M.Sc. Thesis

Forensic examination of log files

IMM-THESIS-2005-2

Written by
Jóan Petur Petersen (s022087)

Supervised by
Robin Sharp

IMM

Informatics and Mathematical Modelling
Technical University of Denmark
January 31, 2005

Preface

This master's thesis is written to fulfill the requirements to obtain a degree in Master of Science. The project is done in the Informatics and Mathematical Modelling department at the Technical University of Denmark (DTU). The project is called "Forensic examination of log files", and was carried out through a 5 month period, starting September 1, 2004, and finishing January 31, 2005. The project was requested by the Danish CERT, carried out by Jóan Petur Petersen, and supervised by Prof. Robin Sharp.

Acknowledgements

I would like to thank the Danish CERT for the support and data it provided for the project. I would especially like to thank Mikael Stamm, Preben Andersen, and Morten Schiønning. Finally I would like to thank my supervisor, Robin Sharp, for his guidance and support, and all the interesting discussions.

DTU, 31 January 2005
Jóan Petur Petersen

Abstract

Forensic examination of logs plays a big role in modern computer security, but it has become a time consuming and daunting task due to the sheer amount of data involved. It is therefore necessary to make specialized tools to aid the investigation, so that the digital evidence can be extracted in a fast and efficient manner.

In this thesis a system is developed that can identify malicious traffic in router logs on a log entry level. This is done using specialized feature extractors and a classifier based on a neural network. The system is developed for Netflow logs, and problem associated with flows are investigated, such as how unidirectional flows should be handled. As a proof of concept, the system is developed to detect host scans. This is done using real router log data, and log data derived from the 1999 DARPA Intrusion Detection Evaluation data set. The system could easily be extended to detect other kinds of malicious traffic, such as Denial of Service attacks and probes other than the host scan. New contributions in this thesis are use of artificial neural networks to classify router logs, classification of each log entry, and development of feature extractors for Netflow logs.

Keywords: Network Forensics, Log Analysis, NetFlow, Probing, Denial of Service, Flow Classification, Feature Extraction, Traffic Aggregation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the introduction of the personal computer (PC) in the late 70's, and its widespread adoption through the 80's, resulted in a new forensic science, which was called computer forensics. Computer forensic deals with the collecting of digital evidence from digital devices, such as laptops, home computers, PDAs, and servers.

Another forensic field called network forensics was introduced with the wide adoption of the Internet in the 90's. This forensic field examines network traffic for digital evidence. This could for example be examination and analysis of router logs, firewall logs, or eavesdropped data from a network.

In this thesis we will be developing a forensic tool suite that can aid an investigator in examining and analyzing a router log. The tool suite therefore belongs to network forensics. The forensic tool suite will be able to manipulate log files, and to identify different types of malicious traffic within the log. Throughout the thesis the system will be developed gradually, and problem associated with analyzing router logs will be examined.

Computer and network forensics are not only of interest to law enforcement, but also companies, the military, and to some degree private users. Computer and network forensics should be part of a company's security policy, so that they can analyze incidents if they should occur.

The tool suit is made for the Danish CERT (Computer Emergency Response Team), which is called DK•CERT for short. DK•CERT provides security related services to Danish companies, such as incident response, reporting of new vulnerabilities, and security surveys. DK•CERT is a division of Forskningsnettet (Danish Research Network), which is a joint research network. Forskningsnettet provides services to research institutes and to companies doing research related activities. Technically Forskningsnettet consists of a backbone which connects the main points of the network in Lyngby, Odense, Århus, and Ålborg. The capacity of the backbone network is 822 Mbits/s. Forskningsnettet is connected to other ISPs through DIX (the Danish Internet eXchange point). The purpose of DIX is to handle the traffic between the networks that constitute the Danish part of the Internet. Internet access is gained through the other ISPs. A more detailed description of Forskningsnettet can be found on their web page [2].

DK•CERT has access to many routers through Forskningsnettet, and could therefore gain a considerable amount of information with a tool suite that could analyze router logs. For this project DK•CERT has provided several days of traffic from one of their routers (the Lyngby 2 router). This data will be used to develop and test the forensic tools in this thesis.

Most of the routers that DK•CERT uses, are Cisco routers, which use a log sheme developed by Cisco called NetFlow. The forensic tool suite is only developed for this log format, but even though, it should be possible to modify it, so it can support other router log schemes too.

The rest of the chapter is organized as follows: First we will look at what role routers play in the network, and what their logs might provide. The we examine the NetFlow router log scheme closer, and look at some of the more technical details. Then we briefly look at the difference between postmortem and real-time analysis of logs. The chapter ends with an examination of related work in this area.

1.1 Routers

Now we will look at what the role of the router is on the network, and why it is a good place to look for digital evidence.

Routers play an important role in most computer networks. Routers are responsible for getting the packets forwarded to the right destination; which is mostly a collaborative task between many routers.

Often routers are placed at borders of networks, edge routers, connecting the internal network to the outside, which could be the Internet. The router is therefore the first device an attacker encounters when attacking a network. All the attack traffic also goes through the router, and it is therefore a good centralized place to search for evidence.

Important information could be found by examining router logs. The ISP router logs can give detailed information on the traffic on the Internet in general, and reveal attacks against the ISP's customers. Edge routers also can give information on attacks to and from networks.

Many routers have the ability to create logs of the traffic that pass through them. The log files can be used for traffic measurements, billing systems, security, amongst many other things. There are many security applications of such logs; it is possible to determine services running on the hosts, probing, denial of service attacks, just to name a few. Probing is an technique used by attackers to gain information about the hosts on the system.

It is not practical to log all the information that gets forwarded by the routers, because the storage space needed would be excessive, even just storing the header of the packets being forwarded would require several GB of space. Therefore it is normal to aggregate the traffic, so that only the essential information is stored. Aggregation groups similar traffic together, and stores information about each group as a whole. Aggregated data for network traffic could for example be the average packet size, transferred octets, and duration of the traffic grouped together. The aggregation is a tradeoff between the logged volume and how fine grained the log is. Finding the right balance is hard, especially if the same aggregation scheme is used for multiple applications. The aggregation used varies between router manufacturers, but they mostly have the same features. In this thesis we will look at the Cisco's router log scheme, NetFlow, which does aggregation of the network traffic. We will look at the details of how NetFlow works in the next section.

1.2 Log files

In this section we will look at router logs, especially the Cisco NetFlow logs. Many other router log schemes exist, but most of them have much in common with NetFlow.

Routers work on the network layer (OSI layer 3), so they only see the traffic as a man in the middle. But the routers can store information from the higher layers to the log, such as the TCP header from the transport layer (OSI layer 4).

Due to the large amount of data that is goes through a high-end router, it is not very realistic to store all the data, not even the data in the TCP/IP headers. Similar packets are therefore aggregated into records, so that they take up less space (in most cases). There are many ways to aggregate traffic, but we will only look at how it is done in case of NetFlow.

NetFlow is scheme devised by Cisco Networks for generating router logs. It consists mainly of an aggregation method and a datagram for exporting the logs from the router. Its main purposes are network traffic accounting, usage-based network accounting, network planning, and security.

Netflow aggregates network traffic by grouping packets into flows. A NetFlow flow is defined as an unidirectional sequence of packets between two points. Flows are uniquely identified by source and destination IP addresses, source and destination ports, protocol type (layer 3), the input interface, and the TOS field.

A flow is uniquely identified on the router by the tuple given in equation 1.1.

$$flow = (src_{addr}, src_{port}, dst_{addr}, dst_{port}, ip_{prot}, src_{if}, tos) \tag{1.1}$$

For each flow on the router, there exists a record that contains the accumulated information about the flow. The aggregated NetFlow data is about 33.3 times smaller than the full packet trace [3].

There are several NetFlow versions, as it has been modified over time to satisfy the needs as they occurred. The different versions all work more or less in the same way, and the only difference is some variation in the information supplied.

Typical information found in NetFlow data records are [4]:

- Export timestamp
- Start and end timestamps
- Source and destination IP addresses
- Source and destination TCP/User Datagram Protocol (UDP) ports
- Type of service (ToS)
- Packet and byte counts
- Input and output interface numbers
- TCP flags
- Routing information (next-hop address, source autonomous system (AS) number, destination AS number, source prefix mask, destination prefix mask)

Each router has a cache where the flow records are stored. The size of the flow cache depends on the router's memory and setup. Each time the router encounters a new flow identified by equation 1.1, a new record will be added to the cache. Any successors with the same identification will be use to update some of the fields: The number of packets in the flow, number of octets transferred, and in the case of TCP, the subsequent TCP flags are OR'ed onto the flag field in the record. Other fields might also be updated in the various NetFlow versions.

The record of a flow is kept in the router until one of these conditions is met [5]:

- Flow has been idle for a specified time (usually 15 seconds).
- Long lived flow records are removed (normally 30 minutes).
- When the cache becomes full heuristics are used to remove whole groups of flow records.
- A TCP connection reaches its end (the FIN or RST flag is set).

When a flow is retired, it is collected into a NetFlow export UDP datagram and sent to a collection server. Flows are exported at least once a second, or when a datagram is full. The NetFlow export UDP datagrams come in various versions. Version 1 can hold 25 flow records, version 5 can hold 30, and version 7 can hold 28 flow records. Figure B.1 and B.2 in appendix B show the datagram for NetFlow version 5. This is the datagram used by the routers that have collected log data for this thesis.

The paper "NetFlow: Information Loss or Win" by Sommer et al. [3] gives a good evalution of the NetFlow aggregation scheme, and it also discusses it strengths and weaknesses.

Figure 1.1 show a simple example of two flows collected from a user connecting to a web server. Because the flows are unidirectional, a TCP connection will at least consist of two flows. A single TCP connection could easily be split up into more flows; for example it could be exported because it has been idle for a certain amount of time.

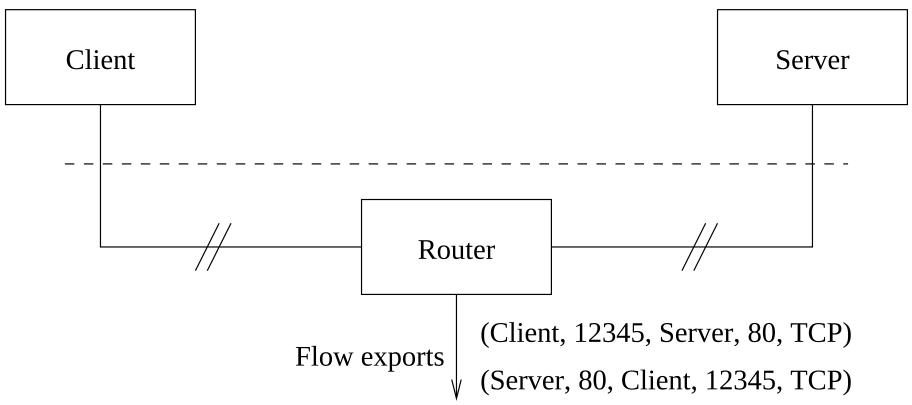

Figure 1.1: An example of flows collected by a router when a client connects to a web server.

Because of the high speeds of very fast routers, it is not possible to log all the packets even if they are aggregated. A solution to this problem is to use sampling. This is done by selecting some random packages from the stream. In Cisco NetFlow an *average* *sampling* interval can be specified. Sampling will of course affect the forensic value of the logging, because now the communication sequences are lost, and some flows are not even detected. We will not investigate this problem closer in the thesis, but it might be an interesting area for further work.

Forensic examination of NetFlow logs

Now we will look at how NetFlow logs can be used in forensic examinations, and how the NetFlow aggregation affects the results.

The aggregation of the traffic into flows reduces the amount of space used for the logs considerably, but it also reduces the information. Important information regarding the forensic investigation can be lost.

The way that the TCP flags are handled is quite unfortunate; when they are OR'ed together over all the packets in the flow. This way one cannot check if the TCP traffic is legal. There could for abnormally many SYN's in a flow, but it cannot be seen from the NetFlow record. On the other hand we know that there is only one FIN and RST per flow. NetFlow does not have any field that have information about error reporting fields in the IP/TCP headers; there is no way to detect checksum errors, fragmentation, TTL field values, and so on. But this is probably a tradeoff between log size, system complexity, and security.

Another drawback to NetFlow is it can vary much, depending on the specific router and its setup. The timeouts can be specified in a wide range and the cache size can also be set, which result in very different flow characteristics. The used parameters are not present in the NetFlow logs, so the operator has to provide this information manually with the logs.

Many fields in a NetFlow record are based on the information in the TCP/IP packets, and therefore they can easily be spoofed. Spoofing is a technique used by attackers, where they create network packets with falsified fields. The duration of packets can also be spoofed to an certain extent, by fragmenting the IP packets.

Even if Netflow has some weaknesses regarding security, it is still useful. It is already implemented in many routers that are on the market, and it would be to expensive to change all of them just to introduce a new router logging scheme. Security tools that support Netflow will therefore be a cheap and efficient way to increase the security as an initial step. In the future more standardized log schemes have to be developed with more focus on security. One such scheme, a draft by the IETF for a router log standard, IPFIX, is based upon NetFlow version 9 [6].

1.3 Postmortem and real-time analysis

Forensic examination of logs today can roughly be split up into two categories: Postmortem and real-time analysis. Postmortem analysis of logs is the investigation of something that already has happened, and which one can not do anything about now. The purpose of this analysis is therefore to find out what has happened. Real-time analysis is an ongoing process, which returns results with a low latency, so that the system or operators can respond to the attacks.

Postmortem analysis can therefore be more exhaustive than real-time analysis. Real-time analysis needs to find the attacks quickly to be effective. Postmortem analysis can be used to examine the attack in more detail and give a more thorough result/report.

Another thing that differs between the analysis methods, is that real-time analysis can only go through the log data once, whereas a postmortem analysis could go through the file many times, and examine interesting flows it had found in previous runs.

The focus of this report is the postmortem analysis, but the real-time analysis is also of interest, because there are many ideas that we can borrow.

The program has to be able to process a day's or more traffic, in a timely manner; the program has actually to process data faster than a real-time system, but with a higher latency.

1.4 Related work

There has not been done much prior work that only investigates forensic examination of router logs. But on the other hand, there have been many papers published in the area of Intrusion Detection Systems, which can be said to be related to examination of router logs. As mentioned earlier, the IDS is a real time analysis of the incidents, whereas the forensic examination is a postmortem. Most of the ideas used in IDS systems can directly or with little modifications be transferred to off-line forensic examination systems.

One of the well known papers is "A Framework for Constructing Features and Models for Intrusion Detection" by Wenke Lee and Salvatore J. Stolfo [7]. Its main contribution is an automatic feature creation scheme, where feature extractors are created by using various datamining techniques. It looks at the IDS from the network layer and all the way up to the application layer. The paper is heavily based upon Wenke Lee's Ph. D. thesis [8]; it is is actually a compressed version of the thesis, and therefore it might seem a little cursory; but even so, it is a popular paper.

An important event for the development of IDS was the 1998 and 1999 DARPA off-line intrusion detection evaluations [9]. It provided researchers with attack and normal traffic, and a common set to evaluate their methods on. Before the DARPA evaluation set was available, most systems were evaluated with a few attacks, and little background traffic. The traffic used was not publicly available either, so the results obtained were not reproducible. The DARPA network models a typical Air Force network. The DARPA test bed was setup with real computer, consisting both of Microsoft and Unix based machines. Traffic was created using synthetic users, that act like normal users; surfing the net, checking mail, file transfers, etc. The network was divided up into two parts: One internal network, and one external (modeling the Internet)[10]. Tcpdump was used to collect and log the traffic seen from the inside of the network and from the outside. Other application level logs where also collected. Planned attacks were performed against selected hosts, and the traffic belonging to the attacks labeled. The DARPA data is freely available on the Internet.

One paper that looks at the use of NetFlow logs for security is the paper "The OSU Flow-Tools Package and Cisco NetFlow logs" by Mark Fullmer and Steve Romig [11]. Flow-tools is a collection of programs to gather Netflow logs, manipulate, and filter them. It supports a wide range of Netflow versions. The file format used by the authors is one that is created for Flow-tools, so it is not widely supported by other software, but there exists an exportation tool, so it is possible to get the log files converted to NetFlow again. Their format supports compression of the log files through use of zlib.

The security tools in Flow-tools are a rules based real-time IDS, activity profiling, and a set of programs for general log file manipulation for incident response. The IDS can be configured to report to rules such as: A source IP containing more than a threshold of destinations within a specified period (port scan). The activity profiler builds a profile of the services running on each host, and can therefore detect new services on the hosts. The Flow-tools are developed on a university network, and therefore the activity profiling feature is of interest to them.

There also exist many non-scientific tools for manipulating and visualize NetFlow logs. As an example we could mention that the Swiss Education & Research Network (SWITCH-CERT) have created a forensic examination tool for examination of Netflow logs. It is called NFSEN, which is an abbrivation for NetFlow Sensor. NFSEN is capable of filtering the Netflow logs, creating Top-N statistics, profiles, and presenting the logs in various graphical plots. As of this writing, SWITCH is not freely available.

Some papers only focus on one feature, and evaluate its efficiency to detect some kind of attack[12][13]. The classifier used in such papers are mostly just based upon a threshold value. The features can often be used both real-time and postmortem analysis. Combining such features extractors, and using a more sophisticated classifier, would potentially yield good results, because these features are quite advanced and yield good results by themselves.

In the paper Detecting Traffic Anomalies at the Source through aggregate analysis of packet header data by Seong Soo Kim et al. [12], a technique for traffic anomaly detection based upon the correlation of outgoing IP addresses of the flows at an egress router. The technique is developed both for postmortem and real-time analysis. In the postmortem analysis the method works with all the log data, but in real-time it works on a window basis (a data set from a short period).

1.5 Summary

We are now ready to pursue the the development of the forensic tool suite. We have now an good idea of what role routers play on the network. We have also seen how the NetFlow log scheme works in detail, where one of the main topics was how NetFlow does aggregation of network traffic.

One might also wonder why there has not been done more research in examining router logs, and logs in general. Log have been around for a long time, and it would seem more logical if it was the IDS systems that borrowed ideas from systems developed to analyze logs.

1.6 Report overview

The report is divided into eight chapters. In the next chapter, chapter two, we will be introduced to malicious traffic, to get a better understanding of the problem, and in this way lead into the third chapter, where we specify the user requirements and an outline for the design of the system. In the fourth chapter we will study how we can obtain sample traffic for the development and testing of the system. In chapter five we will look at how the characteristics of the malicious and normal traffic can be extracted from the log. In the sixth chapter we will briefly study the artificial neural network used for the classifier and the implications associated with it. Now the system is completed, and in the seventh chapter it is tested on various data to estimate its efficiency. Finally in chapter eight the findings in this thesis are summarized and conclusions are drawn about the system developed.

Chapter 2

Malicious Network Activities

In this chapter the attacks that can be found in router logs will be examined. This examination is important for many purposes: The interesting flows are identified in the log file, secondly a better understanding is gained about the attacks as a whole.

