NTNU

Norwegian University of
Science and Technology

Anomaly Detection and Identification in Feature Based Systems: An Empirical Evaluation

Magnus Bjørnar Røgeberg Ask
Helge Skrautvol

Master of Science in Communication Technology
Submission date: June 2011
Supervisor: Yuming Jiang, ITEM
Co-supervisor: Atef Abdelkefi, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem description

Intrusion detection is an important technique in computer network security. Senatus is a concept recently proposed in order to detect intrusions in a fashion of traffic classification and identification.

There are two main tasks:

I. Understand network attacks and their simulation tools then generate the attacks. Then simulate attacks in real backbone network, collect the data of the form Netflow/Qflow and prepare it to the analysis.

II. Senatus performance analysis investigation.

Abstract

Network anomalies can range from network outages and flash crowds, to malicious attacks such as denial of service attacks and port scans. Identifying the anomalies you are dealing with is paramount to take proper counter measures. In a world where more and more mission critical equipment are connected to the Internet, protection against an increasing number of professional cyber criminals are more important than ever. In addition to this, large firms cannot afford long down times due to benign anomalies that could have been detected and avoided.

The aim of our thesis is to compare three different implementations of feature-based detection systems. Our dataset is based on traffic recorded over several months on the Norwegian backbone network of Uninett. We will compare the newly proposed anomaly detection system SENATUS with histogram-based detection – represented by Apriori – and Entropy based detection. Our evaluation is based on two techniques: root cause analysis and a novel technique based on the results of other anomaly detectors. We find that the newly proposed detection system SENATUS performs very well for detecting denial of service attacks and scans. It has a high detection rate and an exceptional identification rate. We find that identifying the root cause of an anomaly is drastically less time consuming with SENATUS compared to the other methods.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfilment of the requirements for the degree of master of science.

This work has been performed at the Department of Telematics, NTNU, Trondheim, with Yuming Jiang as professor and with Atef Abdelkefi as supervisor.

We thank our supervisor Atef Abdelkefi for his astonishing effort at helping us get the best results. Rune Ask for giving us valuable feedback on earlier drafts. Arne Øslebø at Uninett for letting us use the equipment and infrastructure, and for all his helpful advice. And finally, we would like to thank our professor Yuming Jiang.

List of Tables

List of Figures

Abbreviations

FSD flow size distribution

DoS denial of service

DDoS distributed denial of service

TP true positive

TN true negative

FP false positive

FN false negative

BASH Bourne-again shell

SED Stream editor

RPCA robust principal component analysis

KL Kullback-Leibler

PCA principal component analysis

GB gigabyte

TB terabyte

PPS packets per second

BPS bytes per second

BPP bytes per packet

IP Internet Protocol

TFD traffic feature distribution

AS Autonomous System

ED entropy-based detection

HD histogram-based detection

AR Association rule

Nfcapd NetFlow capture daemon

RFC Request for Comments

IDS intrusion detection system

CPU Central Processing Unit

Contents

1

Introduction

Detecting and identifying network anomalies are becoming an important factor to consider when taking care of network security, uptime and performance of ISPs and large scale networks. An **anomaly** is defined as a "Deviation or departure from the normal or common order, form, or rule"[28]. When studying anomalies affecting computer networks, we consider events that differ from normal network behavior, such as use of new protocols, significantly increased traffic and malicious attacks. The foremost challenge in detecting and identifying anomalies, is the fact that they can be caused by a vast set of events. These range from anomalies like flash crowds[1]

[1] An unusually high amount of traffic destined to one destination from a set of IPs. A situation that can happen if a high number people access the same website simultaneously.

and network outages, which have no ill intent behind them to malicious attacks such as denial of service (DoS) attacks and port scans (see Section 2.2). Since there are a high number of anomalies posing different levels of threat to systems, being able to quickly assess what anomalies are present is paramount in keeping a network healthy.

1.1 Detection and Identification Schemes

During our thesis we aim to compare three different approaches for anomaly detection and identification. All techniques are based on traffic features derived from data captured in a packet switched backbone network. The first scheme (entropy-based detection (ED)) apply entropy as its main feature to detect anomalies. The second (histogram-based detection (HD)) is a histogram-based detection scheme which uses Kullback-Leibler distance and Apriori in order to detect and identify anomalies. SENATUS is the last scheme, and it uses a robust version of principal component analysis for detection and identification.

Figure 1.1 is a high-level illustration of the general concepts applied in each scheme. The first and the second step are applied to all schemes, while the third is only applied to HD and SENATUS.

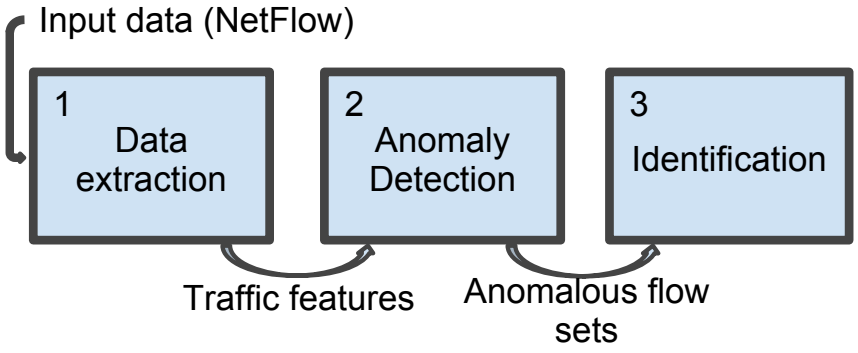

Figure 1.1: General concepts behind detection and identification schemes.

All the aforementioned schemes will be explained, evaluated and discussed in this thesis.

1.2 Equipment and Infrastructure

To be able to conduct experiments in a reliable and convincing way UNINETT[31] has provided access to data and infrastructure in the Norwegian research network. The network connects well over 200 Norwegian educational and research institutions with over 300 000 users and further on links them to international research networks.

The analysis is performed based on data in the NetFlow export format by Cisco, which is fully described in Request for Comments (RFC) 3954 [14]. An Internet Protocol (IP) flow, in the NetFlow format, is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets in a particular flow have a set of common traffic features derived from the data contained in the IP packet and from the packet treatment at the observation point.

Throughout the network UNINETT has deployed several NetFlow capture daemons (Nfcapds). Nfcapd is a program within the nfdump tool suite (see Section 3.2), and it reads traffic data from the network and stores it into files. The output file is automatically rotated and renamed every 5 minutes, hence one day is divided into 288 five minute intervals according to the timestamp "YYYYMMddhhmm".[2] The Nfcapd of our interest resides within the Oslo gateway (oslo-gw.uninett.no) and forwards the captured data to its final destination on iou1.uninett.no (Figure 1.2).

[2] e.g. nfcapd.201011050845 contains the data from November 5th 2010 08:45 and onward.

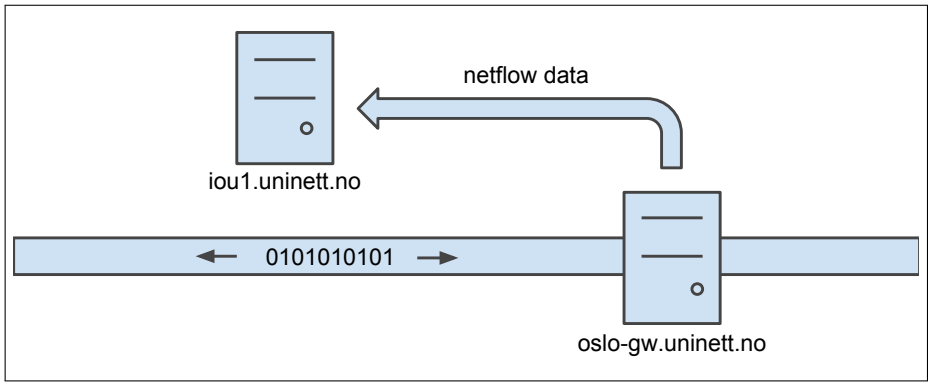

Figure 1.2: Network data is captured by a NetFlow capture daemon in oslo-gw.uninett.no and stored at iou1.uninett.no.

Topology

Figure 1.3 shows a compact view of the topology of interest.[3] Most of the analysis is conducted on the same server as where the captured data resides, namely the iou1 server.[4] At the other end – ytelse2 is used for injection of anomalies, such as port and network scans. We make sure that all the injected traffic traverse the oslo-gw and thereby get captured by Nfcapd for further studies.

Sampling

Because of the huge amount of traffic traversing the oslo-gw, it is not feasible to capture every packet – this is simply too much data for the gateway to cope with. To bypass this problem, the traffic is captured by random sampling at a rate of 1:1000, meaning that Nfcapd randomly chooses to include 1 out of 1000 packets. The fact that the traffic is subject to sampling might affect the results depending on the detection technique. Brauckhoff et. al. empirically evaluated the impact of packet sampling on anomaly detection methods [11]. While byte and packet count is rather unaffected, packet sampling produces inaccurate estimates of flow count. As a con-

[3] Map source: http://www.map-of-norway.com/

[4] The iou1 server runs on a dual-core Intel(R) Xeon(R) E5430 CPU with 32 GB system memory and 4 TB disk space dedicated for our use.

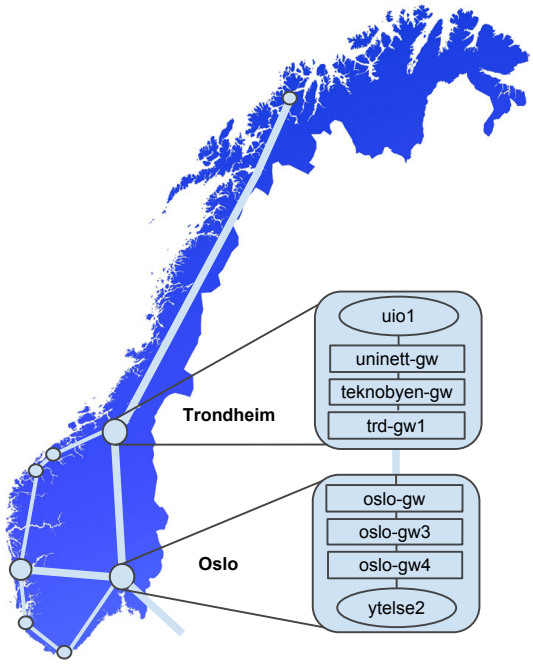

Figure 1.3: Network topology in UNINETT's network [30] between Oslo and Trondheim.

sequence, single packet flows are entirely missed, and volume-based techniques are unable to detect anomalies such as worms. However, feature-based techniques (such as ED, HD and SENATUS) proves to withstand high sampling ratios, thus are still able to detect anomalies.

1.3 Problem Outline

Our work is focused on comparing the recently proposed anomaly detection system SENATUS with other detection methods consisting of ED and HD[citation]. The goal of our work is discover how high the detection and identification rate of SENATUS is, compared to the other methods. Evaluating and comparing anomaly detectors is not an easy task. There are mainly two approaches we will use when evaluating SENATUS with regard to other anomaly detectors. The first method is to inject attacks in the network and then evaluate the performance of each anomaly

detector, based on the known attacks. By implementing working versions of the two aforementioned techniques, we aim to extract the injected anomalies from our dataset with the intent of constructing an artificial ground truth (see Section 4.1). The second method is using the results from ED and HD as a reference point. By running SENATUS on the union of time bins flagged as suspicious by the other two techniques, we can be reasonably confident that any anomalies SENATUS detect are not false positives. The reasoning behind constructing this ground truth, is that one can never be sure whether an anomaly flagged by one of either programs should have been flagged as such. With both anomaly detection programs marking a time bin as suspicious, we can be reasonably sure that there is an anomaly there.

1.4 Outline of the Thesis

The thesis is structured as follows:

Chapter 2 reviews the general concepts behind an anomaly detection system and their evaluation metrics. It classifies two types of anomalies, DoSs and scans, and describes their characteristics. We present the motivation and techniques involved in three feature-based detection systems, more specifically entropy-based detection, histogram-based detection and SENATUS.

