 1 | 0 |
| 13 | 1500 | 0 | 0 |
| 14 | 2000 | 0 | 0 |

Table 14: Results Netflow - Round 4 Field experiment

# D STANDARD DEVIATION CALCULATIONS

| Source IP | Destination port count | Number of sessions | Standard deviation |
|---|---|---|---|
| 172.26.240.100 | 1000 | 1000 | 2.68 |
| 172.16.114.148 | 11 | 8227 | 20.95 |
| 196.37.75.158 | 7 | 3805 | 22.73 |
| 194.7.248.153 | 7 | 4864 | 20.40 |
| 194.27.251.21 | 7 | 5497 | 18.47 |
| 197.182.91.233 | 7 | 6094 | 16.30 |
| 172.16.114.168 | 7 | 5093 | 16.87 |
| 172.16.113.84 | 6 | 6763 | 14.75 |
| 172.16.113.105 | 6 | 10443 | 17.97 |
| 172.16.113.204 | 6 | 5758 | 17.93 |
| 195.73.151.50 | 6 | 2773 | 23.99 |
| 135.13.216.191 | 6 | 5633 | 15.85 |
| 135.8.60.182 | 6 | 3556 | 21.08 |
| 196.227.33.189 | 6 | 4786 | 20.57 |
| 197.218.177.69 | 6 | 4406 | 24.39 |
| 195.115.218.108 | 6 | 3705 | 24.70 |
| 172.16.112.207 | 6 | 12183 | 17.42 |
| 172.16.114.169 | 5 | 6109 | 19.25 |
| 172.16.112.149 | 5 | 2926 | 25.07 |
| 172.16.114.207 | 5 | 5370 | 18.92 |
| 172.16.112.194 | 4 | 4077 | 25.84 |
| 172.16.114.50 | 4 | 518 | 50.29 |

Table 15: Netflow - Results standard deviation calculation lab experiment

| Source IP | Destination port count | Number of sessions | Standard deviation |
|---|---|---|---|
| 172.16.112.100 | 3 | 191 | 62.94 |
| 172.16.117.103 | 3 | 2534 | 65.45 |
| 172.16.112.50 | 3 | 1134 | 58.63 |
| 172.16.116.201 | 3 | 5398 | 49.03 |
| 172.16.117.132 | 3 | 7707 | 39.41 |
| 192.168.1.30 | 2 | 243 | 99.35 |
| 172.16.113.50 | 2 | 54 | 123.64 |
| 192.168.1.90 | 2 | 1777 | 37.69 |
| 172.16.115.5 | 2 | 5778 | 56.76 |
| 172.16.115.87 | 2 | 10897 | 35.77 |
| 172.16.115.234 | 2 | 9571 | 39.27 |
| 172.16.116.44 | 2 | 6512 | 50.31 |
| 172.16.116.194 | 2 | 5712 | 48.68 |
| 172.16.117.52 | 2 | 5010 | 50.34 |
| 172.16.117.111 | 2 | 2994 | 58.65 |
| 172.16.112.20 | 2 | 6821 | 25.07 |
| 216.40.24.2 | 1 | 27 | 108.85 |
| 80.2.16.0 | 1 | 1 | 108.85 |
| 132.25.1.25 | 1 | 27 | 108.85 |
| 134.205.165.120 | 1 | 27 | 108.85 |
| 172.16.112.10 | 1 | 891 | 29.99 |
| 192.64.204.230 | 1 | 27 | 109.87 |
| 192.168.1.10 | 1 | 1720 | 27.77 |
| 192.168.1.20 | 1 | 405 | 74.37 |
| 205.181.112.72 | 1 | 27 | 127.91 |
| 205.185.55.49 | 1 | 27 | 127.91 |
| 206.43.200.71 | 1 | 27 | 127.91 |
| 206.79.21.13 | 1 | 27 | 127.91 |
| 209.67.3.82 | 1 | 27 | 127.91 |
| 209.67.29.11 | 1 | 27 | 127.91 |
| 209.185.191.232 | 1 | 27 | 127.91 |
| 0.0.0.0 | 1 | 1 | 127.91 |