Attacks on hosts are often divided into four groups:

- **Probes:** Used to gain information about the host.
- **Denial of Service:** Make a service unavailable to other users.
- **R2L:** Unauthorized access from a remote machine.
- **U2R:** An unprivileged user gains super user rights through a vulnerability.

Probes and DOS attacks can in many cases be seen from the router logs, but it is harder or not possible to discover R2L and U2R attacks, as the attacks occur at a higher OSI layer. It is possible to get indications of that R2L or U2R attacks are going on. For example if a remote machine is trying to guess a password through SSH, one would see several short connections on port 22 on the victim's host. A U2R might be indicated by the victim having new services, which the attacker has started.

It is of interest to the ISP to detect both probes and DOS attacks. Identifying and dealing with DoS attacks improve their service. DOS attacks are more easily handle by the ISP, than by the customers. Probes are also of interest for the ISP, as this gives information of where threats originate from.

We will now look at how probes and DOS attacks work, and how they will appear in the NetFlow based logs. The DOS attacks are examined first, and then the probes. We will not investigate R2L and U2R attacks any further, because they are easilier detected in other kinds of logs. Before looking at the network based attacks, we will briefly look at some of the basics of the Internet protocols.

2.1 A short TCP/IP summary

The protocols that are the foundation of the Internet are part of the Internet protocol suite, and it is often called the TCP/IP protocol suite for short. As some of the protocols in the TCP/IP protocol suite will play a central role, we will briefly examine the ones that are of interest to us. This way we will also get some of the terms explainded.

The most common datagram used in the network layer on the Internet is the Internet Protocol version 4 (IPv4). The Internet Protocol version 6 (IPv6) has been available for some time, it is not nearly as widely used. An IP datagram consists of a header part and a text part. Figure 2.1 shows the structure of a IP version 4 datagram header. The fields that are available as aggregated data in NetFlow version 5 have been written in a bold face font.

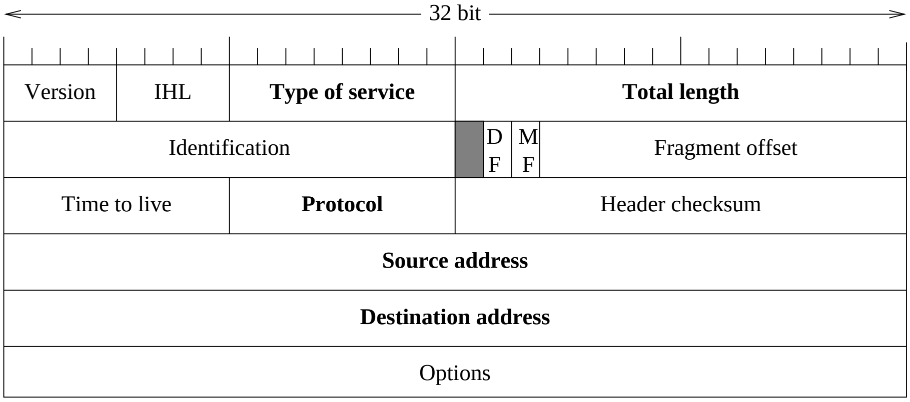

Figure 2.1: The structure of a IP version 4 header[14]. The field written in bold face are present in some form in the NetFlow records.

The IP datagram is constructed by the hosts themselves, which deliver the datagrams to the network layer. Because the datagrams are constructed by the hosts themselves, it is possible for the hosts to put false information into the datagram. Putting false information in the header is called *spoofing*. Some common protocols placed in the text area of the IP datagram, are the UDP, TCP, and RTP protocols.

Protocols are often divided up into connection oriented and connectionless. A connectionless protocol only write an destination in the packet and send it of, there is no checking if it arrives, or if there are several packets, that they arrive in any particular order. An analogy to a connectionless protocol could be the process of sending letters with the postal service. Well known connectionless protocols are IP, UDP, and ICMP.

A connection oriented protocol assures that the packets arrive in the right order, but before this can be done, a connection has to be set up. An analogy to a connection oriented protocol could be a telephone call.

The TCP protocol is connection oriented. To create this service, the TCP header specified by the TCP protocol has some counters and flags, which are used to keep the state of the connection. The flags are provided as aggregated information the NetFlow log and play a central role in many attacks. The flags also play an important role in both network based denial of service attacks and probes.

There are six flags in the TCP header: URG, ACK, PSH, RST, SYN, and FIN. One of the important roles that flags play in the TCP protocol, is to open and close the connections. Figure 2.2 shows how a TCP connection is initiated. The process is called a three-way handshake. The client starts the process by sending a TCP packet to the server with the “SYN” flag set. When the server receives the connection request packet from the client, it will allocate some resources for handling the connection. The server then sends a packet back to the client with the “ACK SYN” flags set, to acknowledge that it received the clients packet and to open a connection back to the client. If the server is unable or unwilling to establish a connection to the client, it will send a packet with the “RST ACK” back to the client, or an “ICMP Port Unreachable” packet. Finally the client sends a packet back to the server with the ACK set, and the TCP connection is established from the clients point of view. When the server receives the ACK packet, it will also regards the TCP connection as established.

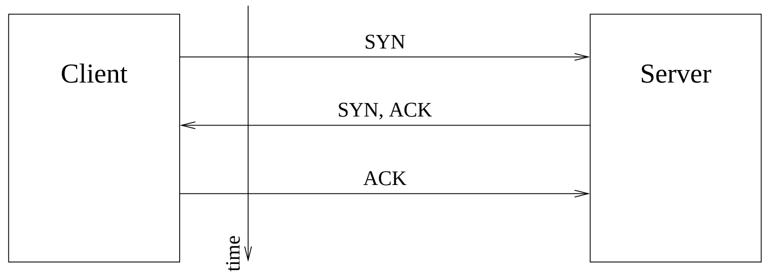

Figure 2.2: The three-way handshake used to establish TCP connection. The arrow illustrate TCP/IP packets.

When the connection is not needed anymore, it has to be closed. This is done using a four-way handshake, where both host send two packets containing ACK and FIN. A normal TCP connection will therefore always contain at least ACK, SYN, and FIN in both directions. Other flags can occur, such as the URG and PSH, but we will not examine them closer here.

Connectionless and connection oriented protocols will have different properties in the logs. With an connection oriented protocol there will always be flows in both direction between two hosts communicating; this does not have to be the case with a connectionless protocol.

This summary of the protocols is very brief, but more information can be found in the Request For Comments (RFCs), which are the technical notes describing the Internet. RFC 793 might be of special interest, as it describes the TCP protocol.

2.2 Denial of Service

A denial of service (DoS) is an attack where a service is made unavailable to the users. This can be done by exhaust the resources of the host running the service, or making it unavailable by exploiting some vulnerability in the system.

The are many types of DoS attacks, which can be organized as shown in the tree in figure 2.3.

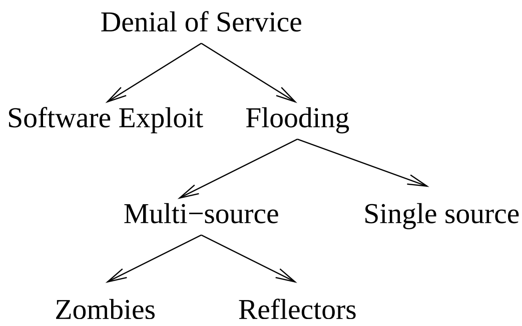

Figure 2.3: Classification of the Denial of Service attacks. [1]

DoS attacks based on software exploits use bugs in the victims software to disable the service. This exploit cannot been seen from the TCP/IP datagram header unless the bug is in the victims network or transport layer. It is more common that the bugs are at higher levels than the network layer. It is hard to detect a DoS attack based on software exploits from the flows, because they do not differ much or not at all from normal flows.

Flooding is another way of launching a DoS attack. The idea is just to swamp the victim with traffic, so that service provided by the victim is useless or degraded. This attack can be used even if there is no known software exploits in the victim's system. Such an attack should be detectable in the NetFlow flows, because it should have different characteristics than normal traffic. For example one could see an sudden increase in the connections to the service.

The flooding attack can be divided into two classes: Single or multi- sourced. The single source attack is the more simple of the two, it just consist of the a sending packets directly to the victim. If the attacker has more resources than the victim, then such an attack could be a success. The attacker can make a single sourced attack look like it comes from many sources, by spoofing the source addresses.

Multi-sourced attacks consist of many machines making a coordinated attack on the victim's machines. A multi-sourced attack is commonly called Distributed Denial of Service (DDoS) attack. To launch such an attack, the attacker must have control over some other machines on the Internet (zombies) or make other machines send packets to the victim in response to spoofed packets (reflectors).

The attacker can get zombies for example by using software exploits to get access to the machines (a R2L attack), or by fooling the user of the machine somehow. In this way an attacker can collect zombies on the net, and use them for coordinated attacks.

Reflectors are machines that respond to IP packets by sending a packet back to the address written in the source field of the IP packet. If the attacker spoofs the packet so, that the source IP is the address of the victim's machine, then all the responses will be sent to the victim. For example machines that respond to ICMP echo request with ICMP echo replies can be used as reflectors[1].

The advantages for the attacker with distributed attacks are that they require less bandwidth and that they are harder to trace back. Disadvantages are that he needs control over more machines.

A sudden rise in the traffic to a host might occur for many other reasons than a DoS attack e.g. many people could try to access the same web page at the same time, a router could fail, or the access patterns could change. If a web page is mentioned in a well-known media, it might look like a DoS attack. For example the interesting web pages often get mentioned on forum called slashdot.org, but this forum is so popular, that after a little while the mentioned web pages are inaccessible.

2.2.1 Examples of DOS attacks

Now we will look at some examples of how DOS attacks can be done. First we look at attacks using the three-way handshake. Afterwards we will look at a DOS attack which uses the UDP protocol.

The seemingly innocent process of establishing a TCP connection can be exploited. The attacker can construct IP packets that look like TCP connection requests. Instead of using the right source IP address (the attackers address) in the packet, the attacker uses some other address (maybe randomly selected). When the server receives the spoofed IP packet, it will think that it is a valid request, and will allocate resources to handle it. This DOS attack is called the SYN flood. If the attacker sends enough packet fast enough, the resources of the server will eventually run out, because it will allocate resources for each request[15]. Figure 2.4 illustrates the SYN flood.

The SYN flood is a good example of a DoS attack that does not try to exhaust the bandwidth of the victim; here it is the implementation of the TCP protocol on some system that is exhausted.

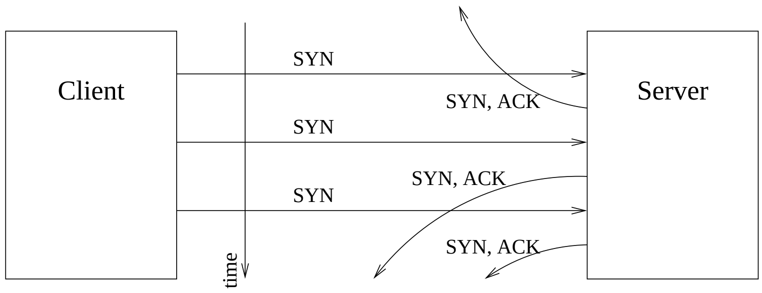

Figure 2.4: The three way handshake used for a reflection attack. The arrows illustrate TCP/IP packets sent on the network.

The three way handshake in the TCP protocol can also be used for a reflection attack as shown in figure 2.5. Now the server is not the victim, but it is used to reflect the attack onto the victim. Instead of forging the IP packet with a randomly selected source address, it could be forget with the victim's IP address. Sending a large amount of packets to many different machines on the Internet will result in many "SYN ACK" or "RST ACK" packets being sent to the victim. If the attack is successful it should exhaust the victims bandwidth. The idea behind using a reflection attack, is that it is harder to trace back and protect against.

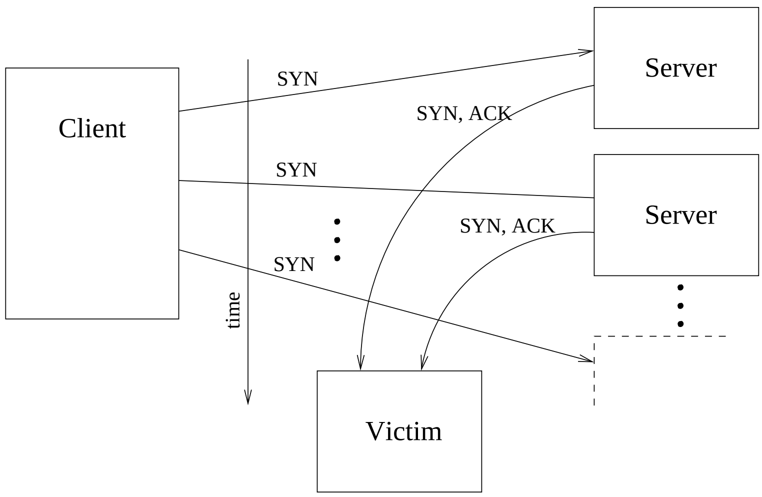

Figure 2.5: The reflection attack on a single victim using multiple servers. The arrows illustrate TCP/IP packets sent on the network.

One of the problems associated with reflection attacks based upon the TCP protocols, is that a good Internet connection is needed: For every SYN packet the attacker sends, the victim needs to send a "RST ACK" packet, a ration 1:1. So the attacker needs a Internet connection that has more bandwidth than the victim. To get around this problem, the attacker needs reflection servers that respond with packet that are considerable larger than the request packets. The TCP protocol is not a candidate, as it requires the initial three way handshake, but with the UDP this is not necessary or left to the higher levels.

Online game servers are one example of servers where the response request ratio is high [16]. Here it is possible to send an UDP packet with commands such as “getstatus” and “getinfo” to the server, and it will reply with a full set of information about the server, map, players, etc. According to [16] the request packets are 56 or 58 bytes, and the response is on average 500 bytes, which corresponds to an average gain of around 9.

Using games servers also makes it very hard to trace the attack back, because the game servers do not keep logs of the requests.

There exists many other DoS attacks, and the three that we have mentioned here, are just a small selection out of a much larger group. But they illustrate the essential features.

2.2.2 Flow characteristics of a DOS attack

Some of the characteristics of a DOS attack can be seen from the router log. Common characteristics are:

- Sudden rise in traffic.
- Traffic pattern of a server changes (e.g. server can not respond to all the request it gets).
- Special flags combination in TCP header

2.3 Probes

The purpose of probes is to discover the available services and vulnerabilities on one or more machines. In real life probes are often used gather information from hazardous or difficult to reach places. On a network an attacker might use probes to gather information about a network before attacking it. The hacker can use the information obtained from a probe to identify the operating system and application running on the machines, and exploit their weaknesses. A probe is often one of the first steps in an attack, which has compromising the machine as a goal. Therefore it is especially interesting to examine this kind of malicious traffic. Discovering probes therefore give indication on what the attacker are interested in, and what hosts are likely to be exposed to further attacks.

On a network a probe is normally called a scan, and they are grouped according the way they are performed:

- **Host scan:** One or more ports are examined on a single host.
- **Port scan:** One or more ports are examined on more than one host.

There are many different definitions in the literature, but we will distinguish how a scan is performed with these two.

If the scanned machine exists, then each examined port can have one of two outcomes: Either the port is closed, or it is open. The flows between the two host will look different for each case. So for each scan type we might expect at least two patterns of traffic. If the scanned machine does not exist, then there will only be the flows from the probing machine.

There are many types of probes, but the most commonly used are based upon the TCP protocol, as it is connection oriented. There also exist connectionless probes, for example using UDP and ICMP. Popular scanning programs are: Nmap, portsweep and ipsweep. The TCP based scans can roughly be divided up into three main groups:

- **Open scanning:** Opens and closes a connection to the target port.
- **Half-open scanning:** Only opens the connection.
- **Stealth scanning:** Detects open ports without opening a connection.

We will look closer at scans from each group.

2.3.1 Open scanning

Open connection scans are done using a full three-way TCP/IP handshake. There are severals types of scans which do this, but we will look at the TCP connect scan.

The TCP connect scan simply does a TCP connect to the ports of interest. This process might result in two or more flows in the Netflow log, depending on the particular service and the cache size, load, and setup of the router. If the probe detects an open port, it will at least receive a "ACK SYN" from the victim, and probably also "PSH" and "FIN", but this is not for certain. It depends on the service running one the victim's port. The probing machine might also send a "RST" to the victim. If the probe on the other hand detects a closed port, the victim replies with an "ACK RST".

2.3.2 Half-open scanning

This scan is closely related to the Open scanning methods, as the name might suggest, this scan does not complete the three-way TCP/IP handshake. We will look at the SYN scan.

The SYN scan is related to the TCP connect scan. The only difference is that the client does not send an "ACK" back to the server when it finds and open port. If a port is closed it still receives "RST SYN" from the server, just like the TCP connect scan. With the SYN scan only two flows are seen in the Netflow log for each port scanned.

2.3.3 Stealth scanning

Stealth scanning is broad term, but it is mostly associated with scans that avoid detection in some way or the other. Avoiding detection is dependent on the current state of technology, so the stealth scans today might not be the stealth scans of tomorrow.

Nmap can do three stealth scans: Stealth FIN, Xmas Tree and Null scan. These scans all send packets with special flag combinations. Stealth FIN sends a packet with the "FIN" flag set, Xmas Tree sets the "URG PSH FIN" flags high, and finally the NULL scan sends a packet with no flags set.

The idea behind these scans are that closed ports are required by RFC 793 to reply to these packets with “ACK RST”. So if a port does not reply, one can assume that it is open. Machines running the Windows OS do not confirm with the RFC 793 at this point, and will not reply to these packets.

The stealth scan are called “stealth” in the Nmap documentation. In the paper by Raj Basu et al. “Detecting Low-Profile Probes and Novel Denial-of-Service Attacks” [17] they define a stealthy probes as: Probes are considered stealthy if they issue ten or fewer connections or packets or they wait longer than 59 seconds between successive network transmissions. This definition of a stealthy probe is probably more realistic in our case, because they illegal flag combination used by the scans Nmap calls stealthy are detected easily, because they do not occur in normal traffic. When the Nmap documentation refers to an probe as stealthy, they mean stealth with regard to the firewall.

2.3.4 Flow characteristics of a scan

The flows characteristics will depend on the kind of port scan used, but there are some common characteristics.

Here we summarize the general characteristics for scans:

- Flows differ for open and closed ports.
- Special flag combination.
- The “ACK RST” flag is often in the response if a port is closed.
- Some flows do not get a response (machine does not exist or a MS machine).
- Flows are small.

Special characteristics for a port scan are:

- Many flows from a single host to many hosts.
- Often the same ports are scanned each host.

Special characteristics for a host scan are:

- Many flows one host to another.
- Many different ports access.