Chapter 3 will guide you through the implementation of three anomaly detection schemes. Moreover, we will explain in detail the work flow of every scheme as well as the tools used in order to realise them. This chapter also presents the tools used to inject attacks, and for analysis.

Chapter 4 describes the methodology and criteria used in order to produce our results.

Chapter 5 presents the results of our analysis. We evaluate and compare the three detection schemes presented in Chapter 3

Chapter 6 summarizes the contributions of this thesis and further more identifies open problems that might be worth investigating in the future.

2

Background

Anomaly detection in backbone (large-scale) and mid-sized computer networks has been a field of research for several years. From a high level view point, anomaly detection is generally divided into three categories (schemes) [7]:

I. Supervised

II. Semi-supervised

III. Unsupervised

Supervised refers to detection techniques that learn to classify anomalies by knowing the pattern of both normal and anomalous traffic. In general, these techniques

consists of two phases, (i) a learning phase and (ii) the actual detection phase. In the learning phase one needs to collect and label data traces to generate network behavior models. Based on these models, the system later detects traffic and classifies it to be either normal or anomalous. Supervised detection techniques fail to recognize behavior that is not previously modeled, thus it lacks the ability to classify unknown or emerging anomalies.

Semi-supervised detection systems require knowledge of normal traffic traces; that is traces without any anomalies present. The typical approach is to create models based on the non-anomalous traffic traces, and then generate an alert whenever it detects a deviation from this model. A significant drawback in semi-supervised detection is the confrontation with previously unseen, yet legitimate traffic. The traffic is flagged as anomalous and increases the possibility of a high false positive (FP) rate.

In unsupervised detection, it is assumed that anomalies are very rare compared to normal data. Thus, the main principle in an unsupervised detection system is to compare the current interval to either the previous interval or a reference interval, in order to detect abnormal behavior. If there is no, or little ”difference” between the two compared intervals, the current interval is perceived as non-anomalous. The ”difference” can be detected with a wide range of techniques, where relative entropy, Kullback-Leibler (KL) distance and clustering are three such techniques.

The modeling, often referred to as *baseline*, can be done in several ways for both supervised and semi-supervised detection techniques. Since the traffic pattern in a network fluctuates, not only with the time of the day and the day of the week, but also with the week of the month and sometimes time of the year, one typically creates separate baselines for different intervals. The challenge of determining the number of models is a trade-off between complexity and accuracy. Creating a model for each time bin during a week ($288 \times 7 = 2016\ models$) will increase the detec-

tion accuracy, but will also have a tremendous effect on the complexity. Whereas only creating one model spanning a whole week will decrease the complexity, but anomalous behavior in one day might correspond to normal behavior another day, hence it will also decrease the accuracy of the detection.

2.1 Feature Versus Volume-based

Further classification divides the techniques described in literature into two groups, namely (i) volume-based and (ii) feature-based anomaly detection. Implementations of (i) and (ii) can adopt all three aforementioned schemes (supervised, semi-supervised, and unsupervised detection) depending on the techniques used for detection and whether or not the system must undergo a learning phase.

Volume-based detection models the traffic with respect to the number of flows, packets, bytes, and other volume metrics. A wide range of mathematical techniques can then be applied on volume metrics in order to detect deviations in the traffic pattern. The easiest approach is to extract volume metrics and simply visualize them in terms of a plot over time (e.g. #flows per time bin during a day), an approach that needs a relatively small amount of pre-processing. Other, more complicated approaches, involve wavelets to distinguish between predictable and anomalous traffic [8]. Volume-based techniques prove to be efficient in detecting deviations in network-wide traffic. However, anomalies that do not introduce a notable change in any volume metrics (e.g. scan anomalies) pass by unnoticed, a break-in to a network may therefore go undetected.

Feature-based techniques are different from volume-based techniques with respect to the measurement metric. The metrics used in a feature-based system is extracted from packet header fields (referred to as traffic features), and commonly includes [IP/AS(s+d), Port(s+d), Protocols, Packet size, Flow duration] (where s =

source, d = destination). The motivation behind this approach is to use traffic feature distributions (TFDs) created by capturing each traffic feature during a time interval, which has proved to detect a wider range of anomalies compared to volume-based techniques [18]. A common denominator for most feature-based detection systems is the use of information theory to compare differences between TFDs, e.g. entropy, KL distance and principal component analysis (PCA).

2.2 Anomalies

As mentioned in the introduction, anomalies do not necessarily originate from malicious events. A surge of traffic due to a popular link being shared on a social media website can have the same effect on a server as a distributed denial of service (DDoS) would have – leading to the site becoming unavailable. Table 2.1 shows how the different traffic feature distributions are affected by a number of anomalies according to Lakhina et. al.[18].

Anomaly	Definition	TFDs affected
Flash Crowd	Unusual burst of traffic to single destination, from a typical distribution of sources	Destination address Destination port
Alpha Flows	Unusually large volume point to point flow	Source address Destination address
DDoS	Large amount of traffic to one destination from several sources	Destination address Source address
Port Scan	Probes to many destination ports on a small set of destination addresses	Destination address Destination port
Port Sweeps	Scanning by worms for vulnerable hosts	Destination address Destination port

Table 2.1: A list of traffic feature distributions (TFDs) affected by anomalous events.

The following sections will describe the two anomalies we will focus on in our work: denial of service attacks and scans.

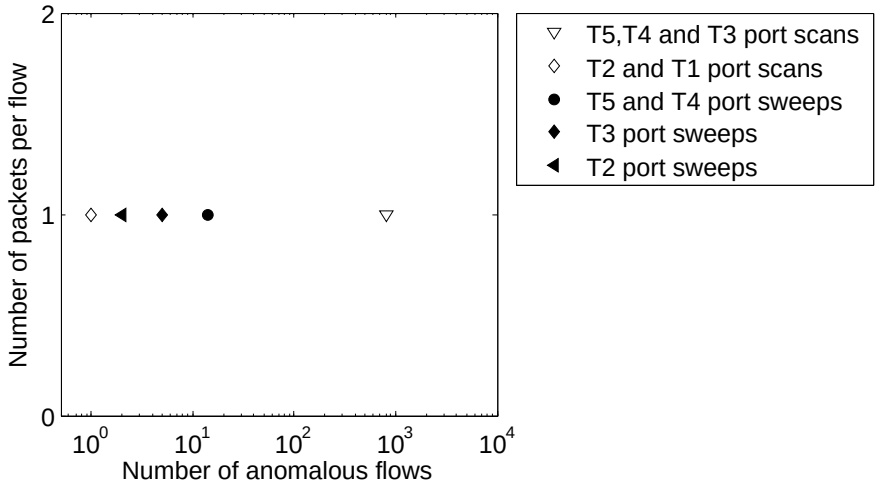

Figure 2.1: The number of packets per flow set up against the number of anomalous flows

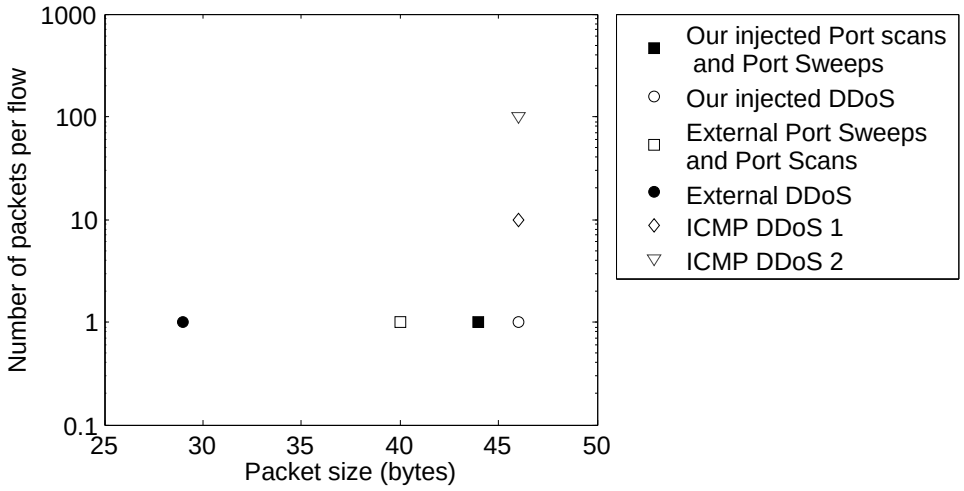

Figure 2.2: Packet size set up against the number of anomalous flows

2.2.1 Denial of Service Attacks

A denial of service attack is an attempt by one or more persons to take a computer resource out of service. In the case where there are several computers involved in the attack, it is known as a distributed denial of service attack. A denial of service attack can be achieved with different techniques, but a common method is to saturate one or several servers with a stream of requests for service. By keeping the server busy

with bogus requests, it will not be able to respond to legitimate users in a reasonable time.

A number of DDoS attacks originate from botnets [1] – a collection of infected computers where an attacker has exploited security vulnerabilities found on legitimate users' computers. A user may unknowingly join a computer to a botnet, if he or she downloads a bot agent (e.g. from an email attachment or through clicking a malicious link). The agent will then install programs to open up for remote control by the intruder[15].

Another form of DDoS attacks, one which has risen in prominence during the last years due to the advent of social networks, is a voluntary and cooperative form of attacks. People from all around the world can now, out of their own free will, join in on planned attacks against data centers or websites with the help of open source programs like LOIC [27]. The hacker group Anonymous [32] has been known to use, LOIC for attacks in this category. By spreading information (i.e. the IP address and a time window) about the attack through websites like 4chan and reddit, millions of potential users can join in. This has happened to both Visa and Mastercard after they closed down Julian Assange's account during the Wikileaks incident [12].

A denial of service attack can be perpetrated in a number of ways. The **syn flood** attack is based on a host sending a stream of TCP/SYN packets, often with incorrect IP addresses, in an attempt to flood the server. Each packet sent to the server is handled like a connection request, which has the effect that all resources used to establish a TCP connection are occupied [20]. An **ICMP flood**, is a type of DoS attack that sends large amounts of (or over sized) ICMP[2] packets in an attempt to crash the TCP/IP stack. A smurf attack is a variant of this that floods a vulnerable network with a number of spoofed ICMP echo request messages (ping) to the broad-

[1] Also known as zombie networks

[2] Short for Internet Control Message Protocol, it is one of the core protocols of the Internet Protocol Suite

cast address. If the router is not configured correctly, faking the source IP to appear to be the address of the victim will have the effect that most of the hosts will send an echo reply. With many hosts replying, this will multiply the traffic on the network, and cause legitimate traffic to be lost.

Figure 3.2 and 3.3 gives an example of how a DDoS attack appears when studying the NetFlow data with nfdump. As we can see, different source addresses all transmit to the same destination address (the target of the attack). In this case, the source port is fluctuating randomly (common with TCP/UDP DDoS attacks), which leads NetFlow to treat each packet as an individual flow.

A DDoS with ICMP packets is a bit of a special case. NetFlow will divide packets into flows based on protocol, src/dst IP and src/dst port. ICMP traffic does not have port numbers, so NetFlow will set the src port to 0, and the dst port to ICMP type and code. In other words, all the ICMP packets sent from the same IP address will be in the same flow, and the more intensive the attack is, the higher the number of packets per flow will be. This means that the number of packets per flow in an ICMP echo req DDoS attack scales with the intensity.

2.2.2 Scans

A large number of attacks start with a reconnaissance phase, where an attacker scans a large number of ports in order to expose vulnerabilities in his targets armor. RFC2828 [24] defines a port scan as "An attack that sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service". If a port with a known vulnerability is discovered to be open and listening, a hacker can exploit this.

Another variant of port scans are known as port sweeps – or worm scans, where an attacker scans a range of hosts for a single vulnerable port. This type of attack is

common among worms, who may employ the technique in exploiting a vulnerability inherent to a program listening for connections on a certain port. The amount of data sent out by Nmap differs slightly for the two types of scans. As can be seen on the anomaly spectrum in Figure 2.1, port scans have a slight increase in the number of packets sent out in the high intensity segment[3]. Note that this can differ for other types of scanners.