Table 16: Netflow - Results standard deviation calculation lab experiment (continued)

| Source IP | Destination port count | Number of sessions | Standard deviation |
|---|---|---|---|
| 172.26.240.100 | 1000 | 1000 | 2.68 |
| 158.112.96.108 | 7 | 5214 | 38.91 |
| 144.84.15.123 | 6 | 7473 | 13.19 |
| 158.112.96.181 | 6 | 2624 | 50.74 |
| 158.112.96.165 | 6 | 4011 | 18.72 |
| 144.84.15.51 | 5 | 981 | 120.06 |
| 158.112.96.207 | 5 | 1970 | 136.06 |
| 144.84.15.117 | 5 | 1903 | 43.04 |
| 158.112.96.138 | 5 | 5366 | 49.37 |
| 193.45.3.9 | 4 | 126 | 197.42 |
| 144.84.15.53 | 4 | 4212 | 57.16 |
| 158.112.97.102 | 4 | 2541 | 137.82 |
| 158.112.96.194 | 4 | 3635 | 69.89 |
| 158.112.96.151 | 3 | 2376 | 193.24 |
| 158.112.96.178 | 3 | 2676 | 45.59 |
| 144.84.15.55 | 3 | 5199 | 17.72 |
| 158.112.96.159 | 3 | 2030 | 106.46 |
| 10.0.0.21 | 3 | 3528 | 135.09 |
| 158.112.96.4 | 3 | 235 | 147.70 |
| 144.84.15.2 | 3 | 4124 | 20.90 |
| 158.112.96.7 | 3 | 52 | 128.43 |
| 158.112.96.8 | 3 | 4632 | 18.87 |
| 158.112.96.102 | 3 | 4287 | 74.49 |
| 158.112.96.121 | 3 | 2933 | 139.37 |
| 158.112.96.124 | 3 | 6107 | 9.37 |
| 158.112.96.126 | 3 | 2249 | 189.20 |
| 158.112.96.127 | 3 | 2556 | 88.53 |
| 158.112.96.131 | 3 | 114 | 184.19 |
| 10.0.0.23 | 3 | 7803 | 51.23 |
| 158.112.96.137 | 3 | 701 | 83.84 |
| 158.112.96.141 | 3 | 6093 | 32.26 |
| 192.168.200.6 | 3 | 13625 | 7.09 |
| 158.112.96.156 | 2 | 2460 | 51.40 |
| 158.112.96.158 | 2 | 313 | 131.52 |
| 158.112.96.166 | 2 | 520 | 149.28 |
| 158.112.96.174 | 2 | 1299 | 73.21 |
| 10.0.0.22 | 2 | 5620 | 66.29 |
| 158.112.96.221 | 2 | 3920 | 47.12 |
| 158.112.96.222 | 2 | 568 | 145.69 |
| 144.84.15.57 | 2 | 1050 | 64.17 |
| 158.112.96.232 | 2 | 78 | 165.82 |
| 158.112.97.134 | 2 | 750 | 91.01 |
| 66.117.33.44 | 2 | 13 | 179.92 |
| 193.45.3.38 | 2 | 78 | 164.78 |
| 193.45.3.39 | 2 | 98 | 175.36 |
| 81.93.160.181 | 2 | 26 | 204.49 |
| 144.84.15.116 | 2 | 1533 | 45.71 |
| 66.117.33.59 | 2 | 13 | 161.84 |
| 158.112.96.106 | 2 | 101 | 157.02 |
| 158.112.96.116 | 2 | 1623 | 92.92 |
| 158.112.96.130 | 2 | 1569 | 116.99 |
| 158.112.96.136 | 2 | 1800 | 128.14 |
| 158.112.96.148 | 2 | 214 | 196.96 |
| 158.112.96.157 | 2 | 961 | 122.72 |