2.4 Summary

In this chapter have defined the various attack types, and found out that the attacks that we can find in a router log are Probes and Denial of service attacks. The detection of the two other types of attacks, R2L and U2R, from the router logs is much more uncertain. We have also had a brief look at TCP/IP protocol, so we can understand what mecanisms the attacks use. Some common characteristics of each attack type has been identified, so that we know what we should look for in the logs.

Chapter 3

User Specifications and Design

In this chapter we will look at the user specifications for the forensic tool suite and its design. Some of the specifications are set by DK•CERT and some are set to the system as a forensic tool. The design will try to fulfill the requirements as much as possible.

In this chapter we will first look at the requirements and then at the design. At the end of the chapter we also briefly discuss an alternative design to the one used in this thesis.

3.1 User Specifications from DK•CERT

Here we will look at the requirements set by DK•CERT . The requirements set by DK•CERT are inspired by problems they have been faced in relation to router logs.

DK•CERT 's requirements to the forensic tool suite are that it should able to:

- **Analyze an incident:** An incident might be reported and the investigator wants to extract the traffic related to that incident.
- **Find compromised machines:** Identify machines that have been compromised by a hacker or malicious software (worms, viruses, etc.).
- **Identify malicious traffic:** Isolate malicious traffic such as probes, denial of service attacks, etc.

Another inevitable requirement is that the tool should be able to handle large amounts of data in a timely manner. The log files can be several gigabytes, and the investigator has a limited time to complete the investigation. Some other requirements to the program, but not directly related to forensics, is that they should fit into the UNIX and Linux environments.

To be able to fulfill these requirements the program must be able to easily manipulate the log files, such as filtering and sorting the entries. For example one could filter a log, so that only traffic related to a certain machine is left. The program should also be able to classify the each entry to normal traffic or to some group of abnormal traffic.

3.2 Forensic requirements for the tools

There are some special requirements, which are set for forensics tools. The results of a forensic tool suite could be used as digital evidence, or result in actions being taken, which could be costly for a company. It is therefore important that the forensic tool suite only helps the investigator analyze and interpret the logs in a correct manner.

In the article Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers by Brian Carrier[18] a set of requirements is given for forensics analysis tools:

Usability: The program should solve the complexity problem, because data at its lowest level is to difficult to interpret. This should be done by presenting the data in a layer of abstraction and format that can help the investigator. The data should be presented in a clear and accurate format, so that the investigator does not misinterpret it.

Comprehensive: Both inculpatory and exculpatory evidence should be identified. The investigator must have access to all output data at a given abstraction level.

Accuracy: The tools should ensure that the output data is accurate and the margin of error should be presented to the investigator, so that it can be interpreted appropriately.

Deterministic: The tool should always produce the same output data, when presented with the same input data.

Verifiable: To ensure the accuracy of the tool, it should be possible to verify the results. This could be done manually or by using another tool.

It is also important that the input data is sanity checked before it is analyzed. If there are something wrong with the input data, then the investigator should be informed of it. For example presenting the forensic tool suite with an unknown file format should make it stop and return an error message to the investigator.

One of the goals in this thesis is to make programs that fulfill these requirements to such an extent as possible.

3.3 Design/Solution

In this section we will look at a design that fulfills the user requirements and the forensic requirements. The practical issues associated with the chosen design are also studied.

Figure 3.1 shows an overview of the design of the forensic tool suite. The main input to the system is the log file itself; the log is feed to system on record basis, so that an entry in the log is feed to the system one at a time.

The first block encountered by the entries is the filter, and it is here that the investigator can do some of the required log manipulation. The next block is the feature extraction, which can shortly be explained as a transformation of each log entry into a set of numbers, which can be used to describe the characteristics of the log entry. The third block is the classifier, and its purpose is to take a set of numbers from the feature extractor and classify it as a certain group of traffic. The last block in the system is the post-processing, here the output from the classifier and the original log entries are joined to give the final result.

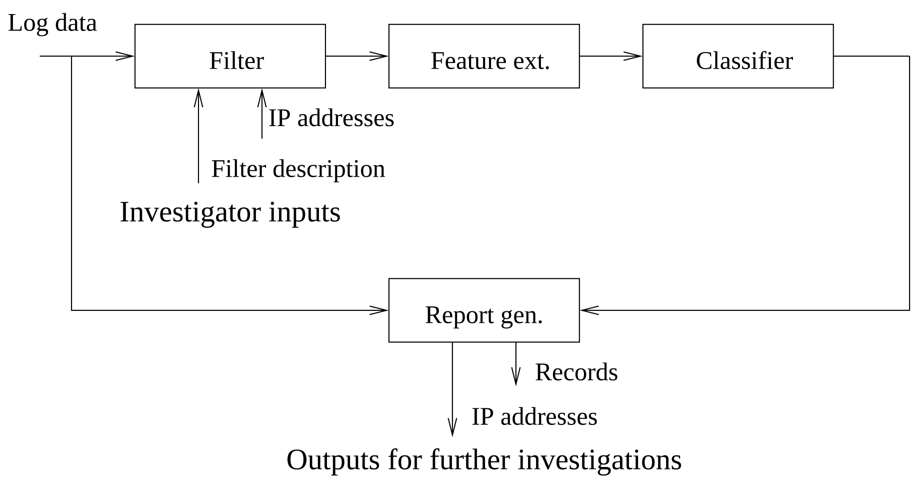

Figure 3.1: The main structure of the forensic tool suite.

The filter block can be used to filter the entries in various ways, such as filter it after start time-stamp, source and destination addresses and ports, and protocol number. The investigator can also provide a list of IP addresses (in a file) instead of writing them all manually. But the forensic tool suite has to be able to manipulate the in others ways too, such as merging and cutting logs. We will look closer at this problem later in this chapter.

The feature extraction process can explained as a process that takes the log entries are transformed into a set of numbers that the classifier can use to classify each entry with. The set of numbers produced by the feature extractor is called the feature values. A single feature values is based upon some feature or characteristic of the entry in the log file.

In this thesis an artificial neural network based classifier is used to recognize the feature value patterns that are associated with the various traffic types. Using a neural network has the advantage, that it can be trained to recognize new types of traffic, without having to modify the system itself.

The output of the classification stage is combined with the original log file, so each entry is assigned to a class. The combined information is post-processed into information easily interpreted by the investigator. The post-processing stage should also be able to extract the IP addresses that are part of an attack, so that they can be used in further investigations. Post-processing could optionally also create an overview report to the operator, with the attack name, category, start time, duration, and involved hosts.

The feature extractor and classification blocks in figure 3.1 could together be called for the attack detector. They are responsible for identifying the malicious traffic in the logs. The attack detector is special for our forensic tool suite, and this thesis will focus mostly on it.

In the following sections we will look closer at the design of the filter, feature extractor, and classifier.

3.4 Log file manipulation

One of the more basic operations that the tool suite should be able to do, is to manipulate the log data. For example one could be interested in only looking at the traffic to one specific machine for a given time.

The basic operation the tool suite should be able to do are:

- Select a period out of a larger log file, given by a start and a stop time.
- Filter after fields given in the flows.
- Merge logs together.

These are basic operations, and many more could be added. But these where sufficient to solve the problems encountered in this project.

We will now briefly look at how the filtering function is added to the tool suite. A filter will be specified as a disjunction of sentences, where a sentence is a conjunction of conditions, see equation 3.1.

$$filter = \vee \wedge C_{ij} \tag{3.1}$$

Where C_{ij} are conditions. Example conditions could be that the destination or source IP address should have a certain value. So the filter specifies what should be let through. Because the filter is so simple, it might be very cumbersome to do some expressions. But it can be helped a little by adding an additional filter, just like the one given by equation 3.1, but which just removes entries if true.

We could take an example where one or more hosts have been found to be port scanned. Now it is interesting to see the other traffic that these machines have been involved. One way to do this would be to give the programs a filter parameter, which specify that the source or destination address should be a specified value. An example:

```
logconverter -f ip=80.121.12.12;ip=80.121.12.13 portscanlog.nf
```

Here the ";" is used for a disjunction, and "," is used for conjunctions. If there are more than a few IP addresses involved, this method of writing a filter might become too time consuming and error prone. Instead is should be possible for the investigator to extract the addresses to a file, and supply the filename to the program instead.

```
logconverter -f ip=ipaddressfile.txt,port=80 portscanlog.nf
```

Here we filter after traffic to and from port 80 in a list of IP addresses supplied by the file "ipaddressfile.txt".

3.5 Feature extraction

In this section we will look at how the features extractors will extract a set of feature values for each entry in the log. We will also look at the data structures used to create the feature extractors, and the complexity that they introduce to the system.

Assigning a log entry a set of feature values can be done by looking at the fields in entry, and by looking at the entry's relation to the other entries in the log.

Related entries in the log file can be assumed to be temporally close, and also spatially close if the log is ordered after time. It therefore seems a good idea to use a time based window to select the entries that the features are extracted from.

This principle is also used in IDS (Intrusion Detection System), for example in Lee et al. [19] they use a 2 second long window. IDS work in real-time, and therefore they

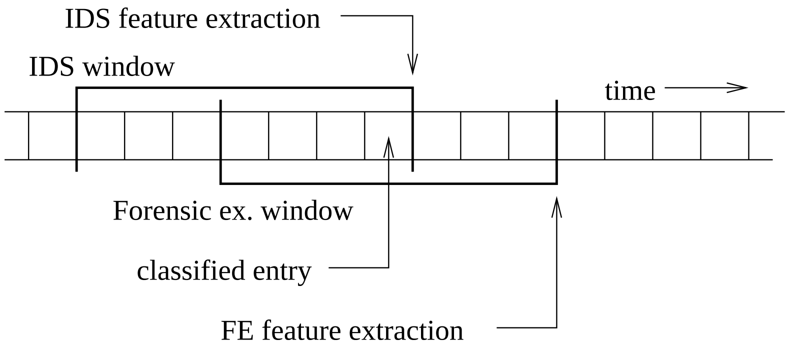

Figure 3.2: This figure illustrates the window often used in IDS systems, and the window we will use in our forensic examination tool.

use a window that looks at the entries in the past, and make decisions based upon that. The advantage of doing this is low latency.

In a forensic examination we do not only have to base our classification on the past, but we can look into the future (by going forward in the log file). For example it would be hard to classify a probe based on only the first flow related to it; but if we look further in the log file before classifying it, it will become much easier. Figure 3.2 shows both the IDS window and the window used for the forensic examination.

When working with Netflow logs the duration of the flow has to be taken into consideration. The flow should be used for the feature extractions as long as some part of it is inside the window.

The feature values for each flow are assigned to them when the are in the middle of the window; but the flows have a duration, so they be in the middle of the window for a period of time. A logical approach would be to assign the values then the middle of the flow is in the middle of the window. But this might add unnecessary complexity to the program, as the flows will be assigned values in a different order than they entered the window. If the window is large compared to the duration of the flows, it will not matter that much when the feature values are assigned on the flow. This is probably the case for probing flow, as they are quite short, and this is the approach that we have used. The same argument can also be used for removing flow from the window.

The logs needs to be sorted after the start time-stamp, so that a all the relevant log entries are inside the window when a entry is assigned a set of values. This can be done by having a sorting front end, in front of the feature extractors, that sorts the data. The sorting can be done again by having a sliding window. The length of the sorting window should be the same time it takes a flow to expire [1] from the router. The flows inside the sorting window are sorted by inserting them into a red-black tree which is ordered after the start time-stamp. Almost the same approach is taken by Fullmer et al. [11], where they use a sliding window with heap sort.

One of the requirements, was that the forensic tool suite should handle large log files. Using the sliding window approach will help towards this goal, as the file is analyzed from the start to the end in one go; this way utilizing the hard-drive and OS cache optimally.

As we saw in the previous chapter, a NetFlow entry can be uniquely identified on

[1] The window is actually a little larger than the time it takes a flow to expire to allow for uncertainties in the system.

the router by the tuple in eq. 1.1. But when a flow is exported, the tuple becomes insufficient to identify the flow, because on router we only look at flows that are active. When a flows is exported it belongs to a certain period in time. So an additional field is needed in eq. 1.1 to identify exported flows. There are two choices: The start or end time stamp of the flow. We will use the start time stamp, as it is more practical for our application. So in the application a flow is uniquely identified by equation 3.2.

$$flow = (start_{stamp}, src_{addr}, src_{port}, dst_{addr}, dst_{port}, ip_{prot}, src_{if}, tos) \tag{3.2}$$

This tuple in impractically large to be used as a unique identifier (key) in most data structures. So in the application only a subset of the fields will be used, for example the time stamp, destination address and port. This will reduce the memory consumption and make lookups faster, but at the expense of precision

The data structure that has been chosen for handling the NetFlow records is a Red-Black tree. It has been chosen because it has very predictable behavior and running time. The complexity of the basic operations like deleting and inserting is $O(lgn)$, where n is the number of elements in the tree, which in our case is the number of entries inside the sliding window. By using a red-black tree it is very easy to look up flows based their key, and also to look up the corresponding reverse flow.

Ideally the log analyzer must be able to handle all kinds of log files. It should not crash or change alarm rates, because an attacker has crafted a special log file, which the analyzer cannot handle. But we have not investigated how this could be handle further.

3.6 Classifier

Now that we have the framework for the feature extractors, it is time to look at the classifier, the last step in the attack detector.

There will be a classifier for each type of traffic, and the classifier will give a grade to each flow, which says how likely it belongs to the traffic it is trained to detect, or some other traffic. The higher the mark assigned to an entry, the likelier the classifier finds that the entry belongs to a specific kind of traffic. If the mark is low, then it is likely that it does not belong to that group.

One of the important features of the system is that it classifies each entry in the log. This way the investigator can see exactly what entries triggered the alarm. It is also the discovered log entries themselves that are the digital evidence. Another possible way of classifying the traffic would be just to give an report that said that one host had sent some malicious traffic to another. But then forensic tool suite would not fulfill the forensic requirement that it should be comprehensive, because the step between assigning a suspicion value to each entry and generating the report would not be accessible to the investigator. A report cannot be considered evidence, it is just information to help the investigator.

3.7 Fulfilled specifications and requirements

We will now briefly look at how this design fulfill the specifications set by DK•CERT and the requirements set to forensic tools.

With this system most of the requirements set in [18] are satisfied to a certain extent. The *usability* requirement is satisfied, because the task of identifying the entries in the log files are done by the classifier, and post-processing can be used to give a easily interpreted output. The *comprehensive* requirement is not wholly fulfilled, because the system is not able to find exculpatory evidence, but the at least the investigator has access to the lower levels of the system, so that a manual investigation can be carried out. The inculpatory evidence is the records that have been classified as an attack by the classifier; the report from the post-processing is not evidence in it self, just a summary of the evidence. It is hard to assure *Accuracy* in this system, as we do not know the transfer function within the classifier. Testing can be used to get a qualitative measure of the performance. The system is *deterministic* as it always gives the same result to the same input. And the system is *verifiable* too, because it is possible to go through the log files manually and verify the classification of each entry.

3.8 An alternative feature extractor design

We will now briefly look at another alternative design for the attack detector, which could be another approach to the problem or a complement to the solution already developed.

The design that we have selected is based upon a sliding time based window, but there are alternatives. Another approach be to store a constant number of entries for each pair of hosts, which could be considered as a "constant number of entries" window. A pair of hosts creating many flows would soon fill the window, and pairs that create a few flows would not even fill the window. The flow in the middle of the window would be classified when the window was full, based upon features extracted from the window and other windows. This design would be good for slow probes, as slow probes would be collected into a separate window. The disadvantage of this design would be that the entries in the log file would be classified out of order, and the resulting classification has to be reordered to match the original log file. Another disadvantage is that the memory usage is not so predictable, as with the sliding time window. If there are many addresses in the log, then the memory consumption would be high, because a window has to be allocated for each pair of hosts. The memory consumption could even be so high that the log analyzing machine would run of it, and thus failing to analyze the log. An attacker could easily create many IP addresses in the log file by sending spoofed packets through the router.

It was originally the plan that we would also implement this design, but differed to much from the design based on the sliding time window, so it was dropped. But it is an interesting design, which has other properties than the one used in this thesis.

3.9 Summary

We have now seen the specifications set by DK•CERT , and the requirements set to forensic tools. The design given in this section should be able to fulfill most of the requirements. The design has given us the framework for further development of the forensic tool suite. In this thesis we will especially look at the attack detector, which consists of the feature extractor and the classifier; both will be studied in detail in chapter 5 and 6. We have seen a alternative design to the feature extractor, which is an part of the attack detector. It would be interesting to investigate this design further, but time did not allow it.

Chapter 4

Data Acquisition

Obtaining realistic data is an important part of the development of a attack detector. It will be used for constructing the feature extractors, training the classifier, and to test the whole system. Without realistic traffic, the attack detector would not work when used in a real network environment. The whole system's performance is dependent on obtaining good data.

Another problem that we face with router logs, is that the methods used later in this thesis, require the entries in the log to be labeled, which means that the there has to be some accompanying information for each entry in the log, saying if it is an attack or not.

In this chapter we will look at how realistic router data can be obtained, and how it can be labeled, so that it can be used in the development of the attack detector.

4.1 Labeled traffic

The methods that we will use in this thesis, require the obtained logs to be labeled. Labeled traffic is traffic that has been assigned to certain groups, for example normal, Denial of Service or probe traffic; groups that we want the classifier to distinguish. Labeling the traffic is not an easy task, especially when working with the large amounts of data present in router logs.

A real logs is not labeled, so it is not known which kind of traffic the flows belongs to. The log could be labeled by using an already existing attack detector, or by doing a manual labeling. Both approaches have their drawbacks. It is not such a good idea to use an already existing attack detector, because these are mostly rule based. The final attack detector's performance will also be worse or equal to the one used for the labeling. The manual labeling is time consuming and error prone, but it is not based up on simple rules, but on the operator's understanding of the attacks. On the other hand the manual labeling will result in an biased result, which could give the attack detector better performance than it actually has. For example the operator might not want to include flow with only the FIN flag set, because he thinks that it belongs to the Stealth FIN scan, but for some reason might occur in ambient traffic. Software and hardware do not always conform to standards, and therefore normal traffic could include traffic which does not conform to the standards either.

We will divide the traffic up into two groups: Attack and ambient traffic. Ambient traffic is all the traffic that is not the actual attack traffic. Ambient traffic could be understood as an analogy to ambient sound, it is a background signal, that mixes with the signal that we want to detect.

The traffic labeled as attack traffic will be the flows created by the attacker or attackers. The response traffic from the network will not be labeled as attack traffic. For example if we have a port scan, then all the port scan traffic should be labeled as attacks. All other traffic should be labeled as ambient traffic, also the Denial of Service traffic, because it is not the attack in question.

4.2 Ambient traffic

For this thesis ambient traffic is obtained from two sources: Ambient traffic from a real routers log, and ambient traffic derived from network traffic provided by DARPA (Defense Advanced Research Projects Agency). We will now look at why these are used, and what implication they might introduce.