The anomaly spectrum in Figure 2.2 shows the difference in packet sizes between a scan and a DDoS. As we can see, the packet sizes are fairly small, with sizes ranging from 29 to 46 bytes. ICMP DDoS 1 and 2 represent the increasing number of packets per flow for ICMP attacks that scales with the intensity of the attack.

The principal challenge for an intrusion detection system with detecting scans, is finding an algorithm that can distinguish between a legitimate request and a malicious scan (e.g. a perpetrator trying to discover all the HTTP servers in a network).

Another challenge in detecting scans compared to attacks like DDoS, is the fact the amount of data transmitted are orders of magnitudes smaller. If an attacker wishes to keep the scans hidden, it will be virtually impossible for an intrusion detection system (IDS) to detect it. As shown in [26], the amount of data needed for a detection scheme like entropy to detect a port scan, surpasses 7% of the total data. The same amount of data needed to detect a port sweep is as high as 15%.

2.3 An Alternative Way of Defining Anomalies

Based on studying the anomalies with Nfdump (see 3.2), we find that we can define scans and DoS attacks with another method than by looking at what traffic distributions are affected, as described by A. Lakhina et al. [18]. By the nature of what it attempts to do – discover if a port is open, a scan has no need for large packet sizes.

[3]For the port scanner Nmap, a T5 scan is the highest intensity, and T0 is the lowest intensity.

By studying the traffic from both our injected scans, and the external scans found, we find that close to every packet sent, is under 50 bytes. In addition to this, since a scan is based around either scanning various hosts for one port, or various ports of one or more hosts, a scan will always contain one packet per flow (unless congestion control or a similar mechanisms make it send the same packet more than once).

In the case of DoS attacks, all the external attacks we discover have packet sizes of 29 bytes, and our injected attacks (done in [26]) are between 28 and 46 bytes per packet. As described earlier, certain DoS attacks (e.g. some forms of ICMP floods) have packet sizes which are large, and have several packets per flow. However, attacks based on IP broadcasts, such as ICMP floods and smurf attacks, are easy to block[19], and although there are likely to be a few attacks in this category, we do not regard them to be a big factor to consider. Based on the attacks we discover, and descriptions of attacks found in [21], we make an assumption that most DoS attacks will have smaller packet sizes than 100, and are most likely to have one packet per flow. Table 2.2 describes the alternative way of defining the anomalies.

Anomaly	Defined by packet size and number of flows
DoS	Small packet sizes, usually < 100 bytes. Often only one packet per flow.
Port Scans & **Port Sweeps**	Small packet sizes, often < 55 bytes. Often one packet per flow.

Table 2.2: Anomalies defined by packet size and the number of flows.

2.4 Detection and Identification Metrics

There is a vast range of metrics defined for evaluating the detection and identification performance in a detection system. The metrics used are based on the assumption that malicious activity (e.g. scan or DoS) is a subset of all the anomalous events that are detected. When identifying an anomalous event it will be classified as either

a true positive (TP) or a false negative (FN). The former means that the anomaly was due to a malicious activity, whereas the latter corresponds to a malicious event that did not trigger an alarm. Events that are not malicious are often referred to as *benign* activities. A benign event is either classified as a true negative (TN) or a false positive (FP). A TN is when a benign event do not trigger an alarm, whereas benign activity that generates an alarm is classified as a FP. An illustration of this concept is shown in Figure 2.3.

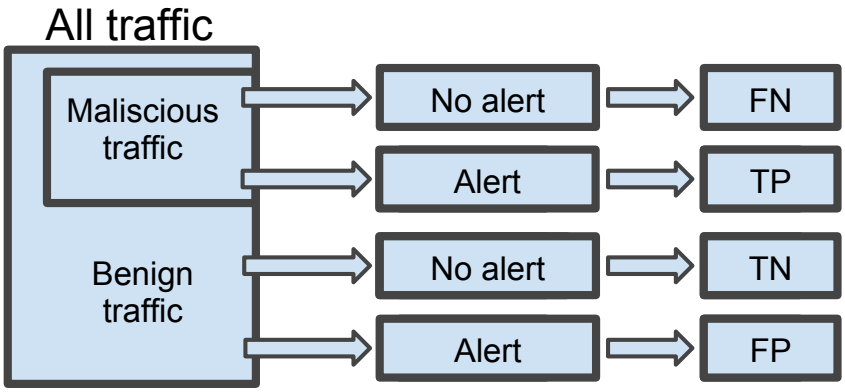

Figure 2.3: Illustration of the possible outcome of an anomaly identification process.

Detection and identification rate are two approaches in evaluating a detection system. Detection simply refers to the anomalies discovered by a detection system, whereas identification is the act of **verifying** whether a flagged anomaly is a true or false positive. To compare detection systems one typically computes the detection and identification rates. The detection rate is the number of detected anomalies by *one* system, divided by the union of the detected anomalies (i.e. one rate for each system). The identification rate for a specific system is the number of identified anomalies (TP+FP) divided by the number of identified anomalies + the number of unidentified anomalies.

The evaluation metrics used in this thesis is further explained in chapter 4.3.

2.5 Entropy-based Detection

Traffic feature distributions are highly dimensional objects, and it can prove challenging for classical time series techniques to extract useful information from such distributions. Entropy based metrics provide a more fine-grained insight into the distributions than volume based detection can provide, and gives a fairly compact representation of the traffic features. If a time series of entropy values (e.g. for source ports) are plotted in a graph, one can by visual inspection see the change in entropy, and unusual patterns in the traffic spectrum may be revealed. A. Lakhina et al. [18] shows that by looking at the changes in the distribution of traffic features, it is possible to discover a broad spectrum of anomalies. Compared to volume based techniques, entropy can also detect anomalies that do not introduce large changes in the amount of traffic [18] [21].

Entropy Formula

Let X denote a random variable representing the distribution of values of a specific traffic feature. Each X can take n outcomes $\{x_i : i = 1, .., n\}$. Entropy, a measure of uncertainty, is defined as [33]:

$$H(X) = -\sum_{i=1}^{N} p(x_i) log(p(x_i)) \tag{2.1}$$

To be able to quantitatively compare entropy values across time, one might follow the same pattern as suggested by Nychis[21], and compute the normalized entropy:

$$H_n(X) = H(X)/log(N_0) \tag{2.2}$$

Where $log(N_0)$ is the upper limit for the entropy, and is achieved if every element in

X appears exactly once.

Detection and Identification

An abrupt decrease in entropy values corresponds to a concentration of that particular feature's value in the traffic spectrum. On the other hand, an abrupt increase corresponds to a dispersion in the distribution of feature values. For a DDoS attack this would imply a dispersion in the distribution of source ports, and a concentration in both destination port and address (depending of the configuration of the DDoS).

The aforementioned behavior leads us to another feature of entropy-based detection, namely the ability to classify the anomalies by their respective entropy values. This is achieved by evaluating the entropy values of traffic features, and search for irregularities in the same time interval across the different distributions. For example, during time interval t, one can see an abrupt decrease in entropy values for both destination ports and destination IPs. Firstly, this indicates that there is as anomaly present, but one can also see that the anomaly has the same nature as a DoS attack. If we in the same time interval experience an increase in source IP entropy values, the sum of all observations strongly indicates that the anomaly is a DDoS attack.

2.5.1 Limitations

Even though entropy-based detection can identify and classify anomalies, the technique lacks the ability to output specific feature values. Moreover, if the time series of entropy values for each traffic feature suggest the presence of a DDoS attack, we are still not able to extract the exact feature values which cause the alarm.

Previous work on entropy-based detection [26] concludes with the fact that anomalous traffic (e.g. scans or DDoS) must occupy a certain ratio of the total traffic in

the current time interval in order to be detected. The ratio of a ”normal” scan in a large backbone network is typically lower than the suggested threshold and therefore entropy-based detection is not able to detect nor classify a small scan attack.

2.6 Histogram-based Detection

A histogram is a distribution of flows, bytes, or packets with respect to the different values of traffic features. The most commonly used traffic features encompass IP address, Autonomous System (AS) number and port, for both source and destination. The main idea behind this approach is that (i) histograms capture regular traffic patterns that reflect the behavior of a network; and (ii) during an anomaly the traffic patterns get distorted, hence it will be visible in the histograms.

Figure 2.4 illustrates two histograms of the destination and source port feature, based on data obtained during night-time on the 26th of January 2010. Each of the histograms represent the distribution of feature values in that particular time bin. The upper plot is the typical way of viewing it, one bar corresponding to one feature value (one-to-one). While in the lower plot, each bar corresponds to a number of different feature values. So instead of designating a bar to only one feature value, it can represent a range of values (e.g. one bar corresponds to port number 1-100) or even a combination of different features (e.g. a single port number with a range of IP addresses).

By generating histograms for each of the time bins, one will be able to see irregularities in the traffic pattern. Figure 2.5 is an overview of the destination port histograms taken from four consecutive days in January 2011. The time bin corresponds to the 0505-0510 (A.M.) interval, and we can observe the following: The histograms have more or less the same pattern, that is a set of ports that appear more frequent than the rest (i.e. [21,22,80,110,443,546-47]). However, in the upper right histogram one

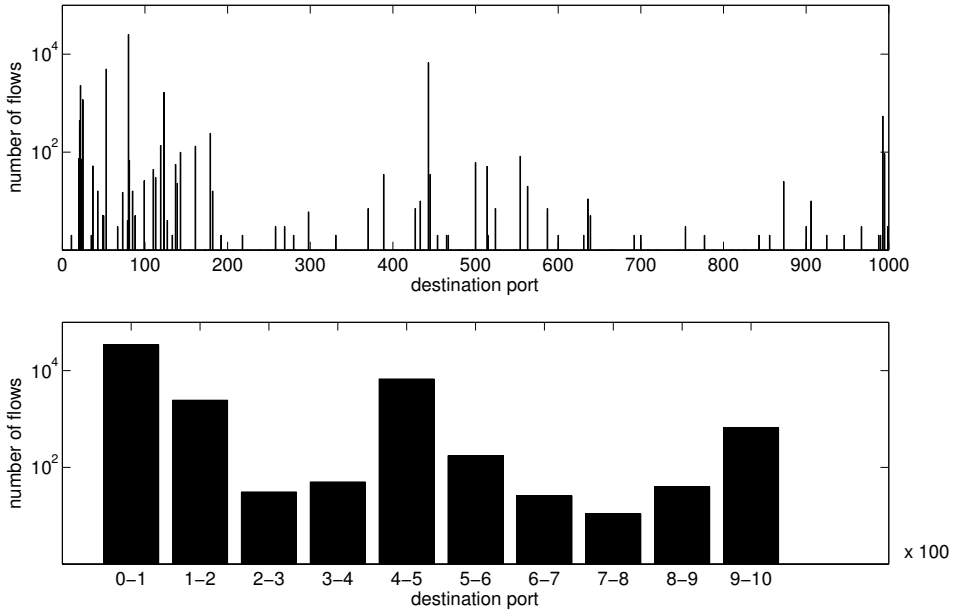

Figure 2.4: Histograms illustrating destination ports and their corresponding flows during a 5 minute interval. In the lower plot features are grouped for every 100th value.

can see that certain ports appear more frequently in the range from 600-800. The consequence is a distortion of the normal pattern and might be flagged as suspicious and detected depending on the detection technique.