Table 17: Netflow - Results standard deviation calculation field experiment

| Source IP | Destination port count | Number of sessions | Standard deviation |
|---|---|---|---|
| 217.199.180.251 | 1 | 13 | 197.86 |
| 10.0.0.10 | 1 | 2491 | 57.78 |
| 64.233.161.99 | 1 | 13 | 165.69 |
| 64.233.161.104 | 1 | 13 | 165.69 |
| 65.54.184.250 | 1 | 13 | 165.69 |
| 66.117.33.58 | 1 | 13 | 165.69 |
| 67.159.5.116 | 1 | 1028 | 54.10 |
| 80.2.4.0 | 1 | 1 | 54.10 |
| 80.239.62.202 | 1 | 13 | 164.35 |
| 128.39.140.38 | 1 | 13 | 164.35 |
| 129.240.154.8 | 1 | 1105 | 52.86 |
| 144.84.15.54 | 1 | 204 | 160.01 |
| 144.84.15.68 | 1 | 117 | 122.60 |
| 144.84.15.105 | 1 | 65 | 163.16 |
| 144.84.15.106 | 1 | 351 | 72.20 |
| 144.84.15.119 | 1 | 3025 | 53.13 |
| 144.84.15.122 | 1 | 132 | 126.42 |
| 158.112.96.3 | 1 | 288 | 125.01 |
| 158.112.96.36 | 1 | 279 | 128.77 |
| 158.112.96.113 | 1 | 13 | 201.56 |
| 158.112.96.179 | 1 | 13 | 201.56 |
| 158.112.96.223 | 1 | 13 | 201.56 |
| 158.112.96.225 | 1 | 13 | 201.56 |
| 192.168.150.2 | 1 | 78 | 139.19 |
| 193.45.3.23 | 1 | 39 | 199.68 |
| 193.45.3.48 | 1 | 39 | 199.79 |
| 193.71.196.222 | 1 | 13 | 236.98 |
| 194.103.154.133 | 1 | 13 | 236.98 |
| 195.18.161.155 | 1 | 13 | 236.98 |
| 205.158.62.86 | 1 | 13 | 236.98 |
| 209.225.34.135 | 1 | 13 | 236.98 |
| 212.130.37.178 | 1 | 13 | 236.98 |
| 217.13.4.150 | 1 | 13 | 236.98 |
| 0.0.0.0 | 1 | 1 | 236.98 |

Table 18: Netflow - Results standard deviation calculation field experiment (continued)

# E NETFLOW ANALYSIS SYSTEM CODE

Some code was developed in the preparations of this thesis. The focus has been to develop a prototype, not to implement the most robust or optimal code.

This appendix includes code that is necessary in order to make the Netflow system functional. It does not include guidance for installing or configuring FreeBSD, PostgreSQL, php or jpgraph.

| Filename | Description |
|---|---|
| rarc | *ra* configuration file |
| argus_datafeed | Perl-script that reads Argus-files and inserts records into the database. |
| argus_datamove | Shell-script that breaks Argus' output file into manageable chunks. |
| stddev.php | PHP code that calculates standard deviation for time intervals between sessions coming from each source IP |
| stddev_bars.php | PHP/Jpgraph code that makes a bar-plot for the standard deviation for time intervals between sessons coming from each source IP |
| portscan.php | PHP code that makes an x-y plot for portscan detection |
| crontab | Alterations done to crontab |

Table 19: Files used by the Netflow system

**crontab (alterations only):**

```
# /etc/crontab - root's crontab for FreeBSD
#
# do daily/weekly/monthly maintenance
*/30    *       *       *       *       root    /usr/local/etc/argus_datamove
*/30    *       *       *       *       argus   /usr/local/etc/argus_datafeed
```

**argus_datamove:**

```
# Breaks argus.out into smaller chunks. Run from crontab

#!/bin/sh

tmp=`date "+%Y%m%d%H%M%S"`;
/bin/mv /usr/tmp/argus.out /usr/argus.data/argus.out."$tmp";
/usr/sbin/chown argus /usr/argus.data/*;
```

**rarc:**

```
RA_ARGUS_SERVERPORT=561
RA_RUN_TIME=0
RA_PRINT_LABELS=0
RA_FIELD_SPECIFIER="startime␣proto␣saddr␣sport␣dir␣daddr␣dport␣pkts␣bytes␣status"
RA_FIELD_DELIMITER=''
RA_PRINT_SUMMARY=no
RA_PRINT_HOSTNAMES=no
RA_PRINT_RESPONSE_DATA=no
RA_PRINT_UNIX_TIME=no
RA_TIME_FORMAT="%y-%m-%d␣%T"
RA_USEC_PRECISION=0
RA_USERDATA_ENCODE=Ascii
RA_DEBUG_LEVEL=0
RA_HOST_FIELD_LENGTH=15
RA_PORT_FIELD_LENGTH=6
```