Realistic data can be obtained from routers in the real world. But there are privacy concerns with this approach. There are laws governing this area, and the policy of the ISP could also prohibit this. It is therefore very unlikely to find such traffic publicly available. Because of the low accessibility to router logs, it is very fortunate that DK•CERT have provided us with NetFlow based router logs obtained from their routers. The traffic is collected from a router called Lyngby 2 over several days using the OSU flow-tools[11]. But after the data collection we discovered, that this router was being faced out. So the traffic on the router was reduced over the days. Even so we think that the data collected on the first days is not to much affected by this [1]. It was not possible to collect from other routers, because DK•CERT sent these to a special logging program for storage.

Figure 4.1 shows some statistics derived from the data from the Lyngby 2 router. There are a few well known ports in the top ranking list, but also some high valued unknown port. Many different protocols are also present.

The approach to handle the data obtained from DK•CERT in this thesis is to manually detect the attacks, and remove them, so that the final result is an ambient only log. The manually selected log entries can be seen in table 4.2 and 4.3.

But there exists some traffic publicly available on the Internet, such as anonymized traffic provided by NLANR [20], and synthesized traffic from the DARPA set [9]. The traffic provided by NLANR (National Laboratory for Applied Network Research) is mainly intended for research, and its main goal is to give a good representation of the Internet traffic workload. The traffic is collected from high-performance IP networks and anonymized. The traffic is not labeled, and therefore it will impose the same problems as the traffic from DK•CERT 's router.

Another source of network traffic is the DARPA Intrusion Detection Evaluation set from 1999. We have already looked briefly at the data provided by DARPA in the introduction in page 6. We can use the attack free traffic from the DARPA set as ambient traffic for our system. The DARPA set does not contain any router logs, it does however contain the dumped[2] traffic from inside and outside of the network. The outside traffic could be aggregated to Netflow data, and in this way be used for our purpose. But there are some problems associated with this approach. A router between the inside and the outside would only log packets that it forwarded between

[1] The first day log data was collected was 27th October, 2004.

[2] Tcpdump was used to obtain the dumps.

Rank	*Src. port*	*Kb trf.*	*Dst. port*	*Kb trf.*	*Prot*	*flows*
01	00080	32989950	00119	326091220	006	7889324 (75.38%)
02	38843	31373742	00025	6750362	017	2274680 (21.73%)
03	38844	31083491	00000	6165324	001	245497 (2.35%)
04	38845	30987677	01080	4169471	050	19631 (0.19%)
05	38842	30751179	00080	3876497	002	17380 (0.17%)
06	38846	29959382	01500	2990496	103	12813 (0.12%)
07	59341	26963522	00020	1807811	047	5440 (0.05%)
08	59338	26774186	00022	1213458	089	1574 (0.02%)
09	59340	26678702	06916	764751		
10	59339	26596015	63625	742443		
11	59337	26034787	52748	738501		
12	00119	7063945	00443	705390		
13	40959	6278846	09875	672608		
14	00000	6243637	01127	635006		
15	04500	4260051	40260	589928		

Table 4.1: Top statistics for a whole day derived from the Lyngby 2 log the 28/10-04. The log file consists of 10.466.339 flows, and the total amount of routed data is 409.957 MB. DK•CERT has given permission to publicize these data.

Rank	*Src. port*	*Kb trf.*	*Dst. port*	*Kb trf.*	*Prot*	*flows*
01	00119	194751	56970	67974	006	37456 (60.67%)
02	00080	148800	56816	63853	017	21321 (34.54%)
03	00110	15328	57132	62309	001	1728 (2.80%)
04	00020	9433	26030	47560	002	1232 (2.00%)
05	65531	9071	00080	24613		
06	01392	8429	15239	6393		
07	34791	8337	00119	5841		
08	32912	7578	23645	3395		
09	01025	7572	29804	2644		
10	36616	6573	40230	2216		

Table 4.2: Top statistics for manually selected entries from the Lyngby 2 router the 28/10-04, from 6:00 to 9:00. Selected entries consist of 61.529 flows and 460 MB of routed traffic.

the two sides of the network, and we do not know the ingress or egress filter rules on the router. Depending on the setup, the dumped traffic might also include traffic between the machines on the outside of the network. Another problem is that Cisco's NetFlow is a proprietary format, and the source for the aggregation scheme is not publicly available. It is therefore not possible to aggregate in precisely the same way a Cisco router would do. The advantage of using the DARPA set is that attack free data is obtained, which was not the case for the data from DK•CERT .

To get the aggregated logs from the DARPA data, *softflowd* [21] and OSU flow-tools[11] were used. Softflowd is able to take a tcpdump file and aggregate it to NetFlow data, which is sent to a NetFlow collector. Softflowd was used to aggregate the packet headers to NetFlow datagrams, which where captured by flow-tools and stored to a file. Softflowd is licensed under the BSD license, and it freely available on the net. The timings used in the program were set so, that they should resemble the standard settings used on a Cisco router. By using this program, the error mentioned earlier is introduced, which is that it is not possible to check how well this aggregated data

Rank	*Src. port*	*Kb trf.*	*Dst. port*	*Kb trf.*	*Prot*	*flows*
01	00119	249662	18811	89339	006	42401 (66.56%)
02	00080	118514	18968	80956	017	19665 (30.87%)
03	00443	84169	19129	78447	001	1499 (2.35%)
04	00020	11691	26030	47575	002	142 (0.22%)
05	65531	9071	00080	18367		
06	01392	8430	00025	13853		
07	34791	8337	00119	8031		
08	01025	7587	34700	5162		
09	32912	7576	00443	5046		
10	36616	6574	39803	2811		

Table 4.3: Top statistics for manually selected entries from the Lyngby 2 router the 30/10-04, from 20:00 to 23:00. Selected entries form 63.707 flows and 561 MB of routed data.

correspond to the aggregated data produced by a real Cisco router. The setup used for softflowd was a timeout of 15 seconds, a max lifetime of 30 minutes, and a cache size with 8192 entries. These are common values.

Table 4.4 shows some statistics derived from the aggregated DARPA traffic. The data is from a whole day, which is Monday from the third week of the DARPA set. It is interesting to note that the ports that send most data, are well known ports and typical for Unix based systems. Another interesting observation is that the TCP protocol (protocol number 6) is totally dominating; it accounts for 99% percent of the flows. The UDP (protocol 17) and ICMP (protocol 1) represent a marginal part of the flows.

Rank	*Src. port*	*Kb trf.*	*Dst. port*	*Kb trf.*	*Prot*	*flows*
01	00080	179459	25126	26030	006	76266 (99.09%)
02	00023	30217	00023	20644	017	549 (0.71%)
03	00020	16464	00080	17388	001	150 (0.19%)
04	00022	2105	00025	6432		
05	00025	1662	06549	4797		
06	12408	860	00022	2617		
07	10929	857	01141	2227		
08	21588	856	03652	2216		
09	21028	850	21343	2175		
10	05560	725	18940	2169		

Table 4.4: Some statistics derived from the aggregation of the outside traffic from the DARPA evaluation set. The data is from a whole day, which is Monday in the third week. The log file consists of 76.965 flows, and the total amount of routed data is 272 MBytes.

Comparing the statistics from the real router log and the one derived from the DARPA set, one can see that they look quite different. There are not as many protocols present in the DARPA set, and the ratio between the number of TCP and UDP flows is very different between the two. In the real log about 21.73% of the traffic is UDP, but only 0.71% in the DARPA set. These differences could indicate the DARPA set is not as versatile as real router traffic.

By having traffic from two sources, DK•CERT and DARPA, the results obtained are based on a better foundation. This is important, because both sets have their respective strengths and weaknesses, as we have seen. For example ambient traffic the DARPA set could include flag combination, which the operator would not include, because he thought the flow were attacks.

4.3 Attack traffic

Obtaining good attack traffic is as important as obtaining good ambient traffic. The attack detector's development will be based upon the difference between the characteristics of those two sets. In this section we will look closer at how we can obtain labeled attack traffic. Obtaining attack traffic is not necessary as complex as obtaining attack free traffic, but it is depends on the attack type.

A simple way of obtaining the traffic associated with a probe, is to let a machine probe another machine, so that the traffic goes through a specific router. The router logs can then be filtered after traffic between the two hosts. The traffic from the probing machine to the victim machine is then labeled as a probe. It should be noted that it is important that there is no other communication going on between the two machines, other than the probe.

An attack type which is hard to generate is a Distributed Denial of Service attack. Both because it requires many hosts to participate, and it could potentially disturb the network. In such an case it is better to simulate the attack using a model, but it can also have its drawbacks.

The traffic, labeled as a certain attack type, can be merged with an attack free log to obtain a router log that can be used for developing an attack detector for that certain type of attack.

The original plan was to obtain the flows associated with some denial of service attacks, but this was not possible, as it would disrupt too much, and would therefore require a separate isolated network. Instead some probes were selected. A small network was located behind a router which was not being logged, and permission was obtained to probe the network. The probes done where host scanning and port scanning. They were done using Nmap. The scans were conducted in polite and normal speed. The scans were conducted twice, to detect any variation, which did not seem to be present. These ports were scanned: 21, 22, 25, 53, 17, 113, 105, 33, 129, 29, 1, 13, and 93. A total of 13 ports. The scans used were Nmap's TCP connect, SYN scan, Stealth FIN, and Xmas Tree. The scans resulted in total of 104 flows labeled as probes from the scanning machine to the victim. When scanning the ports politely it took around 5 seconds to scan all the ports, at normal speed it took about 0.2 seconds. The duration between the scan types is around 30 to 60 seconds (the duration when no scanning occurs). Only port 21 turned out to be open.

The scans should ideally be done on a wide range of machines and operating systems, and also hitting more open port, so that it would be harder to detect. But this was not possible, as we had to select a network which would be willing to participate, and which was on Forskningsnettet.

4.4 An alternative way to collect labeled traffic

In this section we will briefly look at an alternative way off collecting labeled traffic. It requires good control of a large network, and access to the router logs, but it should have potential.

A way to obtain attack free traffic, would be to have a list of “good” hosts. All traffic between a pair of “good” hosts would be included into the attack free traffic collection, the idea being that “good” host will not send malicious traffic. In the list of “good” hosts there should be a good representation of all hosts on the network, e.g. user machines, web servers, routers, and streaming media. The amount of traffic is not so important, but it should include most types of traffic. “Good” hosts will have to be both on the local networks and the Internet, so careful ingress and egress filtering must be done, so an attacker can not pollute the attack free traffic, by sending packets with a spoofed source address. Removing outliers for each address pair, should improve the quality of the data. Outliers are values that are not consistent with most observations. For example a computer used by students on an university might occasionally be used for malicious activities, and the traffic created will not be consistent with normal traffic. It is therefore important to collect mostly normal data, so that the abnormal can be detected easily and removed.

The attack traffic still have to be generated manually or isolated in some other way.

4.5 Summary

We have looked at the importance of obtaining realistic logs for the development of the system, and its performance in a real environment. We have also seen that it can be difficult to label the data. Different methods exist to do the labeling of the data, but the manual labeling has been selected for this project, as it can be done within the time frame of the project, and it could yield potentially good results. Logs are obtained from two source: A real log and from the DARPA evaluation set. It will be interesting to see the differences between the two set in the development of the attack detector. We have already seen that they differ considerably in top statistics.

Chapter 5

Feature Extractors

The main goal of this chapter is to develop some feature extractors based on the traffic obtained in the previous chapter. Good feature extractors give values that can be used to tell the normal and attack traffic apart. The attack traffic obtained in the previous chapter were some host scans, so the feature extractors are developed to detect this kind of attack traffic. But the methods used in this chapter are not limited to this type of attack. The problems solved are general problems, associated with feature extraction from router logs. So if attack traffic for a Denial of Service attack was obtained, then the same methods could be used to generate some feature extractors for this attack. The feature extractors developed in this chapter could be considered as a proof of concept.

Feature ranking will be used to estimate the rank of the features, and to select the most important features. The values of the features will be modified with value reduction, so that they can be used efficiently with a classifier.

The chapter is organized as follows: First we look at feature ranking and value reduction, as they are the basic tools used to develop the feature extractors in the rest of the chapter. Then the intrinsic features are examined, which are features that can obtained directly by reading the fields of the entries. Finally some traffic features are examined, which are features that look at the relation between the flows in the log. The traffic features are mostly modified versions of the features used in the paper by Lee et al.[19].

5.1 Feature ranking

Feature ranking is a very important part of the development and evaluation of the feature extractors. Here the worth of each feature is measured, and it can greatly improve generalization and the speed of the classification, and also give a better understanding of the problem at hand.

Feature ranking methods can roughly be divided into two groups: Feature ranking using wrappers and feature ranking using heuristics.

Wrapper based feature ranking use the classifier itself to evaluate the usefulness of a feature. The process consist of training the classifier with all possible combination of feature inputs, and evaluating the output for each combination. No need to say that the wrapper approach is very time consuming, and it does not scale well with the number of features, or the size of the training set.

Heuristics try to evaluate the usefulness of each feature without training the classifier, but based on the characteristics of training data.

Duch et al. have written a paper [22] on a comparison of heuristic feature ranking methods. Duch et al. compare a few entropy based methods and a chi-sqaure based statistical method, and their conclusion is that none of the methods that they have tested emerges as a winner, but they found out that the classifications results were significantly influenced by the ranking index. So basically any feature rankinging method from their paper can be selected.

One of the problems with the heuristic methods presented in Duch et al., is that they do not take into account the interaction between the features. The heuristic methods can not discover that using two particular features together yield good result, and the heuristic methods cannot discover if two features are correlated, and together they do not provide more information that one of them.

We will use the entropy based ranking algorithm with the normalization given in equation 5.1, as Duch et al. found it to give good classification accuracy with up to 16 features for a certain experiment based on real data [22, page 3]. If more than 16 features are used there is a classification accuracy drop of about 5 percentage points from 96 percent, but the accuracy does also depend on the number of samples used.

$$U_H(C, f) = \frac{MI(C, f)}{H(f, C)} \tag{5.1}$$

C is a set of classes, and f is a set of feature values. The output range of $U_H(C, f)$ is $[0; 1]$. If $U_H(C, f)$ is 1 then the class can be determined wholly by f, if $U_H(C, f)$ is 0 then it does not say anything about the class. $U_H(C, f)$, $MI(C, f)$ and $H(f)$ are defined according to Shannon [23] as:

$$H(C) = -\sum_{i=1}^{K} p(C_i) \lg_2 p(C_i) \tag{5.2}$$

$$H(f) = -\sum_{x} p(f = x) \lg_2 p(f = x) \tag{5.3}$$

$$H(C, f) = -\sum_{i=1}^{K} \sum_{x} p(C_i, f = x) \lg_2 p(C_i, f = x) \tag{5.4}$$

$$MI(C, f) = -H(C, f) + H(C) + H(f) \tag{5.5}$$

Where K is the number of classes, which in our case is 2; one for ambient traffic and one for the specific attack.

It is important to understand how the entropy ranking works both for using it and its implementation. The entropy method looks at how a class in C depends on a symbol in f; it does not care about the value of the symbol, only its probability and joint probability with symbols in C. For example we could have a feature that was an odd number each time an entry belonged to the ambient class, and even number each time it belonged to the attack class. It could rank potentially high, but it would depend on how many symbols are present.

The number of features that this ranking method can be used on is depends on the number of samples used. If too few samples are used, then the deviation of the ranking value becomes to high, and the ranking becomes incorrect.

5.2 Value reduction

Many learning based algorithms require that the range of the inputs is bounded within a certain range. This is also the case for the neural network based classifier that we will be using. We will develop the feature extractors to use a output range $[0; 100]$. This range can then be scaled into the range used by the classifier.

The feature extractors do not necessarily produce values inside this range, so their output values have to be transformed, and it is here that value reduction comes in. Value reduction looks at the values of the features, and modifies them in various ways, so that the final feature value lies within the given range. Value reduction should be done with caution, because valuable information can be lost. But if the discarded information is irrelevant, then there is also something to gained, e.g. faster training and better usage of the range.

There are two kinds of fields in a NetFlow record: Fields that describe a quantity, and fields that describe a set of symbols. Fields that describe a quantity are e.q. duration, transferred octets, and packet count. Fields that describe a set of symbols, which do not have any value relation to each other, are for example the TCP flags, port number, and the TOS field. Presenting such a feature as a value to the classifier would introduce to many discontinuities. These two kind of field cannot be value reduced in the same way.

The range of the fields that describe a quantity are very large. For example the transfered octets feature could easily be several MB; but in the case of a host scan, the more interesting part of the scale is the lower part, because the flows are small. So here the is a possibility to do some value reduction.

A very simple approach would be to cut of a part of the scale, by using equation 5.6.

$$x_{vr}(x) = \begin{cases} \text{round}\left(100 \cdot \left(1 - \frac{x}{x_{cutoff}}\right)\right) & \text{if } x \leq x_{cutoff} \\ 0 & \text{if } x > x_{cutoff} \end{cases} \tag{5.6}$$

In equation 5.6, the x is the original feature value, $x_{vr}(x)$ is the value reduced feature value, and x_{cutoff} is the cutoff value, which has be selected individually for each feature. Eq. 5.6 cuts of the upper range of the feature output value, and scales the rest of the range to lie in between 0 and 100. We will most be interested in doing this for the scans, as the interesting part will be in the lower range of the fields. We also round the value afterwards the range has be reduced. The reason for doing this, is that this is an easy way of creating a fixed amount of bins (in this case 101 bins). Having a fixed number of bins will make the entropy values assigned to each feature more comparable.

5.3 Intrinsic features

In this section the intrinsic features of a flow will be examined, but we will first look at how flows that have been split can be grouped back together, and at how the reverse flow of a flow can be found effectively. Then the intrinsic features of the forward and reverse flows are ranked and value reduced.

The intrinsic features are the easiest to obtain; they are just read directly from the log's entries. Figure 5.1 shows the intrinsic features that we have selected for further investigation.

Feature name	*Description*
duration	The duration of the flow.
protocol	The protocol specified in the IP header.
port	TCP port number.
octets	The amount of octets transferred.
packets	Number of packets in flow.
flags	The TCP flags.
tos	Type of Service field from IP header.

Table 5.1: Intrinsic features extracted directly from the Netflow entries.

Fields that have not been selected are the IP addresses, ingress and egress interfaces, next hop router's IP, autonomous system numbers, and source and destination address prefix mask bits. Because these presumably will not be useful to detect scans, or at least we would have to traffic from more routers, to see a trend. .

The *flags* field is only valid when the protocol type is TCP, but the *flags* feature has to be assigned a value in any case. Assigning zero value to the flags, is not good idea, because it is highly suspicious combination. Instead a very innocent combination will be used: "ACK PSH SYN FIN", which will occur in most normal TCP connections.