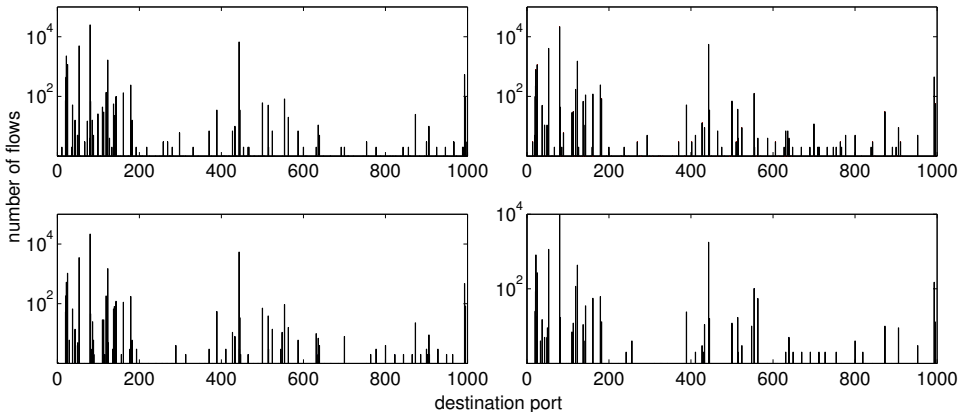

Figure 2.5: Histograms illustrating destination port with corresponding flows of the same time bin in five consecutive days

Histogram-based detection is a technique proposed by several authors in the past years for detecting different kinds of anomalies based on traffic features. Kind et. al. [17] propose a method that can be classified as semi-supervised, but can also be implemented as an unsupervised detection technique. The method contains the following four steps:

I. Select features and construct histograms.

II. Map into metric space.

III. Cluster and extract models.

IV. Classification

The first step consists of **selecting** the appropriate traffic features for further extraction and evaluation. The number of features and the features themselves decide to some extent the performance of the detection algorithm, both the time and resources utilized (i.e. memory usage and processing overhead) and the types of anomalies that will be detectable.

Table 2.3 gives a summary of the most commonly used features and the types of anomalies that are directly related to each of them. Generally, the more features one includes in a system, the more likely one is to detect and identify anomalies, this does however also increase complexity.

The next step is mapping training histograms for each feature into a **metric space**. The objective for such an approach is to (1) place similar histograms close to each other and (2) to keep dissimilar histograms apart. There are several distance techniques that can address such a problem, but is is desirable to utilize one that captures variance characteristics of a dataset. For example, a component t_1 that varies a lot should not be weighted equally to a component t_2 with lower variance, thus a change in t_1 should not be as suspicious as a change in t_2. [17] utilizes the *Mahalanobis distance* to detect patterns of common behaviour to quantify similarities between

histograms and PCA for dimensionality reduction. The latter is important in order to reduce the complexity of the derived models, hence maintaining the quality and the scalability of the classification.

Clustering is another step of the training process, and is needed in order to identify and model patterns of normal behavior. The problem lies within finding the clusters that best describes the data, and distinguishing between clusters that represents normal and anomalous data. The filtering of anomalies is done by removing clusters that correspond to a small fraction of the total observations. Kind et al. suggests to remove clusters that has fewer points than 5% of the total number of histograms. The final set of clusters models the normal network behavior and is refereed to as a *baseline*.

Finally, the network is monitored and for each traffic feature during an interval a vector is generated. Each vector is then compared to a model to see how the network behavior differs. If the network behavior falls inside one of the existing clusters, it is considered normal, and anomalous otherwise. An alarm is initiated if a vector falls more than 3σ (three standard-deviations) away from a cluster.

D. Brauckhoff et. al. [11] propose another system that relates to histogram-based detection. This can be classified as an unsupervised approach and is summarized in the following four steps:

I. Histogram cloning and detection.

II. Voting and meta-data generation

III. Flow pre-filtering

IV. Association rule (AR) mining

Histogram cloning is a promising technique applied to HD. The motivation behind it is to maintain multiple randomized histograms (of the same feature), hence it will obtain additional views of network traffic. This technique is realized in [11] by cre-

Feature	Possible scenarios
Src IP	◦ one-to-many addresses associated with more flows than usual, e.g. due to worms, botnet, or spoofing. ◦ outages that might remove certain servers, hosts, or ASs from a network.
Dst IP	◦ DoS attacks, both single and distributed creates a concentration of the addresses in use. ◦ dispersion in the destination addresses due to scans.
Src Port	◦ DoS attacks originating from a single port ◦ flow to a specific port reflected from an attack, e.g. scans for vulnerable ports.
Dst Port	◦ attacks in general to a specific port, e.g. network scans or worms.
Protocol number	◦ use of unsolicited transport protocols ◦ sudden increase in ICMP packets due to ICMP-based scans.
Packet size	◦ high density of small packet sizes due to DoS attacks or scans. ◦ packets whose size are incrementing by a certain size, e.g. SYN flooding
Flow duration	◦ abrupt changes or concentrations of specific duration patterns

Table 2.3: A list of traffic features and their impact on anomaly detection. (Source [17])

ating n histogram-based detectors corresponding to n different traffic features. For each of the n features there are m bins per time interval, and by applying a hash function to each of the clones, one makes sure that each feature value is placed randomly into one of the m bins. This differs from classical binning, which tends to place adjacent feature values (e.g. source ports) next to each other in a histogram. For **histogram detection**, KL distance (see Section 2.6.1) is computed between every newly created distribution and a reference distribution, more specifically the distribution from the previous measurement interval.

If a KL distance exceeds a given threshold ($3 \times \sigma$) the algorithm generates an alarm on the set of time bins (B_k) and the corresponding set of feature values (V_k) within

each timebin. **Meta-data generation** depends on a voting strategy between each histogram clone for a specific feature. If a clone generates an alarm on a histogram bin it will undergo an iterative process that removes suspicious flows until no alarm is generated. When the iteration completes the set of anomalous feature values, V_k is identified by maintaining a map between values and corresponding timebins. The voting scheme is introduced at this point, and if a sufficient amount of histogram clones generates an alarm on the same feature value it will be included in the meta-data for further **pre-filtering**.

D. Brauchoff et. al. apply the Apriori algorithm, authored by R. Argawal et. al. [6], as the final stage of their detection system (**AR mining**). Apriori iterates over a pre-filtered dataset containing traffic flows and outputs the most frequent item sets based on a minimum support parameter.

Section 2.6.1, 2.6.2 and 2.6.3 describe in detail some of the techniques used by [11], which are the techniques we have chosen to adopt in our histogram-based detection scheme.

2.6.1 Kullback-Leibler Distance

KL distance (or divergence) is a mathematical technique that measures the difference between two probability distributions Q and P [34]. The technique is widely applied to information theory with the purpose of measuring the expected number of additional bits required to represent samples from distribution P based on code from Q, rather than P. In HD, distribution P represents the reference distribution, which is typically the ”true” distribution of data (observations), whereas Q represents the distribution of the current time interval (i.e. a model or approximation of P).

The KL distance is defined as:

$$D_{KL}(P||Q) = -\sum_{i=1}^{m} p_i log(\frac{p_i}{q_i}), \quad where \quad D_{KL} \geq 0 \tag{2.3}$$

The KL distance is zero if both distributions are identical, while deviations in the distributions imply larger KL values. An event (e.g. a denial of service) that has an impact on the traffic pattern will be seen as a spike in the KL distance. If the event spans multiple time bins, a spike will be seen both at the beginning and the end of the total interval.

A KL time series for destination AS is shown in Figure 2.6, with data based on two successive days in March 2010. The graphs to the left show the KL distance for all 288 time bins during a day, while the graphs on the right illustrate the first difference.[4] The dashed line corresponds to the alarm threshold for the current time interval – an alarm is initiated for all KL values greater than the threshold.

Threshold Value

The threshold value T used in our work, similarly to the threshold in [9], is defined as:

$$T = 3 * \sigma_{KL}, \quad where \quad \sigma_{KL} = \sqrt{E[(X_{kl} - \mu_{kl})^2]} \tag{2.4}$$

The vector KL corresponds to the KL values during one day of measurement data, thus a new threshold is calculated for each day. The value is based on observations stating that the first difference is approximately normally distributed with zero mean and standard deviation σ_{KL}.

[4]First difference is the first answer to a multi part equation (i.e. first of many time bins)

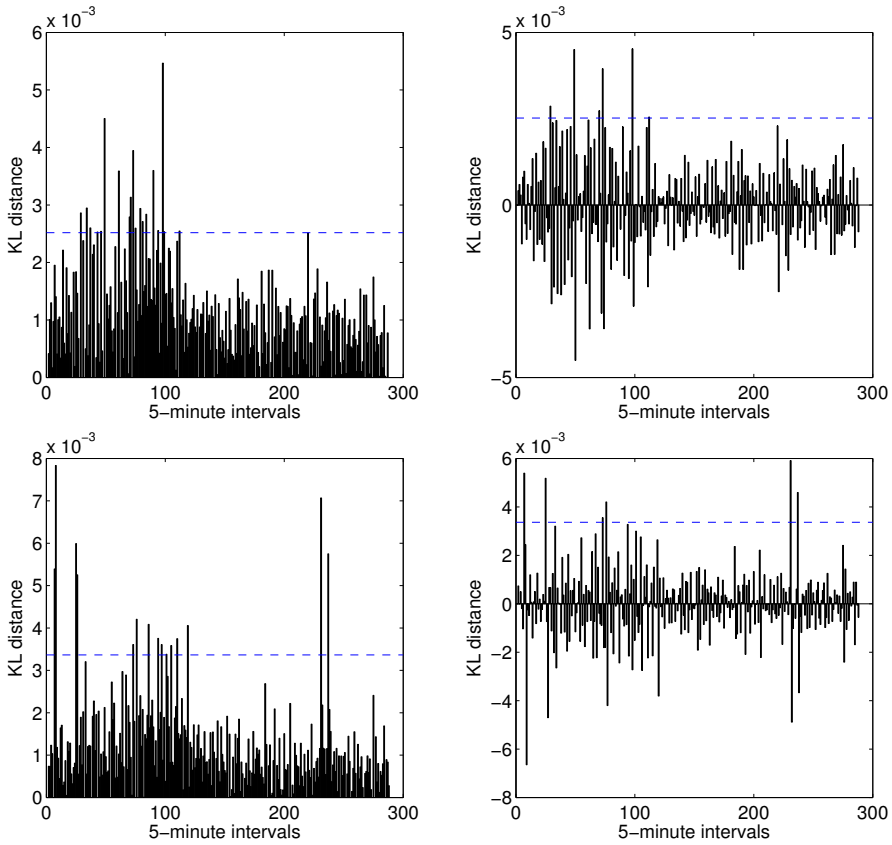

Figure 2.6: The Kullback-Leibler distance and first difference (with corresponding threshold seen as a dashed line) of two consecutive days in our dataset (March 25th and 26th).

2.6.2 Flow Pre-filtering

Before the application of AR mining one wants to filter the dataset based on the set of candidate flows. Each flow record in the candidate flows consists of at least two feature values, and at most four, e.g. [source AS, destination AS, source port, destination port]. The motivation behind this is to filter only the flows that match the union of a flow record in the set of all flows, hence generating a smaller dataset. A smaller dataset will lead to a decrease in the processing time of the remaining steps. Another, and even more important reason for pre-filtering is the impact on the detection rate. If one include all flows without any filtering, it will most likely

result in a higher rate of FP item sets generated by the association rule mining (see sec 2.6.3). Therefore, it is desirable to only apply the Apriori algorithm to a dataset containing a small amount of ”normal” flows.

The basic idea of the candidate flows and pre-filtering is depicted in Figure 2.7. With a set of candidate flows, the size and dimensionality of a pre-filtered dataset may potentially be several orders of magnitude smaller than the original dataset.

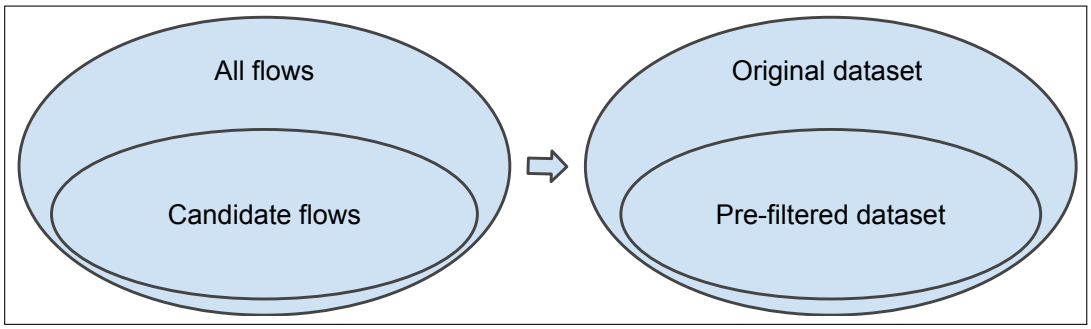

Figure 2.7: Illustration of the creation of candidate flows resulting in a smaller (pre-filtered) dataset.