**argus_datafeed:**

```
#!/usr/bin/perl

use Pg;

$basedir = '/usr/argus.data';

opendir (DIR, $basedir) || die;
@files = grep (!/^\./, readdir (DIR));
closedir (DIR);

open (LOG, ">/usr/tmp/nf.load.$$") || die;

foreach $file (sort @files) {
        print LOG "Beginning␣load␣of␣$basedir/$file:␣";
        open (RA,"/usr/local/bin/ra␣-F␣/usr/local/etc/rarc␣-cn␣-r␣$basedir/$file␣|") || die;
        open (OUT, ">/usr/tmp/$file.copy") || die;
        $lines = 0;
        while (<RA>) {
                chomp;
                ($date, $time, $proto, $src_tmp, $dir, $dst_tmp, $src_pkt, $dst_pkt, $src_bytes,
                     $dst_bytes, $state) = split;
                next unless ($proto eq 'tcp' || $proto eq 'udp' || $proto eq 'icmp');

                if ($proto eq 'tcp' || $proto eq 'udp') {
                        @t = split (/\./, $src_tmp); $src_port = pop @t; $src = join ('.', @t);
                        @t = split (/\./, $dst_tmp); $dst_port = pop @t; $dst = join ('.', @t);
                } elsif ($proto eq 'icmp') {
                        $src_port = 'NULL';
                        $dst_port = 'NULL';
                }

                $timestamp = "20$date␣$time";
                print OUT join ('␣␣␣␣␣␣␣', $timestamp, $proto, $src, $src_port, $dir, $dst, $dst_port,
                                    $src_pkt, $dst_pkt, $src_bytes,
                $dst_bytes, $state)."\n";
                $lines++;
                undef @f;
        }

        close (RA);
        close (OUT);
        print LOG "$lines\n";
        $total_all += $lines;

        open (PSQL, ">/usr/tmp/psql.$$");
        print PSQL "\\copy␣flow␣from␣'/usr/tmp/$file.copy'␣with␣null␣as␣'NULL'\n";
        close (PSQL);
        system ("/usr/local/bin/psql␣-f␣/usr/tmp/psql.$$␣argus");
        unlink ("/usr/tmp/psql.$$");
        unlink ("/usr/tmp/$file.copy");
        rename ("$basedir/$file", "/usr/argus.z/$file");
}

print LOG "Totalt␣$total_all␣rader.\n";
```

**portscan.php:**

```
<?php
include ("../jpgraph/jpgraph.php");
include ("../jpgraph/jpgraph_scatter.php");

$connection = pg_pconnect("dbname=argus␣user=argus");
  if (!$connection) { print("Connection␣Failed."); exit; }
  $res = pg_exec ($connection, "SELECT␣extract␣(epoch␣from␣time),dst_port␣FROM␣flow␣WHERE␣state='RST'");

  for ($lt = 0;$lt <pg_numrows($res);$lt++) {
        $datax[]= pg_result($res, $lt, 0);
        $datay[]= pg_result($res, $lt, 1);
}


# Jpgraph makes the x-y plot:

$graph = new Graph(800,600,"png");
$graph->SetScale("linlin");
$graph->xaxis->SetLabelAngle(45);
$graph->xaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
$graph->yaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
$graph->yaxis->title->Set("Destination␣port");
$graph->xaxis->title->Set("Time");
$graph->xaxis->title->SetFont(FF_VERDANA,FS_NORMAL);
$graph->yaxis->title->SetFont(FF_VERDANA,FS_NORMAL);
$graph->title->Set("Portscan␣detection");
$graph->title->SetFont(FF_VERDANA,FS_BOLD,24);
$graph->title->SetColor('#006699');
$graph->SetMarginColor('white');
$graph->SetFrame(false);
```

```
$graph->SetAlphaBlending();
$sp1 = new ScatterPlot($datay,$datax);
$sp1->mark->SetType(MARK_FILLEDCIRCLE);
$sp1->mark->SetFillColor("red");
$sp1->mark->SetWidth(2);
$graph->Add($sp1);
$graph->Stroke();
?>
```