The intrinsic features should be read directly from the entries, but we have modified the intrinsic features a little to accommodate for the little subtlety in NetFlow, that flows can be split into smaller flows, if they expire for some reason. This problem is handled by having a data structure where flows are identified by $(src_{addr}, src_{port}, dst_{addr}, dst_{port})$. This data structure is handled as follows:

- When a flow is first seen, which means that it has entered the sliding window and it does not have an entry in the data structure, then an entry will be made for it.
- If a flow is already in the data structure, the fields associated with the flow are updated.
- When a flow leaves the the window, and there are no other flows in the window with the same identification, then the entry for the flow is removed from the data structure (and the associated fields are freed).

The fields associated with each entry in the data structure are:

- Duration
- Transferred octets
- Packet count
- Accumulative logical OR of flags

The associated fields are updated by accumulating the values seen in the flows passing the window. This is the way that NetFlow would do it, and now the split flows are effectively grouped together.

Normally the window size should not affect the intrinsic features, but when we group the flows inside the window together, the window size will affect the result. A too small window, and the flows will not be grouped together, and a too large window might result in flows being grouped together, which should not be.

The intrinsic features of the flows in to opposite direction is also of interest, because these are related to the flows. If we have a forward flow identified by eq. 5.7:

$$(srcaddr, srcport, dstaddr, dstport) \tag{5.7}$$

Then the intrinsic features of the reverse flow flow eq. 5.8 are related to the forward flow:

$$(dstaddr, dstport, srcaddr, srcport) \tag{5.8}$$

Finding the reverse flow is just a simple lookup in the data structure that groups forward flows together; and the grouping of split reverse flows is done for free. If a reverse flow does not exist, then the features have to be assigned some values anyway. The transferred octets are set to 20, packet count to zero, duration to zero, the protocol to zero, and the flags all not set. These values have been selected, because they indicate that the flow is a probe, which a missing reverse flow also does.

Before investigating the intrinsic features of the obtained log data, let us look at what one might expect. The *transOctets* and *duration* of a flow could be an indicator for probing attack; both are likely to have small values for flows associated with probing. One should note that these feature could be also be small with flows not associated with probing. The *protocol* is also an important feature for probing attacks, as they normally keep to one protocol type. The *port* important as some ports might be more interesting than others. We will look closer at how some port might be more interesting that others later. Error indicating combinations in the flags, *flag*, is also an interesting feature for probing attacks, as some probing methods generate flag errors.

Figure 5.1 shows the ranking values given to the intrinsic features using the manually selected traffic from DK•CERT 's router. There is quite some variation between the two manually selected sets, especially with the flags.

Figure 5.2 shows the ranking of the intrinsic features when using the host scans and some attack free traffic from the DARPA evaluation set. The used background traffic is from the Mondays of week 1 and 3. There is a negligible difference in the ranking values between the two days. The ranking value given to the flags in both directions is several times higher than the value given to the other features.

Comparing the feature ranking from the two figures, 5.1 and 5.2, one can see that the flags are very important in both directions. The protocol feature ranks very badly with the DARPA set, but did quite well with the real traffic. The reason is the DARPA set mostly TCP traffic, and therefore the protocol is almost always the same value, giving the protocol a low entropy value. In the real traffic it does give information, because we only look for TCP scans in this case, and the non-TCP protocols represent a considerable amount of flows.

The features duration, transferred octets, and packets will be correlated. But we cannot see how much. One would expect the transferred octets to tell more about the scans, because its has value changes even if there is one packet in the flow. If there is only one packet, then the duration is 0 seconds.

The initial feature ranking is not final ranking of the features, and we will not know the true rank of a feature until we have done the value reduction.

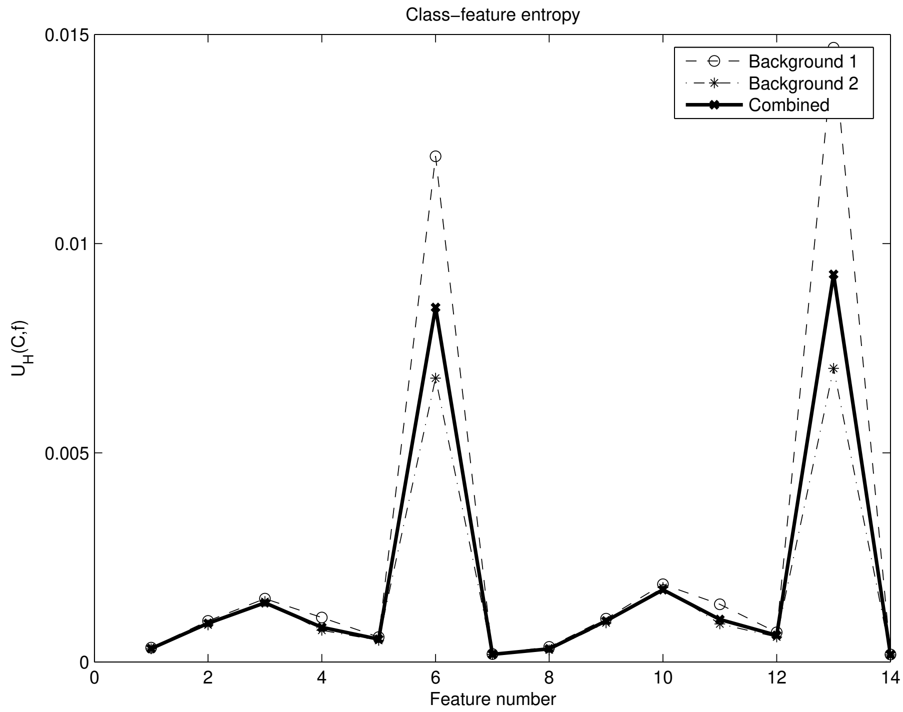

Figure 5.1: Entropy ranking of unmodified intrinsic features using manually selected background traffic from real log data. The probe used is the host scan, and the window size is 120 seconds. The numbers on the x-axis represent 1:duration, 2:protocol, 3:port, 4:octets, 5:packets, 6:flags, 7:tos, and **reverse:** 8:duration, 9:protocol, 10:port, 11:octets, 12:packets, 13:flags, 14:tos.

5.4 Value reduction of the intrinsic features

Now we will look at how the intrinsic features can be modified to suit our needs using value reduction. We will first look at the features that describe a quantity and then afterwards the features that describe a set of symbols. We therefor start by examining the duration, packet count, and transferred octets of a flow, which are the features that describe a quantity. Then we examine the port number, protocol, and flag features, which are the features that describe a set of symbols. The TOS will not be examined, as it ranked very low in both the data sets.

5.4.1 Duration

The duration of a flow scored badly in both sets. But it might be improved a little by collecting the uninteresting data into a smaller part of the range.

Figure A.1 show the entropy value for different cutoffs. We can see that if we should base our classifier on this feature alone, then we should set the cutoff value as low as possible. One should remember that the feature ranking only looks at the relation between the C and the feature in question. It does not take the other features into account. With only one feature, we would only have one dimension, but if we include two feature, the classifier would have a two dimensional space to base its decision on.

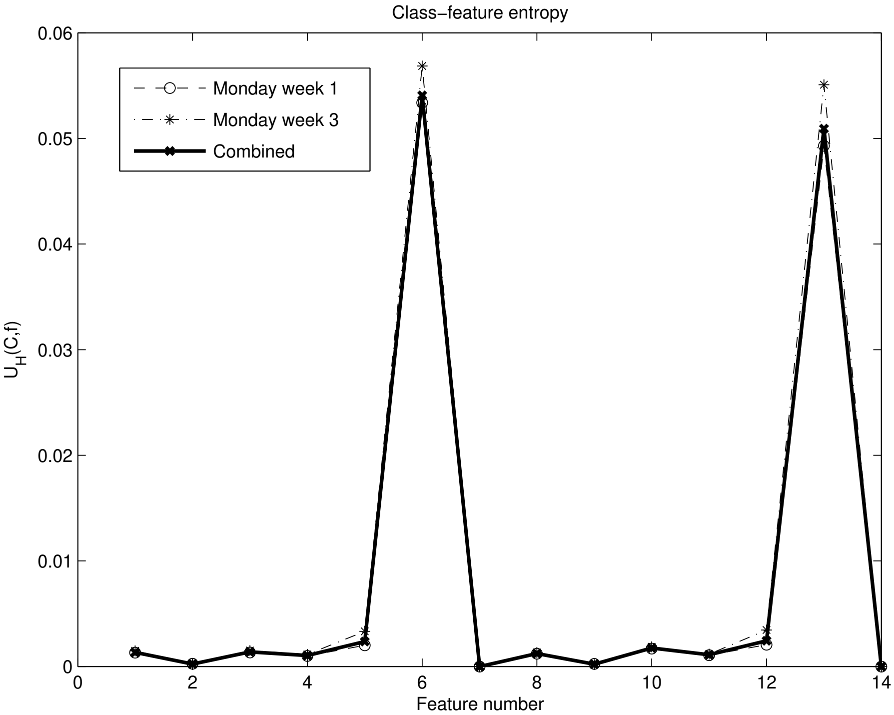

Figure 5.2: Entropy ranking of unmodified intrinsic features using attack free data from the DARPA evalution set. The probe used is the host scan. The numbers on the x-axis represent 1:duration, 2:protocol, 3:port, 4:octets, 5:packets, 6:flags, 7:tos, and **reverse:** 8:duration, 9:protocol, 10:port, 11:octets, 12:packets, 13:flags, 14:tos.

As an initial choice for the cutoff value $x_{cutoff} = 10ms$ is selected. This choice is likely to close to zero, but we will examine the problem of choosing at too small value in the testing section.

5.4.2 Packet count

The packet count and duration of the flow are probably correlated to a certain degree.

The packet count has been reduced in the same way as the duration, and resulted in a cutoff value of 4 packets, and the same for the reverse flow.

Figure A.2 shows the entropy ranking value of the packet number versus the cutoff value.

5.4.3 Transferred octets

The number of transferred octets can vary much; it can range from 20 octets (only the IP header), to several thousands of octets.

Intuitively we can say that scans are small flows, so it is the lower part of the range that is interesting. So we can use equation 5.6 again. Figure A.3 shows the entropy ranking value of the transferred octets versus the cutoff value.

The large the flow sizes get, the more uninteresting they become to the classifier, and therefore it could be a good idea to compress the upper region the range. As the flows sizes get large, the distance between the entries also grows with regard to the size itself.

This relation ship can be found by taking a large sample of normal network traffic, and sort the entries with regard to the flow size; the flows are then divided equally between a number of bins, so that the smallest flows are put into the first, then the next bin, and so on. By looking at the border values between the bins, it is possible to see how distance between the flows grows with the size of the flows. Figure 5.3 a). shows the border values for a whole day of traffic. The flat area is due the fact that there are many flows in this sample that have the size of 96 octets.

The growth can not exactly be said to be logarithmic, but it indicates that we can compress the upper region of the scale. We will do this by taking the logarithmic function of the transferred octets.

To make the transferred octets make use of the whole range [0; 100] we also subtract the header size of the IP packet of 20 bytes, and an additional 20 bytes if it is a TCP packet. The header sizes are subtracted before the logarithm is taken.

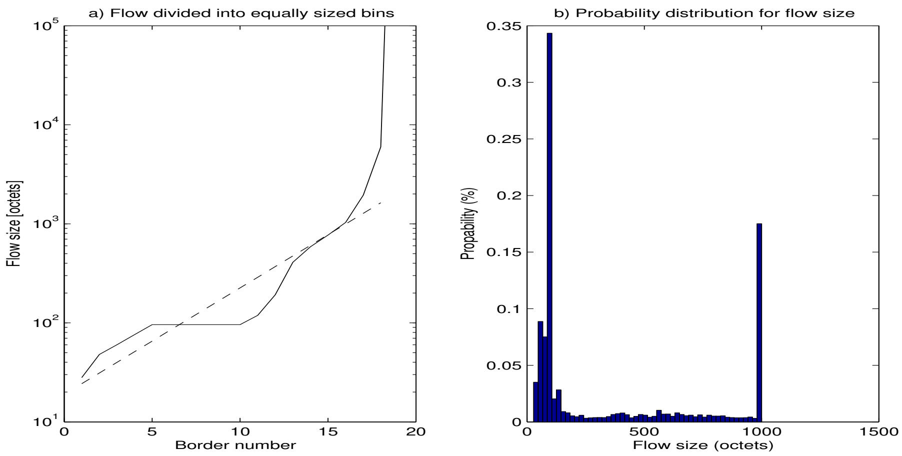

Figure 5.3: *a)* Flows are sorted after size and divided equally between the bins. The flow size at the border between the bins are shown on the vertical axis, and the bin numbers on the horizontal axis. *b)* The probability distribution of the flow sizes up to 1000 octets. These graphs are generated from a whole days traffic sampled for each 100'th entry.

5.4.4 Protocol number

As we have mentioned earlier, it is not such an good idea to present the protocol number to a classifier that is based upon a neural network, because the field does not represent a value, but that the flow belongs to a certain group. For example the TCP protocol (IP protocol number 6) will probably be used in many scans, but how will this affect the classifiers perception of TCP neighbor protocols, protocol 5 and 7? To avoid this problem, we just extract the essential information from the protocol number, which could the connection orientation for the particular protocol. For example it connectionless protocols could be given a feature value of 0 and connection oriented could be given the value 100. The idea behind this being, that it is usual to have flows in both direction for connection oriented protocol, but this is not always the case for connectionless protocols.

Another possibility could be to introduce a feature for each symbol, which is the protocol in this case. The feature could be 100 if the flows belongs to the specific protocol, or 0 if it does not belong to it. There are many protocols, so one has to find out which to include. A guess could be to include the TCP, UDP, and ICMP protocols, which should cover most network based probes.

A third possibility would be to just filter all non TCP traffic away, as we are only looking for TCP based scans. This would not affect the intrinsic features, but information could be lost in the traffic features. For example if a hacker is doing a combined UDP and TCP scan, important information is lost in the traffic relation between the UDP and TCP flows.

5.4.5 Port number

It is probably not such a good idea to feed the port number directly to the neural network, because then it will just be trained to those specific ports. Therefore it is better to use a general feature of the port number. One possibility is to assign the well known port ranges a number, see table 5.2.

Feature value	*TCP Port range*
0	Well known ports (0 to 1023)
50	The registered ports (1024 to 49151)
100	The dynamic and private ports (49152 to 65535)

Table 5.2: A simple feature based on the port number.

Another possibility would be to obtain a top ranking list of destination ports used in scans. And assign a feature value based on the position of the port in the top ranking list. The problem with this method is that the ranking list would have to be updated, and that new port scans, for example a new virus, that is not on the list, would not be assigned a correct feature value. But such a list would on the other hand make the detector better, but our main purpose is to make a general scan detector, so we will leave it out for now.

5.4.6 Flags

Feeding the flags directly to the classifier will not give any good result, as it has $2^6 = 64$ discrete values for each direction, with interesting values distributed over the whole range. So the flags have to be transformed too.

We can transform the flag value, so that it only has a high output when it is likely that the flow is part of a probe, or one could say that the flag combination is suspicious. This transformation will result in a much more specific (not general) feature extractor. With a specific feature extractor the training will faster, and a much smaller neural network is needed. We will study this closer in the testing chapter.

It is easy to find the suspicious flag combinations for the TCP SYN, Stealth FIN, Xmas Tree and Null scans, because here the flags are well defined in both directions. The flags for the TCP connect scan are not so clearly defined, as it stays open until the server closes the connection. For example if you connect to a ssh server, then it normally sends a version information and a password prompt. So the only flags that we can be sure about are the SYN from the client and the SYN ACK from the server. The connection could be closed with FIN or RST.

Based on the flags, it is not possible to make a feature that distinguishes TCP connect scans from normal traffic, if it scans an open port; if it on the other hand scan closed port, it will behave like a TCP SYN scan. The feature extractor given in table 5.3 will therefore also work for closed port scanned by TCP connect scan.

So we can have one feature extractor that goes high when it is very likely that the flow is part of a scan (for the TCP SYN, Stealth FIN, Xmas Tree and Null scans), see figure 5.3. The flags are given for the forward and the reverse flows. The flags forward flags are given before the "/" and the flags of the reverse flow afterwards. Flags that are not stated are ignored.

We will call the feature extractor given by table 5.3 for *suspiciousflags*.

Probe type	*Port status*	*Flags*
TCP SYN scan	Closed	$\neg$A $\neg$P $\neg$R S $\neg$F / A $\neg$P R $\neg$S $\neg$F
	Open	$\neg$A $\neg$P R S $\neg$F / A $\neg$P $\neg$R S $\neg$F
Stealth FIN	Closed	$\neg$A $\neg$P $\neg$R $\neg$S F / A $\neg$P R $\neg$S $\neg$F
	Open	$\neg$U $\neg$A $\neg$P $\neg$R $\neg$S F / (no response)
Xmas Tree	Closed	U $\neg$A P $\neg$R $\neg$S F / A $\neg$P R $\neg$S $\neg$F
	Open	U $\neg$A $\neg$P $\neg$R $\neg$S F / (no response)
Null scan	Closed	$\neg$U $\neg$A $\neg$P $\neg$R $\neg$S $\neg$F / A $\neg$P R $\neg$S $\neg$F
	Open	$\neg$U $\neg$A $\neg$P $\neg$R $\neg$S $\neg$F / (no response)

Table 5.3: Flag combinations for forward and backwards flows, which indicate that it is likely that the forward flow is a probe. This table does not hold for machines that run a Windows based OS, because it will not respond to Stealth FIN, Xmas Tree, or Null Scan. So for a Windows machine we should not expect any response for those.

We call the feature that implements table 5.3 for *suspiciousflags*. Another possible feature based upon the flags could be one that was high each time a flag combination occurred that was not allowed by the RCF 793. But that would require us to go through all the combinations.

The flags could also be split up into a feature for each flag, so instead of having a feature, we would have six, one for each flag. We will also look at this option in the testing section, and compare it to the *suspiciousflags* feature.

5.4.7 Summary of intrinsic feature

The interesting intrinsic feature have now been examined and values reduced. The new entropy ranking is shown in figure 5.4 and 5.5. A logarithmic scale has been used, as the *suspiciousflags* ranked at least a factor 10 higher than the others.

The combined flags are with no doubt the most important intrinsic feature. But we should keep in mind that the *suspiciousflags* is a combination of two intrinsic features that already ranked high.

The transferred octets is now ranking high in both the manually selected log and DARPA log. Intuitively the transferred octets should also rank higher than the packet count and the duration. One reason for the transferred octets ranked badly originally is probably due to the large amount of different values present in the log, which unmodified would result in many bins.

Two features that did not improve were the protocol and port number, but this was expected.

We have seen that cutting off most of the higher range of intrinsic features values gives high entropy values, but this does not necessarily mean it will yield good results with the classifier. The classifier will look at several features at one, and not just one at the time. Cutting too close to zero, could mean that the classifier looses information. The selected cutoff values are 50 octets, 10 ms, 4 packets.