2.6.3 Association Rule Mining

Association rule mining (also known as learning) is a data mining technique that grew rapidly in popularity partly due to an article authored by R. Agrawal et. al [5]. The technique was originally proposed to find relations between products in a large scale database consisting of customer transactions. The association rules could further on be employed as a tool to increase profit, in particular how to design coupons and what to put on sale. For example, a rule might say that 80% of customers purchasing beer also purchase potato chips. This particular fact may be exploited by placing both products next to each other on the shelves, and will most likely lead to an increase in sales.

Agrawal et al. defines the problem of AR mining as follows:

Let $I = i_1, i_2, ..., i_n$ be a set of n binary attributes called items. Let $D = t_1, t_2, ..., t_m$ be a set of transactions (i.e. the database). Each transaction D has a unique transaction ID and contains a subset of the items I. **A rule** is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$ [6].

The problem of discovering association rules can be decomposed into two subproblems; (1) find all combination of items (from now on called item sets) that has a support above a user specified minimum support parameter (referred to as *minsupport*) resulting in zero-to-many **large item sets**, and (2) derive association rules for the given large item sets.

The technique of AR mining is indeed applicable to the anomaly extraction problem. An anomaly will typically result in many flows that exhibit the same structural pattern, e.g. same source addresses, and destination ports, because they have a common root-cause (such as a DoS attack). In the anomaly extraction domain, each transaction D corresponds to a flow record, and the items $I = i_1, .., i_7$ to the following traffic features: {source IP, destination IP, source port, destination port, protocol, #packets, #bytes}.

Apriori

Apriori is an AR algorithm proposed by R. Agrawal et al. It has received attention due to its high performance, and according to [5] it outperforms other algorithms by factors ranging from 3 (on small problems) to approximately an order of magnitude for large problems.

Apriori requires as stated earlier the *minsupport* as an input parameter. If the parameter is selected to be too small, Apriori will potentially produce many item sets consisting of non-anomalous flows (FP). On the other hand, selecting a parameter

value that is too large, can cause Apriori to ignore anomalous flows, and lead to an increase of FNs.

In the first round, Apriori iterates over the dataset to identify the support for all candidate 1-item sets. At the end of the iteration, Apriori selects the 1-item sets with frequency above the *minsupport* parameter. The item sets created in round one is then used in round two to create 2-item sets. If Apriori makes at most h iterations and $l = 1, 2, ..., h$, then Apriori will stop when no item sets that contain $(l+1)$ items meet the *minsupport* requirement. In the final step, Apriori outputs all the unique item sets found to have a frequency $\geq$ *minsupport*.

A simple illustration of the iterative process of Apriori is depicted in Figure 2.8 and an example of the final output is given in Table 3.3 in chapter 3.

transactions: {1,2,3} { 2,3,4,5 } {1,2 } {2,3,4} {2,3} {3,4 }
minsupport: 3

1: 1-itemsets

item	sup
2	5
3	5
4	3

2: 2-itemsets

item	sup
{2,3}	4
{3,4}	3

3: 3-itemsets

item	sup
None	

Figure 2.8: Apriori algorithm applied to a small problem. Only possible 3-item set is $\{2, 3, 4\}$, but it has a fequency < 3 (*minsupport*).

2.6.4 Limitations

HD based detection encompass many techniques and different algorithms for data extraction, detection, and identification. These relatively resource intensive steps can result in a large computational overhead, and make it hard to obtain results for real-time detection systems. Thus, such a system must be highly optimized and equipped with strong computational power in order to offer scalability. Both with

respect to the size of the dataset, and the number of features included in the detection scheme.

Another limitation is the need for manual verification of the suspicious item sets created by Apriori. Whereas some item sets clearly indicate an anomaly, there are often item sets that do not appear suspicious to the same extent and further analysis is desirable. The analysis often involves manual inspection of the traffic, thus a very time consuming process.

2.7 SENATUS

SENATUS is a recently proposed detection system developed by A. Abdelkefi. Similarly to other detection techniques, SENATUS is divided into a number of steps to aid in the detection and identification process:

I. Election

II. Voting

III. Decision

The idea behind SENATUS stems from the concept of a senate, which means the assembly of the eldest and wisest members of the society. These members, called senators, make decisions (e.g. concerning legislation) which represent the opinion of the society.

In SENATUS, the senators correspond to the top-n feature values in the **election** phase. These feature values are elected based on prior knowledge of how malicious anomalies behave. In general, SENATUS applies two filters in order to capture the feature values that are most likely involved in either a scan or a DoS anomaly. Moreover, the data is either filtered to contain feature values with small packet sizes (for DoS anomalies) or feature values with small amounts of packets per flow and

small packet sizes (for scans).

The motivation behind selecting only a few feature values to represent the whole dataset is due to the *curse of dimensionality*. [4] finds that a feature histogram, when ordered, follows a power-law distribution, which implies a high compressibility of the histogram. While a dataset in reality needs all feature values f (where f>>n) to represent it completely, a relatively small amount can be included and still accurately represent the whole set. Figure 2.9 gives a clear picture of the dimensionality problem, and one can observe that a relatively small amount of traffic features carry most of the flows. The ratio of which an added feature value provides a better representation of the set, decreases significantly above a certain threshold. On the other hand, each selected feature value will proportionally add the same amount of dimensionality.

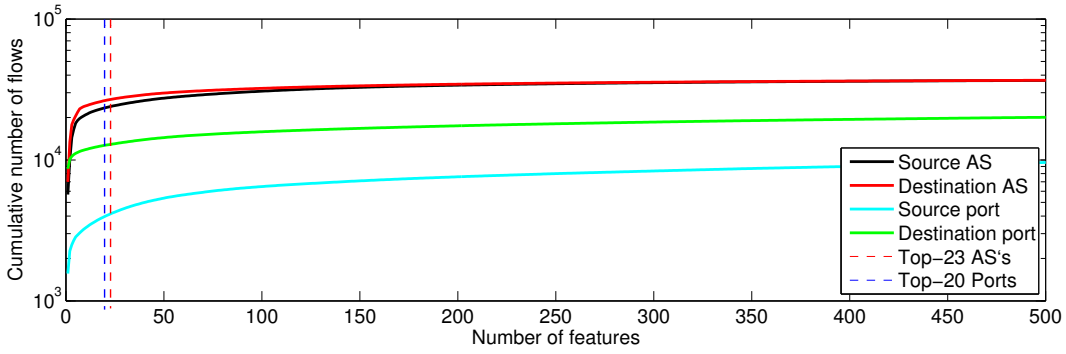

Figure 2.9: Illustration of the increase in number of flows with an corresponding increase in the number of feature values.

After selecting the top-n feature values (senators), SENATUS initiates the **voting** procedure to identify whether there is a problem or not. The voting is based on the detection of abrupt variations in the amount of flows for the top-n feature values per time bin. Abrupt variations are detected by applying *Principal Component Pursuit* (PCP), which is a robust version of the statistical technique; *PCA* (described later in this section).

When PCP has identified suspicious feature values they are further used to generate the set of candidate flows. All feature values are significant within each time bin when creating a candidate flow. For example, if only three out of the four traffic features are flagged as malicious, they will not be included in the set of candidate flows for that particular time bin.

Principal Component Persuit

PCA has proven to be very effective when applied in data analysis and for dimensionality reduction, however it has a known sensitivity issue occurring due to outliers in the data. In some scenarios these outliers may be removed prior to the application of PCA, but in anomaly detection, the outliers are not known beforehand. There have been several studies on the performance of PCA, and it is found that it suffers from inability to capture temporal correlations [10] (referred to as *poisoning*). However, several techniques have been proposed to counter the drawback of PCA, generally called *robust principal component analysis (RPCA)*.

Atef et at. [3] have shown that *Principal Component Pursuit* is resistant to the problem of poisoning. In general, the idea is to apply weights to the data objects based on their estimated relevancy. By applying the inexact version of the Augmented Lagrange Multiplier (ALM) solver, [3] also find that a satisfying performance is achieved when setting the tuning parameter to a fixed value – making the detection free of tuning parameters.

In SENATUS, the tuning parameter λ is set to:

$$\lambda = \frac{weight}{\sqrt{(max(n, m))}} \tag{2.5}$$

SENATUS' default parameter for $weight$ is 2.5, and n, m are the dimensions of each traffic feature matrix.

2.7.1 Limitations

SENATUS is optimized to detect anomalies that match certain criteria, where one of them is to only capture feature values with low overall bytes per packet (BPP) ($< 55-100$). This might exclude other types of anomalies, such as network outages. In other words, the detection scheme of SENATUS lack the same generic approach as adopted by both ED and HD.

SENATUS works on whole days of data, and in its first iteration it captures a feature value if it resides within the top-n set. Further more it has to capture all flows that relate to the captured feature values. This implies iteration over the same datasets two times, a process proved to be very time consuming, hence a drawback with respect to runtime.

3

Tools and Implementation

For data mining of NetFlow data and for aid with the different detection techniques, we utilize a set of Unix based tools, namely nfdump, Bourne-again shell (BASH) [13], Stream editor (SED) [16] and AWK [2]. Nmap is used for injecting various forms of scans. In addition to the aforementioned tools, we have also implemented data manipulation and calculation in Matlab® (v. R2010b, The MathWorks). This chapter contains an explanation of these tools and in what way they are utilized in our masters thesis. We will also give an detailed explanation of the workflow in the three detection systems – entropy-based detection, histogram-based detection+Apriori and SENATUS.

3.1 Nmap

Nmap, which stands for Network Mapper, is a free and open source tool for discovering available hosts on a network. It can list the status of a host's ports, operating system, running services, what firewalls are in use and other information that might be of use to both a systems administrator, and people with more malicious intents. What makes Nmap such a powerful tool, is the possibility of micro managing the scanner to behave exactly as you want it to. One example of this is how NTNU's network do not accept host discovery requests[1]. By using Nmap's -Pn argument, which turns of pinging (i.e. telling Nmap to not discover which hosts are online), we circumvent this problem and are still able to discover which ports are open on the IT systems connected to the network.

We will use a sample Nmap command to explain some of the most important arguments used and how they work:

```
nmap -T5 -A -Pn -v -p 100-6400 158.38.178.0/24
```

`-T5` is an argument designating the intensity of the scan, it ranges from `-T0` to `-T5`, with `-T5` being the most intensive. If a stealthy scan is of essence, lowering the intensity of the scan reduces the probability of being detected by an intrusion detection system. Scanning with a very high intensity can also lower the accuracy of the scan[1]. The reason for this is the possibility that packets are deleted by congestion control or collision detection. In the case of an UDP scan, deleted packets will confuse the scanner, as an absence of response leads Nmap to the conclusion that the port is open.

`-p 100-6400` instructs Nmap to only scan port 100 to 6400. By default Nmap scans the first 1000 ports of a host. This option will be used when doing port sweeps,

[1] If no host discovery options are given to Nmap, it will by default send an ICMP echo request, a TCP SYN packet to port 443 and a TCP ACK packet to port 80

where an attacker is interested in the status of certain high profile ports (e.g. port 1433 which is used by Microsoft SQL server).

The last argument in the command string is the IP scanned.

Table 3.1 explains in more detail the different commands and their effects.

Options	Description
-T<number>	Sets the intensity of the scan. These range from 0 (low) to 5 (high)
--min-rate <number>	Intensity is not lower than <number> per second (if physical equipment allows this)
--max-rate <number>	Intensity is no faster than <number> per second
-A	Turn on OS detection and traceroute
-v	Turn on verbose. Scan gives a more detailed report
-p	Sets the interval of ports Nmap will scan, -p 1000-6222 means that only port 1000 through 6222 will be scanned
-sU	Enables UDP scanning
-PE	Enable ICMP pinging for host discovery
-Pn	Turn off pinging (no host discovery)

Table 3.1: Nmap options and their descriptions.