**stddev.php:**

```
<?php
function standard_deviation($array) {

    //Get sum of array values
    while(list($key,$val) = each($array)) {
        $total += $val;
    }

    reset($array);
    $mean = $total/count($array);

    while(list($key,$val) = each($array)) {
        $sum += pow(($val-$mean),2);
    }
    $var = sqrt($sum/(count($array)-1));

    return $var;
}

$connection = pg_pconnect("dbname=argus␣user=argus");
  if (!$connection) { print("Connection␣Failed."); exit; }
  $res = pg_exec ($connection, "SELECT␣src_ip,count(distinct␣dst_port)␣from␣flow␣group␣by␣src_ip
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣order␣by␣count(distinct␣dst_port)␣desc");

print("<html><body>");

print("<table><tr><td>Source␣IP</td><td>Dest␣port␣count</td><td>Number␣of␣sessions</td><td>
␣␣␣␣␣␣␣␣␣Std.dev</td></tr>");

  for ($lt = 0;$lt <pg_numrows($res);$lt++) {
        $src_ip= pg_result($res, $lt, 0);
        $count= pg_result($res, $lt, 1);

  $res2 = pg_exec($connection, "SELECT␣distinct␣extract␣(epoch␣from␣time)␣as␣test␣FROM␣flow
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣where␣src_ip='$src_ip'␣order␣by␣test");

  $n=pg_numrows($res2);

  #Put all unix timestamps in an array
  for ($a=0;$a<pg_numrows($res2);$a++){
        $temp[$a]= pg_result($res2,$a,0);
  }

  #Calculate delta between timestamps
  for($i=0;$i<pg_numrows($res2)-1;$i++){
        $delta[$i]=$temp[$i+1]-$temp[$i];
  }

$var = standard_deviation($delta);
echo "<tr><td>$src_ip</td><td>$count</td><td>$n<td>$var</td></tr>";
}

print("</table></html></body>");
?>
```

**stddev_bars.php:**

```
<?php
  include ("../jpgraph/jpgraph.php");
  include ("../jpgraph/jpgraph_bar.php");

function standard_deviation($array) {

    //Get sum of array values
    while(list($key,$val) = each($array)) {
        $total += $val;
    }

    reset($array);
    $mean = $total/count($array);

    while(list($key,$val) = each($array)) {
        $sum += pow(($val-$mean),2);
    }
    $var = sqrt($sum/(count($array)-1));

    return $var;
}


$connection = pg_pconnect("dbname=argus␣user=argus");
  if (!$connection) { print("Connection␣Failed."); exit; }
  $res = pg_exec ($connection, "SELECT␣src_ip,count(distinct␣dst_port)␣from␣flow␣group␣by␣src_ip
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣order␣by␣count(distinct␣dst_port)␣desc");


  for ($lt = 0;$lt <pg_numrows($res);$lt++) {
        $src_ip[]= pg_result($res, $lt, 0);


  $res2 = pg_exec($connection, "SELECT␣distinct␣extract␣(epoch␣from␣time)␣as␣test␣FROM␣flow
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣where␣src_ip='$src_ip[$lt]'␣order␣by␣test");

  $n=pg_numrows($res2);

  #Put all unix timestamps in an array
  for ($a=0;$a<pg_numrows($res2);$a++){
        $temp[$a]= pg_result($res2,$a,0);
  }

  #Calculate delta between timestamps
  for($i=0;$i<pg_numrows($res2)-1;$i++){
        $delta[$i]=sprintf('%.2f',$temp[$i+1]-$temp[$i]);

  }
$var[] = standard_deviation($delta);


}

  #Jpgraph draws the graph:

  $graph = new Graph(2048,1024,'png');
  $graph->SetScale("textlin");
  $graph->xaxis->SetTickLabels($src_ip);
  $graph->xaxis->SetLabelAngle(45);
  $graph->xaxis->HideTicks();
  $graph->xaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
  $graph->yaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
```

```
    $graph->yaxis->title->Set("Seconds");
    $graph->xaxis->title->Set("Source␣IP");
    $graph->xaxis->title->SetFont(FF_VERDANA,FS_NORMAL);
    $graph->yaxis->title->SetFont(FF_VERDANA,FS_NORMAL);
    $graph->title->Set("Standard␣deviation");
    $graph->title->SetFont(FF_VERDANA,FS_BOLD,24);
    $graph->title->SetColor('#006699');
    $graph->SetMarginColor('white');
    $graph->SetFrame(false);
    $graph->SetAlphaBlending();
    $bar = new BarPlot($var);
    $bar->SetLegend("Stddev␣for␣interval␣between␣sessions␣from␣Source␣IP");
    $bar->SetShadow('#cccccc');
    $bar->SetFillColor('#006699');
    $bar->SetValuePos('center');
    $graph->Add($bar);
    $graph->Stroke();
?>
```