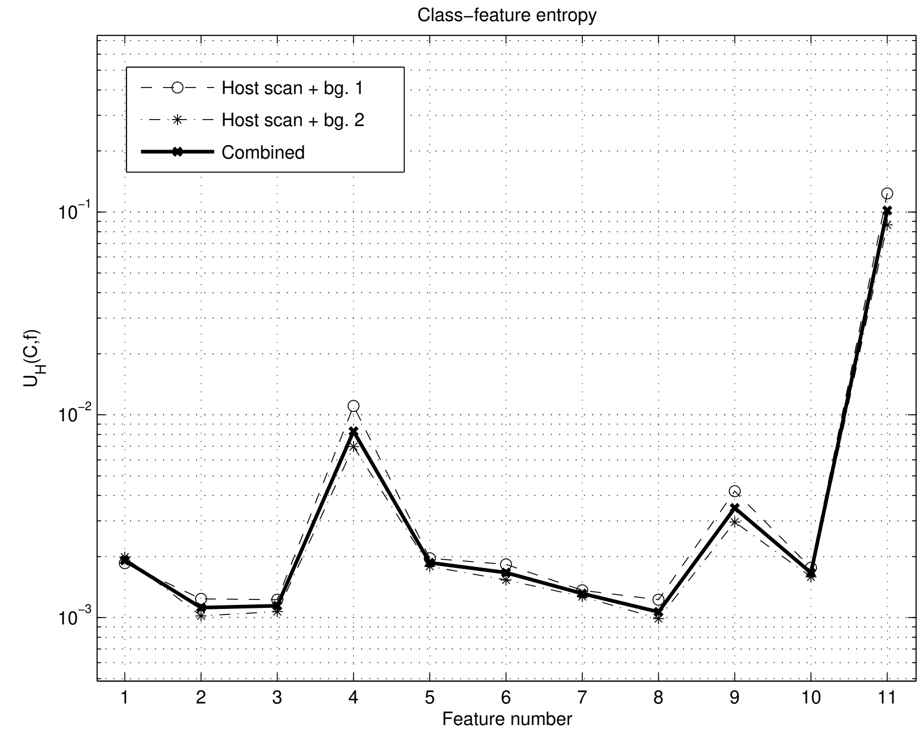

Figure 5.4: Entropy ranking of value reduced features using manually selected data. Numbers on x-axis correspond to 1:duration, 2:protocol, 3:port, 4:octets, 5:packets, and **reverse:** 6:duration, 7:protocol, 8:port, 9:octets, and at 10: suspiciousflags.

5.5 Traffic based features for a host scan

Now we will look at the traffic based features extractors. The traffic features look at a flow's relation to the other flows. Because traffic feature look at the relation between the flows, it should give some additional information to the classifier, which the intrinsic features cannot supply.

Lee et al. create some feature extractors for network traffic in their IDS system [19] [8]. These feature cannot be directly used for Netflow logs, as IDS system normally have

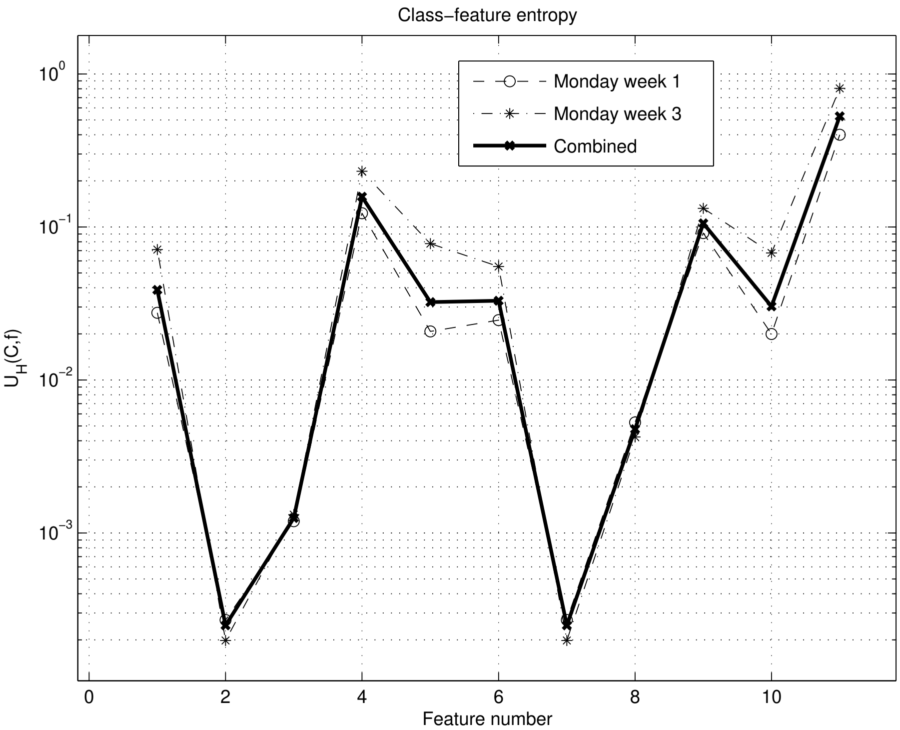

Figure 5.5: Entropy ranking of value reduced features using DARPA data. Numbers on x-axis correspond to 1:duration, 2:protocol, 3:port, 4:octets, 5:packets, and **reverse:** 6:duration, 7:protocol, 8:port, 9:octets, and at 10: suspiciousflags.

access to all the net traffic, but where Netflow only gives access to aggregated information about the traffic. For example the features extracted for each TCP connection cannot be directly transferred to Netflow, because we cannot see the intermediate steps in the traffic associated with a connection. We can however assume that the traffic features of the flows are closely related to the traffic features of a connection. The main difference between a connection and a flow, is that the flow is unidirectional, and can be split up into smaller parts. A scan would create traffic, where many new connections were made to a host inside a short time frame; in the same way many there would be many new flows to the scanned host inside the same time frame.

Table 5.4 shows the traffic features. As with the intrinsic features, we will look at flows in both direction when assigning a set of feature values to a flow, this will to a certain extent make up for the unidirectionality of the flows.

One possibility was to use the methods devised by Lee et al. to create a new set of features, but it is questionable how much we would gain from it. The features which Lee et al. methods can create are limited to the algorithm described in their paper.

The implementation of the features in table 5.4 can be done easily again with the data structure used previously, the red-black tree. For the *thb_count*, *thb_rst_count*, and *thb_rst_rate* we need a data structure with entries identified only by (dst_{addr}). For *thb_same_srv_rate* and *thb_diff_srv_rate* the tuple (dst_{addr}, dst_{port}) is needed. And in the same manner for the port based features, we find that two data structures are needed, with entries identified by (dst_{port}) and (dst_{addr}, dst_{port}). Each time a new flow enters the sliding window, the flow will be added to the data structure. When looking

Feature name	*Description*
The following features are for the same destination addr.:	
thb_count	Number of flows.
thb_rst_count	Count of flows having "RST" flag set.
thb_rst_rate	Percentage of flows having "RST" flag set.
thb_same_srv_rate	Percentage of flows using the same destination port.
thb_diff_srv_rate	Percentage of flows using a different destination port.
The following features are all for the same destination port:	
tpb_count	The number of flow using the same destination port.
tpb_rst_rate	Percentage of flows to the same destination port having "RST" flag set.
tpb_same_host_rate	Percentage of flows to the same destination port, but on different host.

Table 5.4: Traffic based features extracted from the Netflow log based upon a time window or a number of flows window. All the features are to the destination host, except the last one.

up the reverse flow up in the data structures, it can be found by changing the dst_{addr} with src_{addr}, and dst_{port} with src_{port}.

One of the differences between the features given in table 5.4 and the ones used by Lee et al. is that the *thb_rst_rate* and *tpb_rst_rate* now only look at if the "RST" flags set, and not at if the state of the TCP connection enters a state where the connection is rejected. Another difference is that *thb_rst_count* is introduced. The reason for this is the *thb_rst_rate* has the weakness that it becomes 100% if there is only one flow to a host which contains a "RST", which is indistinguishable from a host which receives several. It is much likelier that the the host receiving many "RST" is part of a scan, than the host receiving one, but this cannot be seen from the *thb_rst_rate*, but it can be seen from the *thb_rst_count*. As the *thb_rst_count* and *thb_rst_rate* are two representation of the same characteristic, it is evident that they will be correlated.

The size of the sliding window will affect the values of the traffic features, and the size of the window will also affect a feature's ability in distinguishing different types of traffic. For example if we have a large window, say 600 seconds long, then it will not be as effective at finding probes which only take 1 second to execute, because the rest of the window will only add noise. On the other hand the window could also be too small, so that there will not be a sufficient number of probe flows inside the window, to give them their distinct traffic feature values.

When the window is twice as long as the time it takes to execute a probe, all the flows will be inside the window when assigning the feature values to each flow. The last part of probe will just have entered the window when the first part has reached the middle, where it will be assigned a feature value. It is the same story when the last part of the probe has reached the middle of the window, because then the first part of the probe has not left the window yet. This way all the flows in the probe will be assigned a feature value which is based on the whole probe. So nothing is gained by making the window larger than twice the time it takes to execute the probe. This however does not mean that probes longer than the half the window size will not be detected; for a probe to be detected there should just be a sufficient number of flows inside the window. A sufficient number is not easily defined, because it will depend on the ambient traffic and the probe type. The important point is that if we know that we only want to detect a probe up to a certain duration, we do not gain any additional information by making the window length more than twice the duration of the probe.

Figure A and A show the distribution for the reverse *thb_rst_count* using manually selected traffic and DARPA traffic. The feature seem to utilize the range good enough, so we will not modify it.

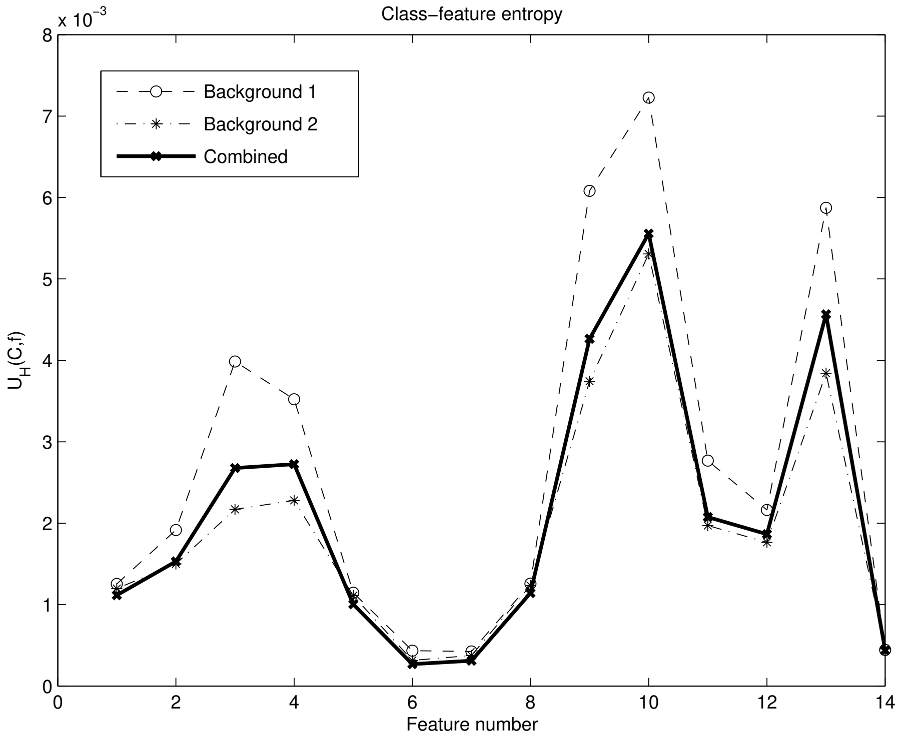

Figure 5.6: Entropy ranking of unmodified features using attack free data from the real router log, and merged with a host scan. The numbers on the x-axis represent 1:thb_count, 2:thb_rst_count, 3:thb_rst_rate, 4:thb_same_srv_rate, 5: tpb_srv_count, 6: tpb_rst_rate, 7: tpb_same_host_rate, and **reverse:** 8:thb_count, 9:thb_rst_count, 10:thb_rst_rate, 11:thb_same_srv_rate, 12: tpb_srv_count, 13: tpb_rst_rate, 14: tpb_same_host_rate

Figure 5.6 shows the feature ranking for the manually selected ambient traffic. The three highest ranking features are feature number 9, 10, and 13, which are respectively *thb_rst_rate*, *thb_rst_count*, and *tpb_rst_rate*. It is not surprising that *thb_rst_rate* and *thb_rst_count* both rank high, as they are highly correlated. The *thb_rst_rate* is probably also correlated to *thb_rst_rate* to some degree. The difference in the entropy value is not as large as we have seen with the value reduced intrinsic features. But there are some traffic features that do particularly badly: The forward *tpb_rst_rate*, *tpb_same_host_rate*, and the reverse *tpb_same_host_rate*.

Figure 5.7 shows the result of the ranking of the traffic features using ambient traffic given by the DARPA set.There are some feature that do considerable better than the others: 2, 3, 9, 10, and 13. It is actually all the features that look at the "RST" flag except feature number 6 the *tpb_rst_rate*.

Comparing the results from figure 5.6 and 5.7 we can see that the entropy ranking

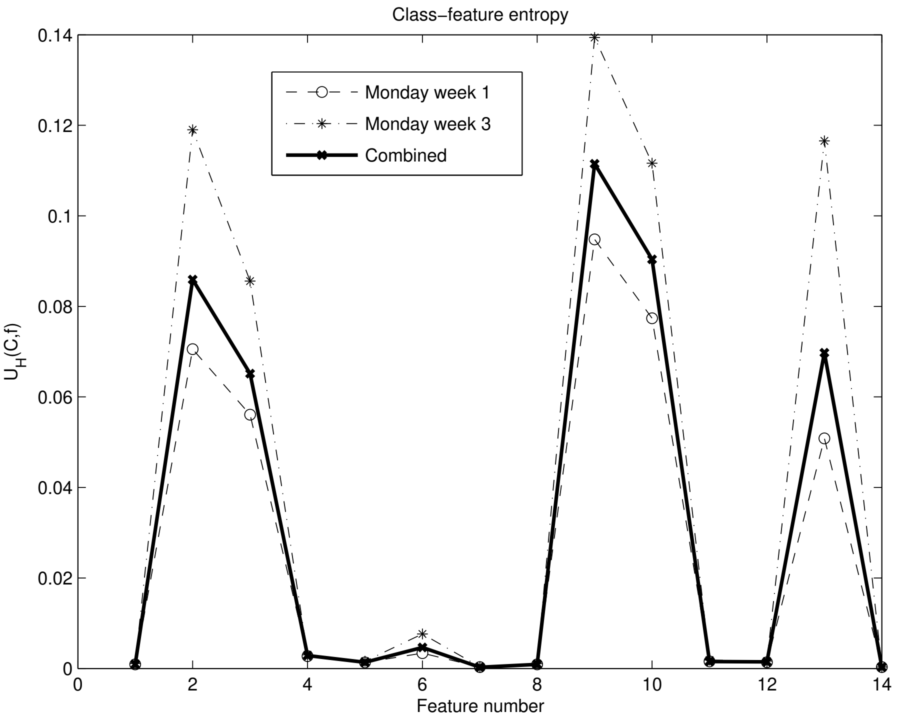

Figure 5.7: Entropy ranking of unmodified features using attack free data from the DARPA, and merged with a host scan. The numbers on the x-axis represent 1: thb_count, 2: thb_rst_count, 3: thb_rst_rate, 4: thb_same_srv_rate, 5: tpb_count, 6: tpb_rst_rate, 7: tpb_same_host_rate, and **reverse:** 8: thb_count, 9: thb_rst_count, 10: thb_rst_rate, 11: thb_same_srv_rate, 12: tpb_count, 13: tpb_rst_rate, 14: tpb_same_host_rate.

values given to the DARPA set are a factor 10 higher than the values given to the manually selected traffic. This indicates that it is much easier for the traffic features to distinguish between the ambient traffic and the host scan traffic in the DARPA based log.

The rankings based on the manually selected log and the DARPA log agree on that feature 9, 10, and 13 are important, but the ranking using the DARPA log wants to give a considerable higher rank to feature 2 and 3, than the ranking using the manually selected log.

5.6 Alternative Features

There are many others features that can be used, some of them will be mentioned here, but they have not been implemented.

Initial contact: A feature that looks at what machine initialized the connection or communication. This could be done by looking at the first packet flow between two endpoints. This feature was not implemented, because some irregularities were found with the start time stamp of the obtained scan traffic. The victim's answer arrived at the router before the probe was sent. The same irregularity was also present when looking at the flows with the OSU flow-tools. The irregularities only existed in the logs from the router used to collect the scanning traffic.

A hot indicator: A feature that increases if a host has already been involved in probing before. I is probably more likely that a machine that already has been involved in probing, that it will do it again (like an analogy to criminals).

Temporal features: A feature that looks at the durations between flows, e.g. a scan often has a constant duration between scan flows that are sent out.

5.7 Summary

We now have developed and examined several feature extractors that can be used to detect host scans. It will be interesting to see how the feature extractors will do in the testing chapter; especially how the intrinsic and traffic features will support or reinforce each other.

We have seen how the features which are related to the flags, except for a few, have a tendency to rank highly. This is the same both for the manually selected traffic, and the traffic derived from the DARPA evaluation set.

For the lower ranking features, there is not that much similarity between the manually selected real traffic and the DARPA set. There are probably several reasons for this: The DARPA set models normal traffic from an air force network. The DARPA traffic is not real router traffic, but has been aggregated by software. The lower features are less correlated with the attack classification, so we need more sample to see the connection.

But we can conclude from the feature ranking that there is a difference between the manually selected traffic and the traffic derived from the DARPA set. This also corresponds well to the top statistics given in chapter 4. The feature ranking values also indicate that it is easier to distinguish ambient and attack traffic in the DARPA set.

Chapter 6

Classification

Classification of log entries is the process of assigning each entry to one or more groups/categories of traffic; the classification is done by recognizing patterns in the set of features values assigned to an entry. This chapter gives a brief description of the classifier used for the testing in the next chapter. We will also briefly look at why we have selected the back-propagation neural network for the classifier.

6.1 Classifier types

There a several ways that the classification can be done, e.g. it can be done with Markov models, neural networks, or genetic algorithms, just to name a few.

A classifier based upon a learning machines normally has two phases:

- The learning phase where the unknown dependencies are found for the system for a given set of samples.
- The prediction phase where the output values are estimated from given input values.

Markov models are normally trained with normal data, which in our case is log files without any attacks. We have seen that it can be a difficult task to obtain such files. Classifiers based on the Markov model will find the entries that are abnormal (anomaly detection). So they will not only find attacks, but also report entries that are changes in trend of the traffic. So if there is a trend change in the traffic, then the Markov model has to be retrained.

With neural networks, NN, it is possible to train it to detect a specific attack traffic type - we could have a NN that detects DoS attacks, another that detects port scans, and so on. If network trends change, it does not necessarily mean that the NN has to be retrained.