3.2 Nfdump

Nfdump is a tool for extracting information from network data captured in NetFlow format by the NetFlow capture daemon – nfcapd. It supports a rich set of tools that enables the user to extract specific traffic data with fine-grained precision. We use nfdump in our scripts to extract data with any particular property we might be looking for. Table 3.2 contains an overview of the most frequently used commands in our scripts. Nfdump has four fixed output formats, and the default output format is given in the following format:

```
Date Duration Src IP:Port  Dst IP:Port Packets Bytes Flows
```

Options	Description
-R **</dir/**first-file:lastfile>	Reads data from a sequence of files found in the directory **dir**
-o "fmt:**<format>**"	Specify output format by element tags (%fl for flows)
-a	Aggregation on srcip, dstip, srcport and dstport.
-A **<v9 field>**	Aggregation on specified **v9 fields** (proto for IP protocol)
'<filter>'	Filter NetFlow data based on feature values

Table 3.2: Nfdump options and corresponding description.

It is possible to specify and customize any desired output format, and depending on what traffic feature we are currently analyzing, the output format will differ.

As an example; when calculating source address entropy, we will need the source address and the number of flows. A complete nfdump command to extract these data would be in the form of:

```
nfdump -r /any/dir -o "fmt:  %sa %fl %pkt"
```

In certain cases we can use nfdump manually as an anomaly detector, and in some cases even as an anomaly identifier. By visually inspecting the plethora of information contained in a five minute time bin, it is possible – all though time consuming – to discover anomalies like scans or DDoS attacks. Without narrowing down the results from nfdump, the data returned is likely to look like Figure 3.1. As can be seen, there are many different sources sending data with differing packet sizes to various destinations, and it is difficult to draw any conclusions based on observations of this data.

When creating an anomaly spectrum as seen in Figures 2.1 and 2.2, we need to find the number of anomalous flows hidden in the traffic. In the case of HD and SENATUS, potentially anomalous time bins are flagged with source and destination AS and port, but without a specific IP address. Based on the information we have

from HD or SENATUS (e.g. Source and destination AS), we can create commands to only extract traffic going between these addresses.

```
nfdump -r
/data/netflow/oslo_gw/2010/10/04/nfcapd.201010040445
'(src as 1299 or src as 11427 or src as 14929 or src as
31375 or src as 23974 or src as 47205 or src as 5650)
and dst as 224' | more
```

This command tells nfdump to extract data from the five minute interval during 04:45 - 04:50, 4th October 2010 containing packets with one of the source AS' (e.g. source as 1299 or 11427) headed to destination AS 224.

Figures 3.2 and 3.3 show the observable anomaly when we narrow down the results with the aforementioned command. As can be seen, several different source IPs send small packets in large numbers to the same host – a typical case in DDoS attacks.

This method of detecting anomalies with nfdump can be utilized for deciding whether a flagged anomaly is a false or true positive.

3.3 Unix Tools

Matlab is a high-level computing language and an interactive system designed to solve mathematical problems in an efficient manner, e.g. matrix and vector computation. It includes a vast library of highly optimized mathematical functions, and is comparable to c, c++ and the like in terms of performance. In our work, Matlab is used for matrix and vector manipulation, candidate flow creation, as well as the application of mathematical detection techniques (KL and RPCA).

BASH is a command interpreter and a high-level programming language, in a GNU

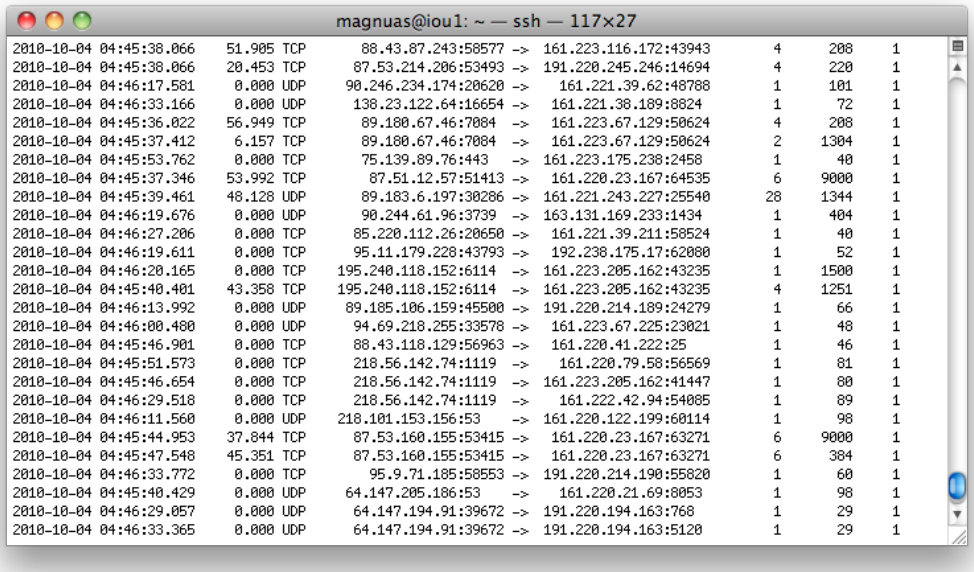

Figure 3.1: A typical view of a random time bin using nfdump

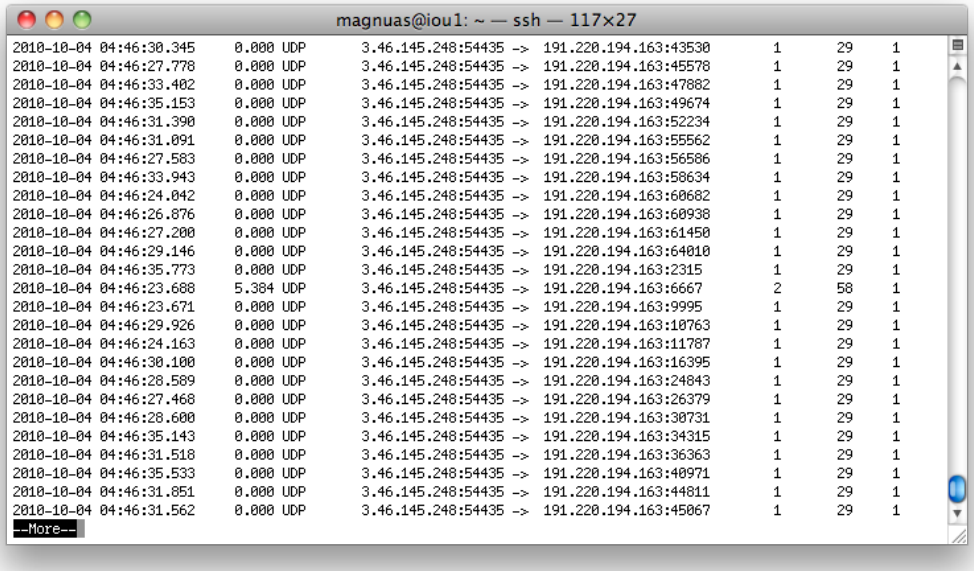

Figure 3.2: Possibly a DDoS attack.

operating system[2]. It is the user interface to a rich set of GNU utilities, such as SED,

[2]In our case a GNU/Linux operating system which is a combination of both.

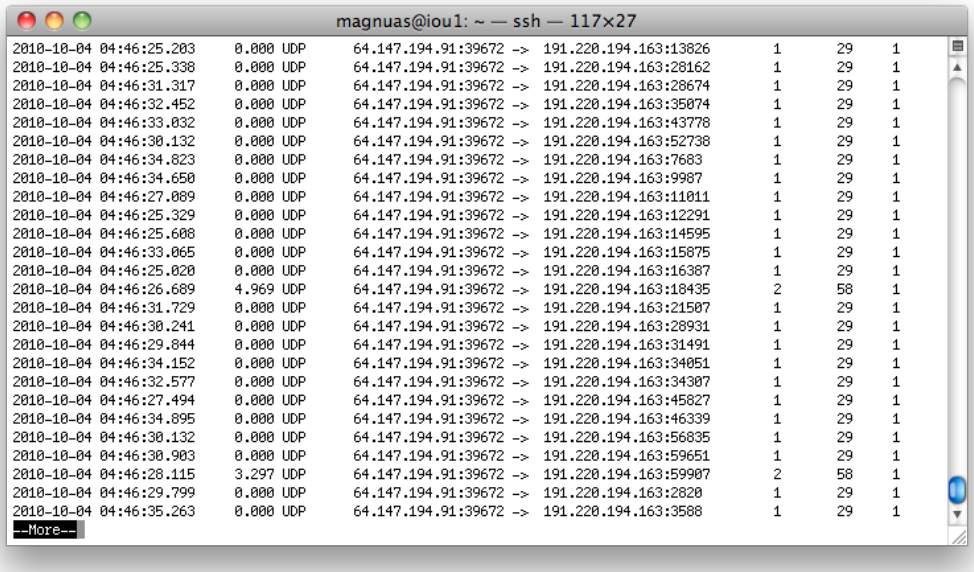

Figure 3.3: Possibly a DDoS attack.

AWK. The shell is simply a macro processor able to execute predefined commands, and like any other language it has variables and control flow commands (e.g. iteration using *while* and *if* statements). We utilize the shell primarily non-interactive, which means that the shell executes commands from a file. The files (also called shell scripts) in our work are listed in appendix A. Basically, our shell scripts is called with zero-to-many parameters, depending on the detection scheme, and executes commands in a top-down sequential manner. BASH is utilized in all three detection techniques, and primarily works as the framework for data extraction.

SED is a non-interactive feature, frequently used inside our shell scripts. SEDs main feature is to perform basic text transformations on an input stream, normally from standard input (stdin), but also from one or multiple files. SED applies transformation to each line, writes to a buffer and sends the output to standart output (stdout). We utilize it in a piped structure (input stream from stdin) with the intention to re-

move unwanted information produced by nfdump (data extraction).[3]

AWK is a programming language.[4] It is developed as a text-processing language – it has simple syntax to match lines of patterns, separate out the fields and operate on them. We utilize AWK both to aid the entropy and histogram based detection. In ED the actual entropy calculation and filtering/thinning is performed by AWK, while HD uses it for formatting prior to the application of Apriori.

3.4 Implementation

The implementation has an important role when evaluating the performance of the different techniques and also when considering resource usage and execution time. However, our priority is centered around the actual detection (all three techniques) and identification (Histogram-based detection and SENATUS), and does not consider other performance metrics to the same extent. Considering this, Sections 3.4.1, 3.4.2 and 3.4.3 describe the implementation of the three different techniques in detail. In entropy and histogram-based detection detection, parts of the implementation are based on related works, while most of it is developed during the past four months of our work. SENATUS which is the last detection scheme, is fully developed by A. Abdelkefi; supervisor and Ph.d. candidate.

3.4.1 Histogram-based Detection

The first step is depicted in figure 3.4 and consists of extracting desirable data from NetFlow records. In this case we want to extract statistics based on different traffic features, namely source AS/port and destination AS/port. Further more, the statistics is piped to SED and AWK for removal of unwanted information. The sole

[3] Unwanted information is typically auto-generated summarization of flows and redundant flow information.

[4] AWK was developed in 1977 by Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

reason for the filtering is to decrease the execution time and not having to deal with unnecessary data at a later stage. The output is simply one file for each feature for every time bin during one day. All in all it produces $288 \times 4 = 1152$ files per day, and increases linearly with the number of selected days from our dataset (Section 4).

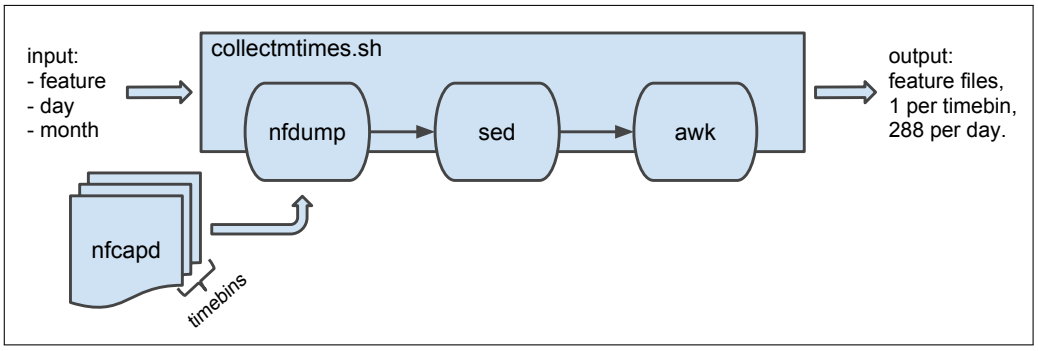

Figure 3.4: Netflow data extracted for further analysis in HD.