# F TUNING SFPORTSCAN

(from README.sfportscan)

The most important aspect in detecting portscans is tuning the detection engine for your network(s). Here are some tuning tips:

1. Use the watch_ip, ignore_scanners, and ignore_scanned options. It's important to correctly set these options. The watch_ip option is easy to understand. The analyst should set this option to the list of Cidr blocks and IPs that they want to watch. If no watch_ip is defined, sfPortscan will watch all network traffic. The ignore_scanners and ignore_scanned options come into play in weeding out legitimate hosts that are very active on your network. Some of the most common examples are NAT IPs, DNS cache servers, syslog servers, and nfs servers. sfPortscan may not generate false positives for these types of hosts, but be aware when first tuning sfPortscan for these IPs. Depending on the type of alert that the host generates, the analyst will know which to ignore it as. If the host is generating portsweep events, then add it to the ignore_scanners option. If the host is generating portscan alerts (and is the host that is being scanned), add it to the ignore_scanned option.

2. Filtered scan alerts are much more prone to false positives.
   When determining false positives, the alert type is very important. Most of the false positives that sfPortscan may generate are of the filtered scan alert type. So be much more suspicious of filtered portscans. Many times this just indicates that a host was very active during the time period in question. If the host continually generates these types of alerts, add it to the ignore_scanners list or use a lower sensitivity level.

3. Make use of the Priority Count, Connection Count, IP Count, Port Count, IP range, and Port range to determine false positives. The portscan alert details are vital in determining the scope of a portscan and also the confidence of the portscan. In the future, we hope to automate much of this analysis in assigning a scope level and confidence level, but for now the user must manually do this. The easiest way to determine false positives is through simple ratio estimations. The following is a list of ratios to estimate and the associated values that indicate a legitimate scan and not a false positive. Connection Count / IP Count: This ratio indicates an estimated average of connections per IP. For portscans, this ratio should be high, the higher the better. For portsweeps, this ratio should be low. Port Count / IP Count: This ratio indicates an estimated average of ports connected to per IP. For portscans, this ratio should be high and indicates that the scanned host's ports were connected to by fewer IPs. For portsweeps, this ratio should be low, indicating that the scanning host connected to few ports but on many hosts. Connection Count / Port Count: This ratio indicates an estimated average of connections per port. For portscans, this ratio should be low.

This indicates that each connection was to a different port. For portsweeps, this ratio should be high. This indicates that there were many connections to the same port. The reason that Priority Count is not included, is because the priority count is included in the connection count and the above comparisons take that into consideration. The Priority Count plays an important role in tuning because the higher the priority count the more likely it is a real portscan or portsweep (unless the host is firewalled).

4. If all else fails, lower the sensitivity level. If none of these other tuning techniques work or the analyst doesn't have the time for tuning, lower the sensitivity level. You get the best protection if the sensitivity level is higher, but it's also important that the portscan detection engine generates alerts that the analyst will find informative. The low sensitivity level only generates alerts based on error responses. These responses indicate a portscan and the alerts generated by the low sensitivity level are highly accurate and require the least tuning. The low sensitivity level does not catch filtered scans, since these are more prone to false positives.