The classic NN is the one based on back-propagation learning. It is quite slow at learning, but is fast at making decisions. The slow learning does not concern us much, because this is process that is done offline, when then investigator has isolated an attack. A properly trained back-propagation NN can give reasonable answers to inputs it has not seen before, or it is said to be good at generalizing. With this property the NN can be trained with an representative set of inputs and outputs, which is smaller than the set needed by classifiers, which are not good a generalizing.

The size of a neural network will affect its ability to generalize. If a neural network is large compared to the problem, it can create more complex functions, and can therefore overfit a simple problem. If it is too small compared to the problem, it will not have the power to fit the data. So the size of the back-propagation NN has to be selected carefully, which can be considered a drawback of using the NN. There exist NN that grow while training, the FANNC described in [24] has this property. Using such a classifier would remove the burden of deciding upon the size of the NN from the operator. A drawback with FANNC is that it does not generalize as well as the back-propagation NN, which is an argument for not using FANNC.

In this project the back-propagation NN is selected to be used for classification. This classifier has both drawbacks and advantages. A reason for selecting the back-propagation NN is that there is source code readily available on the Internet. If one of the more exotic classifiers was selected, then we would have to implement it; which would be a time consuming task, and one which is not the core topic of this thesis.

6.2 The back-propagation classifier

This section briefly describes how an back-propagation NN works, so that we can interpret the results in the testing chapter. The internal settings used for the NN are given in the end of this section.

Figure 6.1 shows the structure of a feed-forward NN. It is called feed-forward because the input only propagates from the input layer to the output layer.

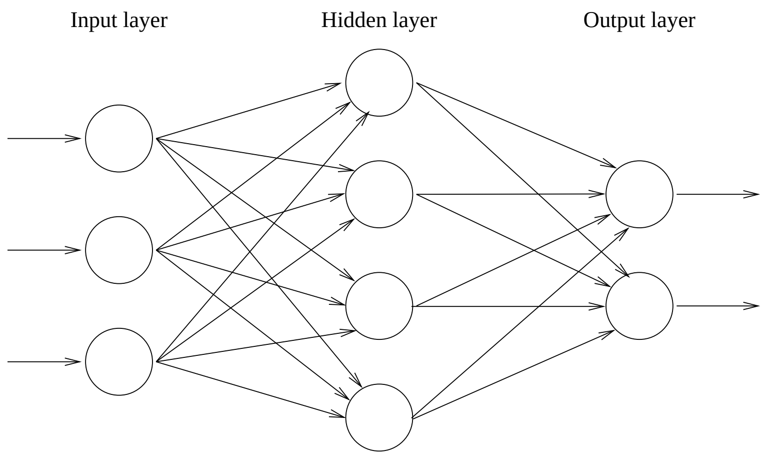

Figure 6.1: A set of neurons connected in a feed-forward neural network.

The first layer from the left is called the input layer, and the last layer is called the output layer. The layers in between the input and output layers are called the hidden layers.

The idea is that we should present the inputs with the feature values, and the neural network should give a result, so that the entry can be classified as ambient traffic or attack traffic. On figure 6.1 there is only one hidden layer, but there can be more.

The circles in figure 6.1 symbolize neurons. Figure 6.2 shows the structure of a neuron. The inputs to the neuron are weighted by individual weights for each input. The weighted inputs are summed together, and used as input to a limiter function. A limiter function is a non-linear function, which output is between 0 and 1. The output from the limiter, is the output of the neuron, which can be passed to other neurons, or used as output of the NN.

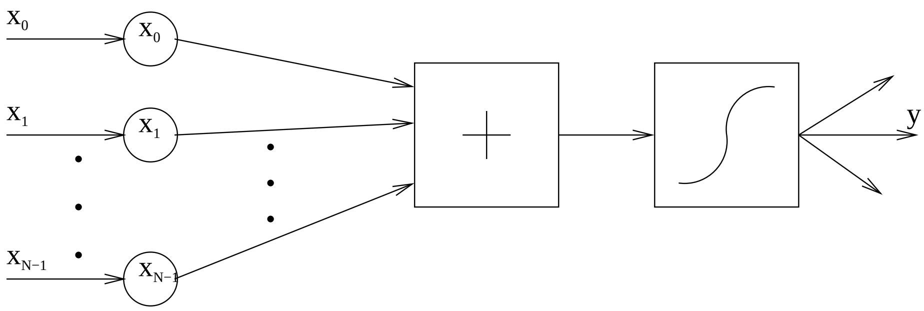

Figure 6.2: The structure of a single neuron.

The output of a neuron, $y(x)$, is given by equation 6.1.

$$y(x) = g\left(\sum_{i=0}^{N-1} w_i x_i\right) \tag{6.1}$$

Where N is number of inputs to the neuron, x_i is the input value for input i, w_i the weight assigned to the i'th input. The function g is the activation function, and the output of the neuron is y, which can be feed to other neurons or used as the output of the NN.

A commonly used activation function is the sigmoid function given in equation 6.2.

$$g(x) = \frac{1}{1 + e^{-x}} \tag{6.2}$$

We will use the sigmoid activation function for all the neurons; it gives outputs in the range $[0; 1]$. This is especially useful for the output neuron, as we will assign the to extremes as clear indications of a certain traffic type. Zero will be assigned to ambient traffic, and 1 to attack traffic.

The prediction running-time of a log entry is $O(1)$, because we do a constant number of calculation per prediction. The number of calculation depends on the number of neurons in the NN, and not on the pattern presented. $O(1)$ is good, because it is less complex than the feature extractors, which have $O(lg_2 n)$ running-time for each entry, where n is the number of elements in the sliding window.

Now we have the neural network, but the weights have not been assigned any values yet. Assigning values to the weights is handled by the learning process. The learning process the consists of presenting the NN with an input, and calculating the output. The output is then compared with the correct result, and the error is used to correct the weights. The mean square error is often used as a measurement for the error.

A commonly used learning algorithm is the back-propagation, which is the one we will use. We will not investigate how it works any further. But we should just mention, that *epochs* can roughly be said to be the number of times that the back-propagations algorithm modifies the weights of the neural network. We will use the term *epochs* in the testing section.

Implementing a back-propagation NN would be outside the frame of this thesis, but there are many freely available on the Internet. We have selected one, FANN [25], which seems to be quite popular and stable .

This brief look at the back-propagation NN has left many parts of the system out, to keep things simple, but it should give the reader a feeling of how the system work in general.

6.3 Summary

We have seen that there are many ways the classifier can be implemented. But the back-propagation NN was selected, because of its ability to generalize, and that it is a well known system. We have now gained a rough understanding of how the NN classifier works, which will help us to understand the results in the next chapter. The prediction running-time for a NN based classifier is faster than the feature extractors, so it does not impose a problem.

Chapter 7

Testing

In this chapter the feature extractors and the classifier will be tested. The purpose of testing the system, is to see how the effective the ranking of the features has been, and to see how the back-propagation classifier will work with the features.

The main objectives of the tests are:

- To examine how effective the intrinsic and traffic features are alone.
- Examine the effect of combining the intrinsic and traffic features.
- To see how the manually selected set and DARPA set differ.

To make things easier, we will call the ambient traffic from Monday week 1 from the DARPA set for DARPA1, and the ambient traffic from Monday week 3 for DARPA2. The same goes for the manually selected traffic, which we will call for MANUAL1 and MANUAL2.

For all the following tests we have used the standard setup of the NN classifier, and a window size of 120 seconds. The training and testing sets are ambient traffic merged with the host scan used for the feature extractor development.

But before we go on to the tests, we have to define how we measure the performance of the system.

7.1 Performance

When evaluating the flow entry classifier a performance measurement is needed.

There are four possible outcomes when comparing the correct classification of an entry and the result from the classifier:

- True positive: $y_{cor}(n) = y_{res}(n) = 1$
- True negative: $y_{cor}(n) = y_{res}(n) = 0$
- False positive: $y_{cor}(n) = 0 \land y_{res}(n) = 1$
- False negative: $y_{cor}(n) = 1 \land y_{res}(n) = 0$

Where n is the entry number, $y_{res}(n)$ is the output of the classifier, $y_{cor}(n)$ is the correct classification. The value 1 indicates that the entry is part of an attack, and the value 0 that it is not.

The number of correct answers in a log, L, consisting of N entries is therefore:

$$cor(L) = \sum_{n=0}^{N-1} \left\{ \begin{array}{ll} 1 & \text{for} \quad y_{cor}(n) = y_{res}(n) \\ 0 & \text{for} \quad otherwise \end{array} \right\} \tag{7.1}$$

Which can be used to express the percentage of correct answers, $cor_{\%}$:

$$cor_{\%}(L) = \frac{cor(L)}{N} \tag{7.2}$$

The $cor_{\%}(L)$ cannot be used by itself as an measurement of performance of the classifier, one also has to know something about the data used for the test. The ratio between the number of attack and normal entries has something to say, and the difficulty of detecting the attack does also have a role. For example it would not be hard get a good $cor_{\%}(L)$ value for a large log consisting mostly of normal traffic.

According to [9] the best measure for the performance of the detection system is the true positive rate and the false alarm rate together. They also state that a false alarm rate over 100 a day make a IDS unusable, because it will require to much manual labour. It is questionable if the same goes for our forensic tool suite, as it tries to classify each entry in the log, and not only to detect the attack, as an IDS would.

The attack traffic used is the host scans we have been using up until now. Eache host scan contains 104 flows classified as attacks. We will report the performance as the number of false positives (fp) and false negatives (fn).

7.2 The intrinsic features alone

Now that we have clear definition of the performance of the system, we can continue by looking at the intrinsic features alone. First we will look at the unmodified flags from the flows in both direction. Afterward the *suspiciousflags* feature is examined.

7.2.1 The flags feature alone with DARPA

The flag field ranked very high using the manually selected data set and the DARPA set. It will be interesting to see how this feature does alone.

Test setup: All the TCP flags in both directions are extracted, and presented to the classifier as a feature value for each flag. If a flag is set, the feature for that flag has a value of 100, and if it is not set, then the feature has the value 0. The NN is therefore presented with 12 inputs. The NN has been set to have 100 neurons in the first hidden layer, and 20 neurons in the second hidden layer. The training set used was DARPA1, and the testing set was DARPA2.

Results: After 10.000 epochs, the training set had $fp = 0$, $fn = 76$, and $cor = 85.112$. The testing set had $fp = 0$, $fn = 77$, and $cor = 77.096$. The bad results are due to the fact, that the NN has insufficient information to successfully classify the entries. We will look closer at why, when looking at the *suspiciousflags* feature in the next test, because it is easier to explain with only one input.

7.2.2 Using the *suspiciousflags* feature alone with DARPA

In the previous test we saw that it was possible for the classifier to distinguish some of the attack traffic from the ambient traffic only by looking at the flags in both directions. Now we will examine how the *suspiciousflags* will do by itself. It is an overkill to use a NN classifier in this case, as we only have one input, and a simple threshold would suffice. But we will use the NN classifier anyway, to see how it works with this simple problem.
Test setup: Only the *suspiciousflags* feature is extracted and presented to the classifier. The DARPA1 is used for training and DARPA2 is used for testing. A NN with one hidden layer with one neuron is used.

Results: The classifier does not detect any of the probe flows. After a closer examination of the classifier output, it can be seen that the scans are assigned a value of 0.06. This is an interesting phenomenon, because the classifier does not have enough information in the one feature. The feature is 100 when a flow belongs in the attack category, but it is also 100 when it belongs in the ambient category. After a closer examination of the classifier's output values, it was found that it were the flows in the traffic that had a forward "SYN" flow, and backward "RST SYN", which the feature assigned a value of 100. In the training phase, the classifier tries to minimize the mean square error, and therefore its output will be somewhere in between 0 and 1 for attacks, depending on the ratio between the number of attacks and ambient flows getting assigned 100 from the feature extractor. When the classifier does not have enough information to learn from, then the output values for the attack traffic becomes lower (if there is more ambient traffic than attack traffic). The same thing happened when presenting the flags to the classifier, the only difference being that there were more inputs. One possible solution would be to remove the pattern given by equation 7.3 from the feature, so that it only would detect the stealthy scans.

$$\neg\mathrm{A}\ \neg\mathrm{P}\ \neg\mathrm{R}\ \mathrm{S}\ \neg\mathrm{F}\ /\ \mathrm{A}\ \neg\mathrm{P}\ \mathrm{R}\ \neg\mathrm{S}\ \neg\mathrm{F} \tag{7.3}$$

But there are other better solution, as we will see in the rest of this chapter.

7.2.3 DARPA set with *suspiciousflags* and transferred octets

We will now use the value reduced transferred octests from the forward flow and *suspiciousflags* together. The reason for including the transferred octets in the forward direction, is that they ranked second highest of the value reduced features using DARPA traffic, see figure 5.5. If we did not use value reduction, then the packet count would be the next feature that we included, see figure 5.2. At the end of the test we will try to replace the forward transferred octets with the forward packet count, and see how it does. We will also test the transferred octets with two different cutoff values to see if there is any difference.
Setup: The network has 5 neurons in the first hidden layer. First a cutoff = 50 bytes used, and then cutoff = 500 bytes.

Results: After 110 epochs there were $fp = 0$ and $fn = 2$ for the training set, and $fp = 0$ and $fn = 2$ for the testing set. After that the results did not improve further.

The two scan flows that were not detected, were both a TCP connect scans finding an open port.

The results from using cutoff = 500 bytes are the same, but it took 230 epochs to obtain the same result.

The results from these two simple test are interesting. The results are the same using 50 and 500 as cutoff values, but the only thing that differs is the training time. So in this case the classifier has not been able to use the additional range, and it has only made the problem more complex.

Estimating from our initial feature ranking in figure 5.2 we would not think that there was any information in the other features. After the flags it would be the packet count that ranked highest, but we have found that the transferred octets have a value reduced rank that is around three times higher than the packet count, see figure 5.5.

We have done the exactly the same test, but with only the transferred octets features replaced with the packet count. After 2000 epochs the results were $fp = 0$ and $fn = 104$. The reason was the NN classifier did not have enough information to distinguish the attacks from the ambient traffic. So we can conclude that the value reduced feature ranking is correct in this case.

7.2.4 DARPA set with *flags* and transferred octets

As we are able to find many of the scan flows with the *suspicousflags* and transferred octets, it would be interesting to see if this is possible with the flags divided into individual feature values, as we have done previously. One could also suspect the "URG" flags for not being important in detecting a scan, so we will run a test without them to see what happens. After the test is done with the "URG" removed, we will also change the cutoff of the octets from 50 to 500 bytes, and run the test again to see if there is a difference.

Setup: A NN with 50 neurons in the first hidden layer. First with "URG" flags, then without them. A final test is done with the "URG" removed, and the octet cutoff increased to 500 bytes.

Results: With the "URG" flags: After 440 epochs the result was $fp = 0$ and $fn = 2$ for the training set, and $fp = 0$ and $fn = 3$ for the testing set.After 1910 epochs it was $fp = 0$ and 0 for the training set, and $fp = 0$ and $fn = 1$ for the testing set.

Without the "URG" flags: The results after 360 epochs were $fp = 0$ and $fn = 2$ for the training set, and $fp = 0$ and $fn = 3$ for the testing set. After 1400 epochs it was $fp = 0$ and 0 for the training set, and $fp = 0$ and $fn = 1$ for the testing set.

From this test we can conclude, that one can use the flags without to much modification, and that the "URG" flag is not important for this case. Without the "URG" flags, the training needed is about 80% of the epochs that were required with them present, and the same results were obtained.

The result is better than the one obtained with *suspiciousflags* in the previous test. The reason is that by presenting the classifier with the flags, it can distinguish between the scan types, and therefore also give different weights for each to the transferred octets. With *suspiciousflags* the classifier only knows that the flag combination is suspicious, but not how it is suspicious.

The results of moving the octet cutoff up to 500 almost the same. After 1280 epochs the result is $fp = 0$ and 0 for the training set, and $fp = 0$ and $fn = 1$ for the testing set.

7.2.5 Intrinsic features alone with real traffic

By comparing the feature ranking values given to the feature for real and DARPA traffic, we can clearly see that there is much harder to detect a scan based only on the intrinsic features. But we will give it a try, and use three highest ranking feature for the real traffic. The *suspiciousflags*, forward octets, and reverse octets.
Setup: A NN with 10 neurons in the first hidden layer.

Results: After 100 epochs the result was $fp = 0$ and $fn = 28$ for both the real sets, and did not improve any further.

This can be compared to the result $fp = 0$ and $fn = 2$ using DARPA traffic, and only with the *suspiciousflags* and forward octets features.

7.2.6 Summary

We have seen what happens to the NN based classifier's results when there is not enough information to distinguish the traffic types. We have seen how important the value reduction is to finding the real rank of features, and that it can be used to speed up the learning process of the NN based classifier. We have also seen that it is easier to detect attacks in the DARPA set.

7.3 The traffic features alone

In this section we will look at how the traffic features can be used alone to detect scans. We will try to use the *thb_rst_count* alone to detect host scans in both manually selected real traffic and DARPA traffic.

Setup: A NN with 5 neurons in the first hidden layer. Manually selected traffic 1 for training, and 2 for testing.

Results: After 10.000 nothing was detected, $fp = 0$ and $fn = 104$ for both the training and testing set.

Setup: A NN with 5 neurons in the first hidden layer. DARPA traffic 1 for training, and 2 for testing.

Results: After 10 epochs all scan flows were detected; the result was $fp = 0$ and $fn = 0$ for both the training and testing set. This corresponds well with figure A.

Again we can see that it is easy to detect the scans in the DARPA set, but hard in the real traffic.

7.4 Traffic and intrinsic features combined

In this section we will look at how the traffic and intrinsic features can be combined to increase the performance of the system. We have already had good success detecting the probes in the DARPA set, so we will only focus on the manually selected traffic.

7.4.1 Using *suspiciousflags* and *thb_rst_count*

Now the two best features from the intrinsic and traffic features, *suspiciousflags* and *thb_rst_count*, will be tested together.

Setup: A NN with 10 neurons in the first hidden layer. Manually selected traffic 1 for training, and 2 for testing. The training set and the testing set are swapped afterwards, for a second test.

Results: After 2500 the result was $fp = 1$ and $fn = 2$ for the training set, and $fp = 0$ and $fn = 25$ for the testing set. The results did not improve further.

When swapping the training and testing set, the results were quite different. Only after 20 epochs the results were $fp = 0$ and $fn = 2$ for the training set, and $fp = 3$ and $fn = 14$ for the testing set. Afterwards it does not improve any further.

The reason for gettig good results with the training set, and bad results with the testing set, is because both sets contain traffic that is not present in the other. Including more feature does not solve the problem, and the only solution is to get two larger sets, which contain more versatile representation of ambient traffic.