The second step in the work flow we apply the actual anomaly detection algorithm, Histogram-based detection (Section 2.6). The scripts reads the output from the first step and creates a TFD for each file. Further on, the KL-divergence between the distributions for each of the features is calculated. The KL-divergence is then evaluated (see Listing 3.1), and based on a threshold the current time bin is either flagged as anomalous or discarded.[5] The next step is to locate anomalous feature values that resides within each of the anomalous time bins. For example, if time bin 100 is flagged as anomalous, we need to flag the feature(s) that causes the time bin to get flagged. When this is done we end up with two vectors as the output, one for the anomalous time bins and one for the suspicious features within the anomalous time bins.

The third step processes the previous output to; (1) create the union of time bins flagged by each of the feature values and (2) to create the suspicious flows based on all combinations of flagged traffic features. For example, if the set [224,64514] is flagged as suspicious source and destination ASs and the set [6667,443] is flagged

[5]For an explanation of the threshold see equation 2.4

Listing 3.1: A code snippet to illustrate how timebins are flagged as anomalous.

```
k=1
for i=1:288              % For every timebin/day
 if (kl(i)>3*std(kl)) % Check if the kl divergence betweeen
    → two
                         % distributions is > 3 x standard
                            → deviations
                         %  of kl(:)
  anomalousbin(k)=i;   % If it is, flag the timebin as
     → anomalous.
  k=k+1;                 % Go to the next timebin..
 end
end
```

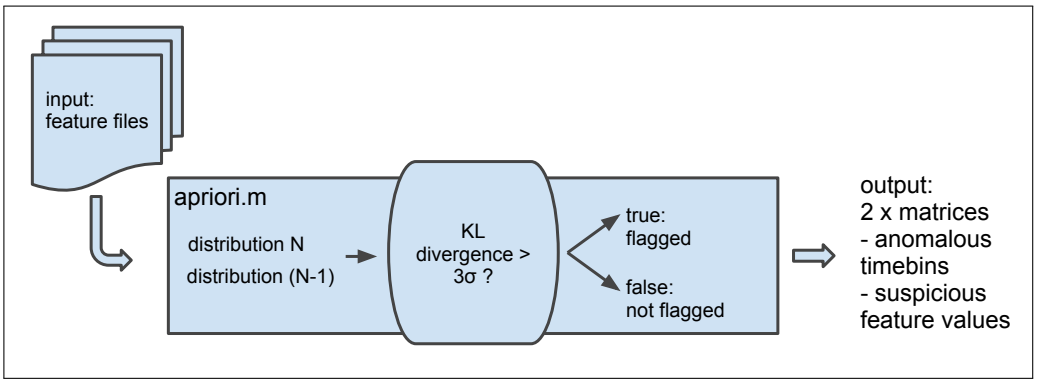

Figure 3.5: Application of KL in HD to detect anomalies in the dataset.

as suspicious source and destination port, then the 4-tuple [224,64514,6667,443] makes up a flow. These 4-tuples are further used in the next step to aid the data mining process. An overview of this step can be seen in Figure 3.6

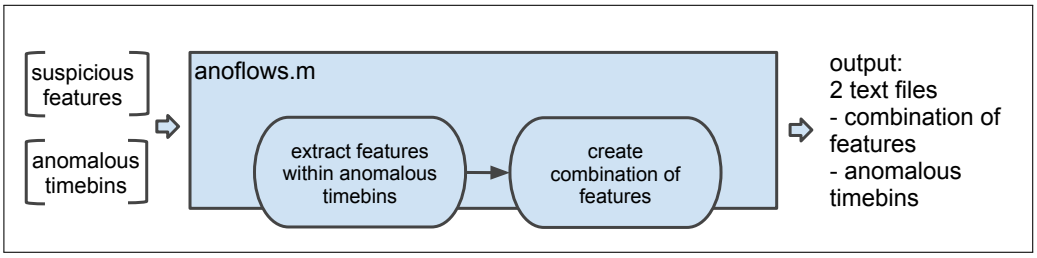

Figure 3.6: Candidate flow creation in HD: Combining the suspicious features into canditate flows for further filtering and data mining.

In the fourth step we apply the most important filtering, which is generally called flow pre-filtering. The combination of flows obtained in step tree is used to filter the NetFlow data. The filtering is only applied to the time bins which is already flagged as suspicious.[6] The step is depicted in Figure 3.7, and a detailed explanation and motivation behind the fourth step is described in Section 2.6.3.

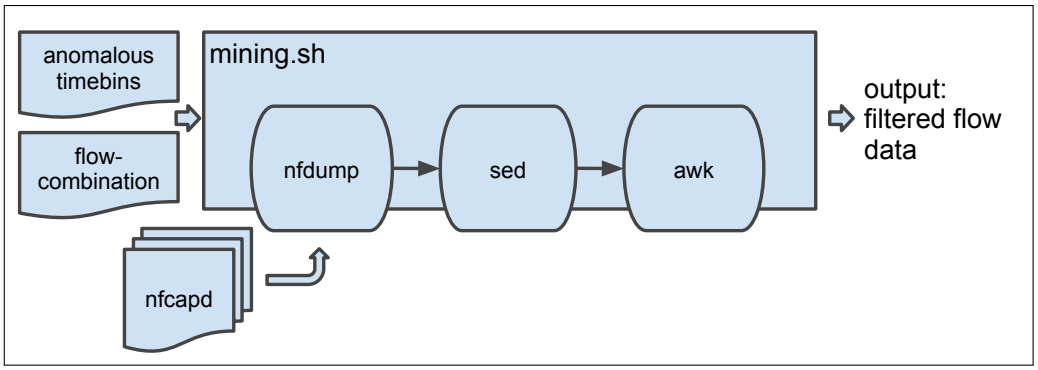

Figure 3.7: Filtering of NetFlow data based on the combination of suspicious features and time bins in HD.

The fifth and final step in our HD system is the application of Apriori [6] (Figure 3.8). It iterates over a preformatted set of candidate flows (the output from step four) and locates item sets based on their frequency (detailed explanation in Section 2.6.3). An example of the output is listed in Table 3.3 – each column corresponds to the traffic features and each row corresponds to an item set, where the first column is the weight of each item set.

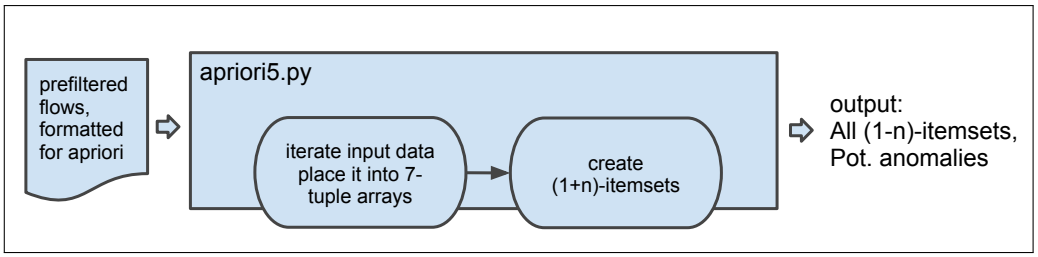

Figure 3.8: Application of Apriori to create item sets based on pre-filtered and formatted flows as input data.

[6] In our case the time bins that is marked by both entropy and histogram-based detection.

	srcAS	dstAS	srcPort	dstPort	#packets	#bytes	duration	support
1	*	*	*	*	2	*	0.100	15998
2	224	*	*	*	1	*	0.300	12297
-	-	-	-	-	-	-	-	-
2	*	*	80	*	1	*	0.100	11292
5	*	224	1024	6667	1	48	0.200	12492
5	*	224	3072	6667	1	48	0.100	12382

Table 3.3: Frequent item sets computed with Apriori algorithm (minimum support: 10000, #flows: 197000)

3.4.2 SENATUS

SENATUS is implemented mainly by utilizing the same tools as described in Section 3, with a small part implemented in Perl [22].[7] Perl is a programming language ideal for text manipulation, it is widely used in resource intensive applications such as bio-informatics, because of its ability to handle large data-sets [29].

SENATUS is applied one day at a time, and the input parameters are hard coded, thus different instances of the same code base are executed depending on the anomaly one wants to detect (that is scan or DoS anomalies.). The same set of traffic features are utilized both for scan and DoS anomalies, namely source AS and port, and destination AS and port.

The first step in SENATUS involves anomaly selection and data extraction based on feature values and is depicted in Figure 3.9. If the code base for scans is executed, the script produces a summary of the top-n feature values (with BPP < 55 and #packets-per-flow < 4) during a specific day.[8] The summary is formatted to only contain a sorted list of the specific feature values. SENATUS initiates further extraction and creates several vectors containing the total amount of flows related to the feature values within each interval (time bin). For example, the first element in vector $\overrightarrow{\mathbf{dp}}_{80}$ corresponds to the number of flows with **d**estination **p**ort 80 within the first time bin

[7] Perl is authored in 1987 by Larry Wall.

[8] Scans typically generate packets with a size less than 55 BPP

during a specific day. There are a total of 288 time bins per day, hence the number of elements in each feature vector is 288.

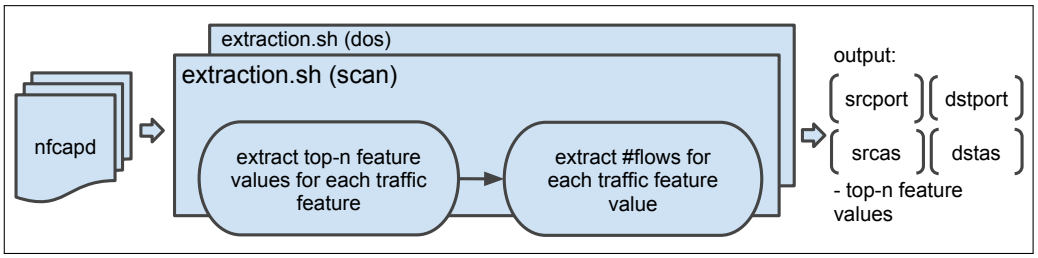

Figure 3.9: Anomaly selection and data extraction in SENATUS.

The final output of the feature value extraction is 4 matrices per day – one for each traffic feature. Each matrix consists of all the feature value vectors for the respective traffic features, e.g the matrix for destination ports during a day is $\mathbf{dstport}_{day1} = [\mathbf{dp}_{80}, \mathbf{dp}_{110}, ...]$ (see Figure 3.10). In addition, the extraction step outputs the summary of the most frequent (top-n) unique feature values for each traffic feature.

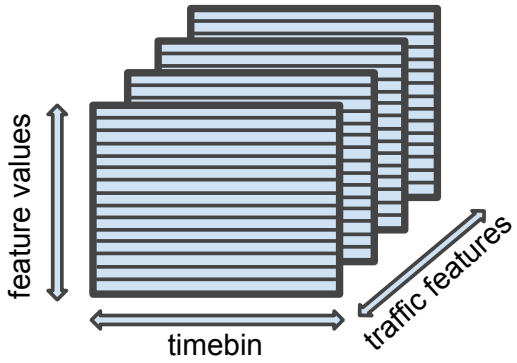

Figure 3.10: Final output of the initial extraction of SENATUS.

The output from the first step is then processed by a matlab script, which applies RPCA to the four matrices (figure 3.11). The index of each entry (i_x and i_y) in the output of RPCA corresponds to a traffic feature value (i_x) and a time bin (i_y) respectively. If the entry value $i_{x,y}$ is greater than zero, the corresponding feature value and time bin will be kept for further processing. After this the script initiates the voting between flagged feature values within each time bin. If all traffic features

have flagged a set of feature values for a specific time bin, they will be used to generate combinations of suspicious flows, referred to as candidate flow.