7.5 Test summary

We have seen that it can speed up the classifiers learning speed, by making specialized features, such as the *suspicousflags* feature. In this case the training epochs required were about $\frac{1}{4}$ of the require using the flags themselves as features. But there were some drawbacks discovered when using the more specific features, such as the classifier not knowing what kind of suspicious flags they were. But the *suspicousflags* have not been useless in the development of the features extractors, without them we would not know the value of the ranking of using the flags in both directions.

Chapter 8

Conclusion

In this thesis we have developed a forensic tool suite for analyzing Netflow based router logs. Specialized feature extractors were developed for detecting host scans, but it is clear that the process used to create the feature extractors could be used to create feature extractors for other attacks. The back-propagation neural network based classifier was used successfully to distinguish the ambient traffic and the attacks based on the feature extracted.

Many problems associated with Netflow based logs have been solved. A simple method was developed for grouping associated flows together, and to find their corresponding reverse flows. We have also seen that the flag field in the Netflow record is very valuable for detecting scans, even if it is handled quite crudely by the Netflow aggregation scheme.

Using a red-black tree as a data structure has been largely successful - it provided fast lookup of both forward and backward flows. Using various fields from the identification of a flow as the key in the tree, meant that we could developed many different features.

One of the main problems with this project is that there is no way of producing a large set of labeled traffic. We have only used three hours of selected traffic from a real log of one router, which cannot be representative enough to describe all the ambient traffic on routers. This means that it is doubtful if the attack detector will work as expected when used on router traffic obtained from other sources. It is therefore necessary to develop a method to gather much more labeled router log data. One method could be the one described in chapter 4.4 on page 27.

So even if we have had access to a real router log, we have not been able to utilize it fully in the development of the system, because there was no efficient way to label the traffic. This does however not mean that we did not gain anything from having access to a real router log. We discovered that there is a huge difference between the traffic on the router and the outside network in the DARPA evaluation set. The DARPA evaluation set does not model real router traffic very well; the statistical differences are quite large, and the scans are much easier to detect in the DARPA traffic. It was possible to detect all the scans in the DARPA traffic just by using the intrinsic features, which is not a realistic scenario. So we have found out that it is not possible to use the DARPA set for developing a good set of feature extractor or use it for training a classifier, which should be used on a real router. The DARPA set is still the only publicly available test bench for IDS, and it can still be used for comparing the performance of IDS. But the performance measured, does not translate into performance in real traffic.

8.1 Future work

From this thesis one might get the impression that there is much more work that can be done to improve the attack detector, which is true.

In this thesis we have been looking at feature extractors for host scans, but for other types of attack traffic, there will be other features which are better. Finding them can be a difficult task, and we have seen that Lee et al. [19] have tried to solve the problem by developing an automatic way of creating feature extractors. Another alternative to automatic feature creation, would be to have an large amount of features, and then select some of the best ones using ranking. A way to create a large number of features, would be to have an publicly open database, were people could submit features that they have created. For such an system to work, a simple language for describing feature extractors based on network traffic would be needed. Developing a language for describing features based on network traffic could be some of the future work in this area.

The feature ranking method used in this thesis is quite common, and it has some disadvantages. It could be improved by developing a hybrid between the entropy based ranking method and the wrapper approach, so that features that are very correlated would not both be included in the final feature set.

Other important future work consists of developing a robust method to create ambient traffic, as we have mentioned earlier, so that large amounts of representative traffic could be created for developing and testing of the attack detector. This would not only benefit our project, but would also benefit most other attack detectors based upon network traffic.

If all these problems were solved, then the attack detector would become an effective tool for analyzing traffic.

Bibliography

[1] A. Hussain, J. Heidemann, and C. Papadopoulos, “A framework for classifying denial of service attacks,” in *Proceedings of ACM SIGCOMM*, 2003.

[2] Forskningsnettet, “Forskningsnettet offical web page, last modified: 10 dec 2004.” `http://www.forskningsnettet.dk/`.

[3] R. Sommer and A. Feldmann, “Netflow: Information loss or win?,” in *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW) 2002*, ACM Press, 2002.

[4] Cisco, “Cisco ios netflow technology data sheet.” `http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/iosnf_ds.htm`.

[5] Cisco, “White paper: Netflow services and applications, last modified: Jul 15 2002.” `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.h%tm`.

[6] IETF, “Ip flow information export (ipfix), last modified: 13 sep 2004.” `http://www.ietf.org/html.charters/ipfix-charter.html`.

[7] W. Lee and S. J. Stolfo, “A framework for constructing features and models for intrusion detection,” *ACM Transactions on Information and Computer Security, Vol. 3. No. 4, November 2000, Pages 227-261*, 2000.

[8] W. Lee, *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, 1999.

[9] J. W. Haines, R. P. Lippmann, D. J. Fried, E. Tran, S. Boswell, and M. A. Zissman, “1999 darpa intrusion detection system evaluation: Design and procedures,” tech. rep., MIT Lincoln Laboratory Technical Report, 2001.

[10] J. W. Haines, L. M. Rossey, R. P. Lippmann, and R. K. Cunninham, “Extending the darpa off-line intrusion detection evaluations,” in *DISCEX 2001, June 11-12*, 2001.

[11] M. Fullmer and S. Romig, “The osu flow-tools package and cisco netflow logs,” in *Proceedings of the Fourteenth Systems Administration Conference (LISA-00)*, pp. 291–304, The USENIX Association, 2000.

[12] S. S. Kim, A. L. N. Reddy, and M. Vannucci, “Detecting traffic anomalies through aggregate analysis of packet header data,” in *Proceedings of Networking 2004, LNCS 3042*, pp. 1047–1059, May 2004.

[13] S. S. Kim and A. L. N. Reddy, "A study of analyzing network traffic as images in real-time," in *Proceedings of IEEE INFOCOM 2005*, (Miami, Florida, USA), Mar. 2005.

[14] A. S. Tanenbaum, "Computer networks." Third Edition, 1996, ISBN: 0-13-394248-1.

[15] S. Gibson, "Gibson research corporation: Distributed reflection denial of service, last modified: 22 sep 2002." `http://grc.com/dos/drdos.htm`.

[16] T. Vogt, "Application-level reflection attacks, modified: 5 jul 2002." `http://www.lemuria.org/security/application-drdos.html`.

[17] R. Basu, R. K. Cunningham, S. Member, S. E. Webster, and R. P. Lippmann, "Detecting low-profile probes and novel denial-of-service attacks," in *Proceedings of the 2001 IEEE: Workshop on Information Assurance and Security*, United States Military Academy, 2001, 5-6 June.

[18] B. Carrier, "Defining digital forensic examination and analysis tools using abstraction layers," *International Journal of Digital Evidence*, vol. 1, no. 4, 2003.

[19] W. Lee and S. J. Stolfo, "A framework for constructing features and models for instrusion detection systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 227–261, 2000.

[20] NLANR, "Nlanr network traffic packet header traces, last modfied: 30 dec 2004." `http://pma.nlanr.net/traces/`.

[21] D. Miller, "Softflowd, last modified: 30 sep 2004." `http://www.mindrot.org/softflowd.html`.

[22] W. Duch, "A framework for constructing features and models for instrusion detection systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 227–261, 2000.

[23] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, th ed., 1949.

[24] Z. Zhou, S. Chen, and Z. Chen, "Fannc: a fast adaptive neural network classifier," *Knowl. Inf. Syst.*, vol. 2, no. 1, pp. 115–129, 2000.

[25] S. Nissen, "Fast artificial neural network library (fann), last modified: 2 nov 2004." `http://fann.sourceforge.net/`.

Appendix A

Value reduction figure and distributions

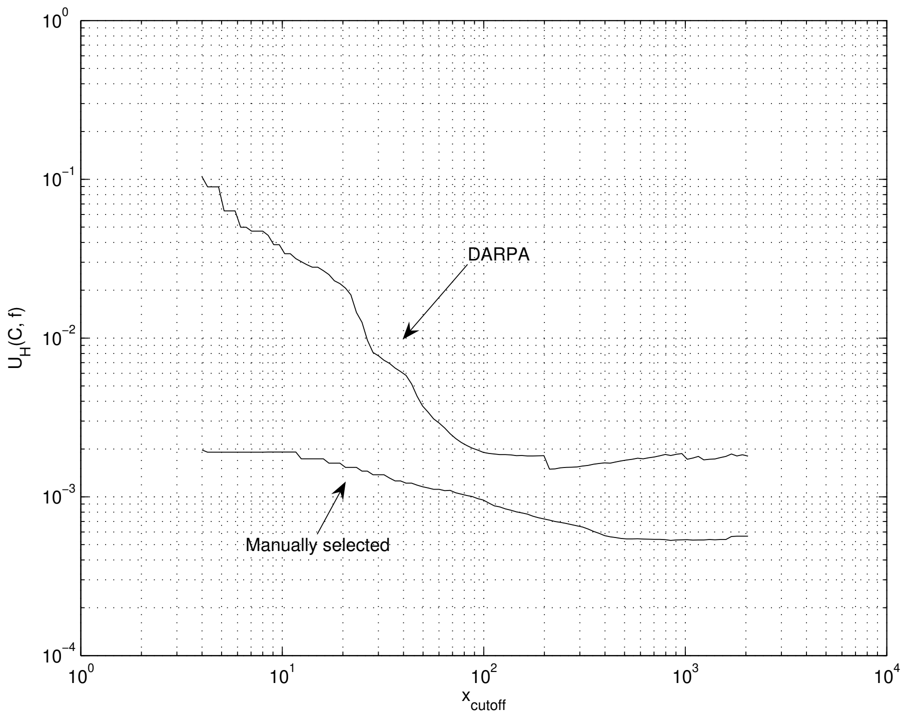

Figure A.1: Entropy index as a function of the duration cutoff value.

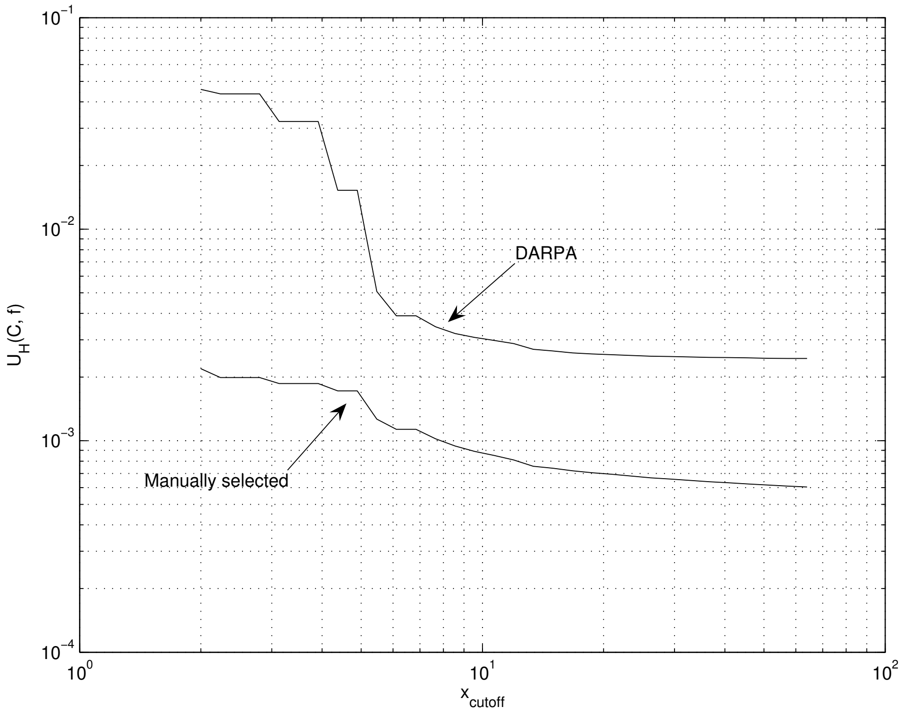

Figure A.2: Entropy index as a function of the packet count cutoff value.

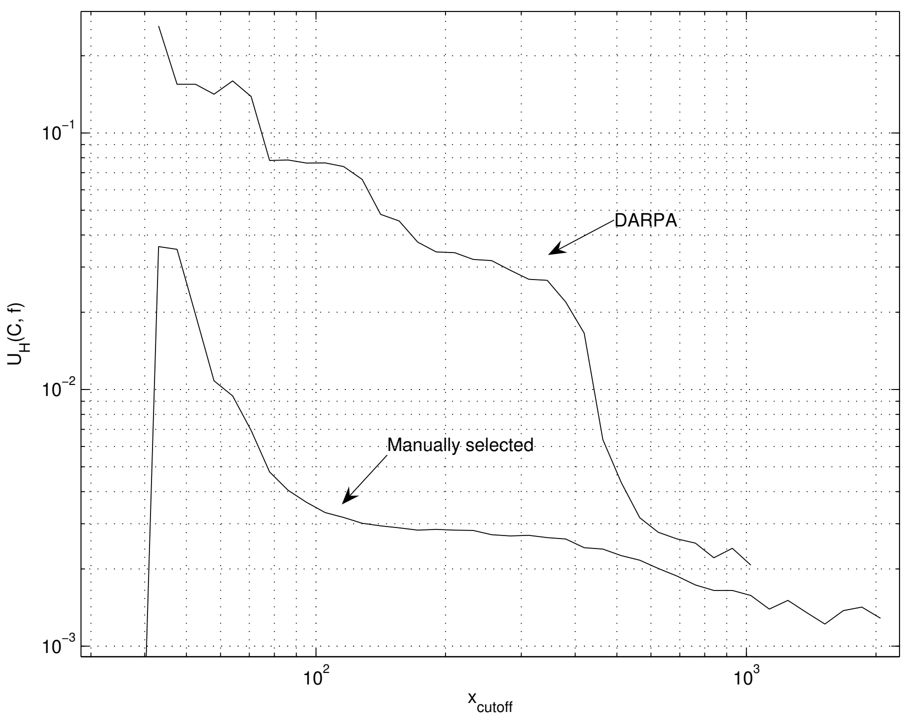

Figure A.3: Entropy index as a function of the transferred octets cutoff value.

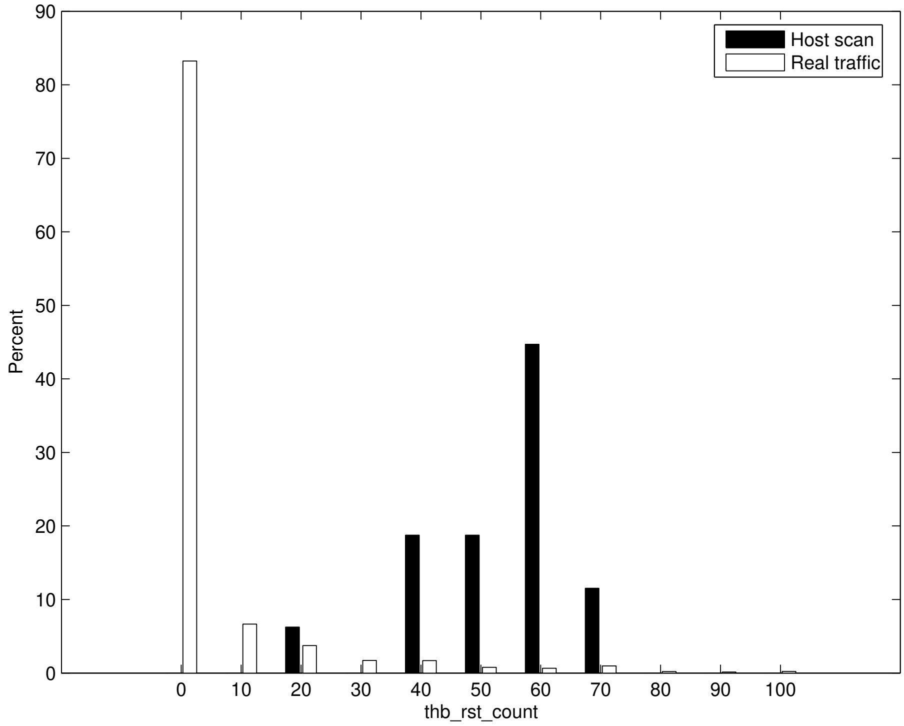

Figure A.4: Histogram showing the distribution of the reverse *thb_rst_count* with real traffic.

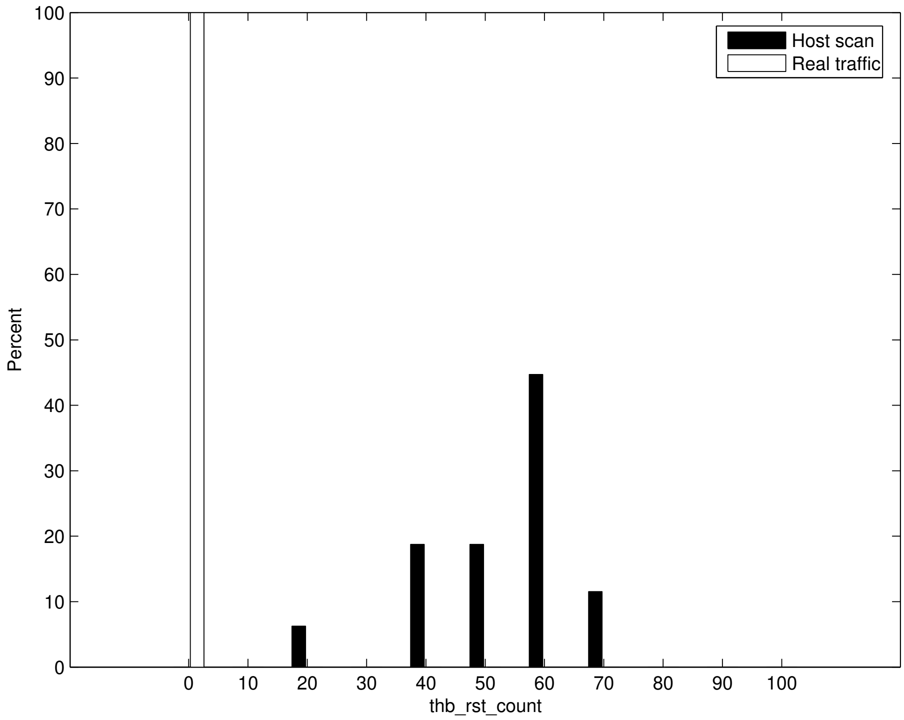

Figure A.5: Histogram showing the distribution of the reverse *thb_rst_count* with DARPA traffic.

Appendix B

Netflow version 5 datagram

There are several Netflow datagrams, but the one used for this project was version 5. Figure B.1 shows the header of the datagram, and figure B.2 show a record from the datagram.

Bytes	*Content*	*Description*
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	reserved	Unused (zero) bytes

Table B.1: Netflow version 5 header.

Bytes	*Content*	*Description*
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes
37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol type (for example, TCP = 6; UDP = 17)
39	tos	IP type of service (ToS)
40-41	src_as	Autonomous system number of the source, either origin or peer
42-43	dst_as	Autonomous system number of the destination, either origin or peer
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (zero) bytes

Table B.2: Netflow version 5 record.