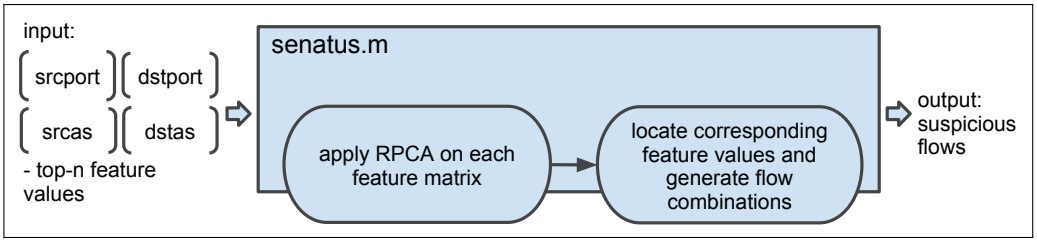

Figure 3.11: Application of RPCA and generation of suspicious flows in SENATUS.

In the final step, SENATUS will verify the various candidate flows by applying them as a filter in nfdump and attempt to extract NetFlow data for each time bin (figure 3.12). If none of these filters produce any matches, there are no anomalies found in that time bin. The output of SENATUS consist of sets with suspicious feature values, e.g. $[4314, 224, 6000, 1433, 180]$ which corresponds to [Source AS, Destination AS, Source Port, Destination Port, Time bin].

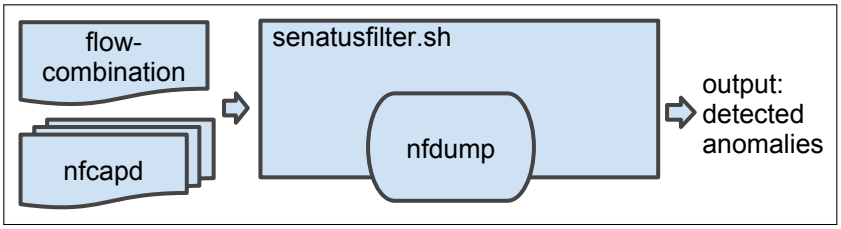

Figure 3.12: Final extraction and verification in SENATUS - based on the combination of suspicious flows.

3.4.3 Entropy-based Detection

The work on entropy-based detection was conducted during [26], and the same work flow is adopted in the this paper. The implementation consists of several scripts, one for each of the five traffic features studied.[9]

[9]Destination IP/port, source IP/port and the flow size distribution (FSD).

The basic function of the script is depicted in Figure 3.13. Firstly, the detection is initiated with three input parameters, namely day, month and the feature. Based on the input, the script extracts desirable information from NetFlow records using nfdump. The output from nfdump is then formatted and filtered for unwanted information before we, in the last step, apply thinning and compute the normalized entropy.

Thinning works by specifying a feature value (e.g. destination IP as 123.123.123.123) and a threshold (i.e. a number between 0 and 1). The feature value decides which flow records to keep. This is useful when filtering on anomalies where none of the feature values are known prior to the entropy-based detection. A threshold value decides at which rate to discard the non-anomalous flow records, which in the end implies that the ratio between non-anomalous and anomalous flow records increases with an increase in the threshold. However, no thinning is applied when empirically comparing entropy against SENATUS and histogram based detection. The reason is simply that malicious feature values are not known prior to the detection – it is a tool to help entropy detect what it previously could not.

Finally, the adjacent entropy values, one for each NetFlow time bin, are evaluated to verify if the anomaly had an impact on the traffic pattern or not. An impact can either be seen manually by making a plot of entropy time series or by identifying clear deviations in the entropy values.

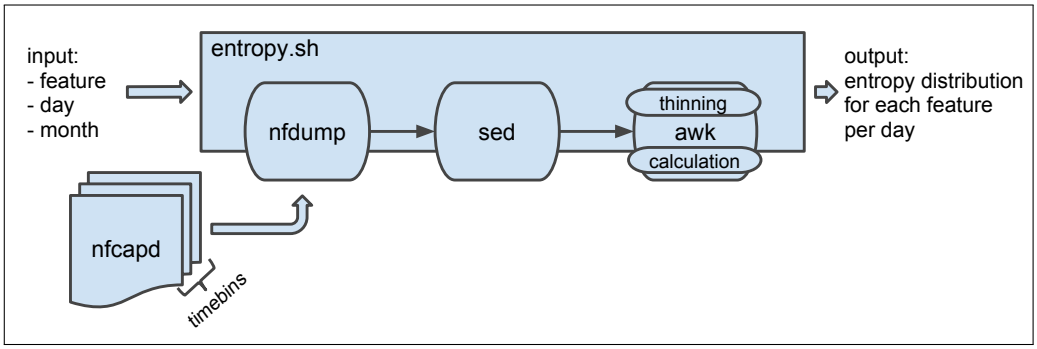

Figure 3.13: Calculation of Entropy based on feature distributions.

4

Ground Truth

Evaluating anomaly detectors are by the nature of their mechanics no easy task. Their job is to extract anomalous traffic out of an enormous amount of normal traffic – which is like finding a needle in a haystack. An important decision to make is what approach to take when evaluating SENATUS. According to [25], there are mainly two approaches the research community view as viable alternatives. One is based on root cause analysis, where each anomaly flagged is inspected to pinpoint its cause [21]. The second approach is injecting artificial anomalies into the network we are studying. There are a couple of advantages with the second method; the first being the ability to test the anomaly detector for different types of anomalies

of your choosing with varying intensities and durations. The second advantage is the ability to accurately find false negatives (see chapter 2.4). If you do not inject anomalies, it is difficult to know if there are potential anomalies you have not found. The drawback is the need for powerful equipment to inject them. As shown in the previous work [26], the level needed to be surpassed for an anomaly detector based on entropy metrics is high (see Table 4.2). There is also the fact that most network administrators do not view it kindly that one inject destructive anomalies in their networks. We will in this thesis set the ground rules for and attempt a third method for evaluating anomaly detectors. This method is based on the intuition that the union of the time bins flagged by all detectors, gives a fair representation of the number of anomalies in the set of time bins. This method will be explained in Section 4.3.

Dataset

The methodologies described in Section 4.2 and 4.3 are based on the dataset from March, November and December. While we have analyzed the whole dataset, we will present detailed analysis of 11 days of data captured at the Oslo gateway in Uninett's backbone network. The selection of days is as follows:

November 2010	1st,2nd,12th & 14th
December 2010	12th,18th & 21th
March 2011	1st, 25th,26th & 28th

The dataset consists of nearly two months of sampled NetFlow data captured in UNINETT backbone network and are thus very large in terms of data volume and multidimensional in terms of traffic features that can be measured and evaluated. The traffic crossing the Oslo Gateway in UNINETT on an average day consists of roughly 800 gigabyte (GB) data and 200 million flows. The days in our data set range from the 1st-15th of November 2010, 12th-21th of December 2010 and the

1st-31th of March 2011.

4.1 Injection Based Ground Truth

Our initial attempt at constructing a ground truth was based on the approach of injecting anomalous traffic into the network. With the approval of the network administrators at NTNU, we were authorized to scan their network with Nmap. Table 4.1 contains a sample of the injected attacks[1]. We use these attacks to create Figure 4.1, which shows the effect the port sweeps and port scans have on the traffic spectrum. The reason we can not see port sweeps with the -T1 intensity is in all likelihood due to the fact that NetFlow samples packets with a rate of 1 packet per 1000. In essence this means that the rate is on average less than 1000 packets per five minute interval for the -T1 intensity. As can be seen, the percentage of anomalous traffic is not high enough to be detected by entropy based metrics.

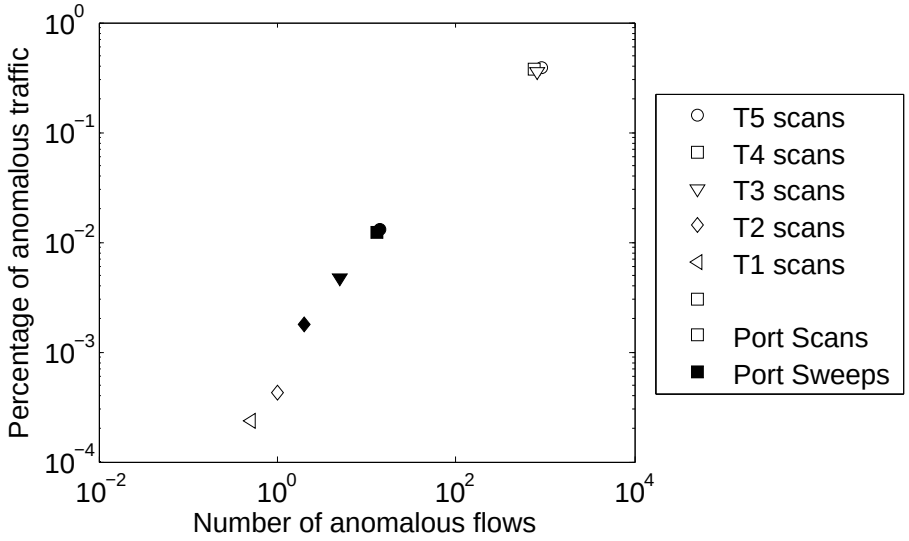

Figure 4.1: The percentage of anomalous flows set up against the number of anomalous flows

[1]See the appendix for an exhaustive list of our injected attacks

Date & time	Command
Port Sweeps	
June 23rd, 2010	
20:33	sudo nmap -Pn -T5 129.241.0.0/16 -p 1433
20:50	sudo nmap -Pn -T4 129.241.0.0/16 -p 1433
21:20	sudo nmap -Pn -T3 129.241.0.0/16 -p 1433
22:00	sudo nmap -Pn -T2 129.241.0.0/16 -p 1433
22:37	sudo nmap -Pn -T1 129.241.0.0/16 -p 1433
Port Scans	
March 28th, 2011	
15:05	nmap -T5 -Pn 129.241.50.0-254 -p 1-50000
May 16th, 2011	
11:36	nmap -T4 -Pn 129.241.50.0-254 -p 1-50000
14:06	nmap -T3 -Pn 129.241.50.0-254 -p 1-50000
May 19th, 2011	
14:00	nmap -T2 -Pn 129.241.50.0-254 -p 1-50000
May 20th, 2011	
13:52	nmap -T1 -Pn 129.241.50.0-254 -p 1-50000

Table 4.1: A sample of scans made with Nmap

Running SENATUS on the days with injected attacks did not provide adequate results either. By studying the flagged anomalies reported by SENATUS with Nfdump (i.e. by looking for the source IP of the computer doing the scans), we found that none of the anomalies found originated from our injected attacks. After discovering that injecting attacks did not prove to be a viable option for us, we had to use another approach.

Anomaly		Ratio
Port Scan	$\approx$	7%
DDoS	$\approx$	8%
Port Sweep	$\approx$	15%

Table 4.2: The intensity of anomalous traffic needed to be detected by entropy metrics

4.2 Analysis of Root Cause Based Ground Truth

SENATUS

To gain precise knowledge of the number of true and false positives, we manually inspect each of the time bins SENATUS flag in the dataset. The manual inspection is done with Nfdump. Table 4.3 shows the output from SENATUS of a single flagged time bin.

Source AS	4134 CHINANET-BACKBONE
Source Port	6000
Destination AS	64536
Destination Port	1433
Time bin	33
Hour	2
Minutes	40

Table 4.3: The output of SENATUS

The time bin from Figure 4.3 is an example of an identifiable anomaly. Source As,

source port and destination port all give clues pointing in the direction of malicious activity. The first thing to notice is the source AS which is Chinese. A large percentage of the traffic coming from China to Norway is port scans or port sweeps. The source port gives a fair indication of what type of anomaly we are dealing with. Out of 106 port scans and port sweeps found in our dataset, 55 had the source port set as 6000. The final clue is the destination port, which is set to 1433. This strongly indicates that the packets are sent from a host infected with the MS-SQL worm [18].

One should note that although SENATUS outputs 4 traffic features, it is possible that one of them, e.g. the destination port in the case of a port scan, varies. If this is the case, the anomaly in Table 4.3 could in reality be a port scan. To verify the anomaly, we manually inspect it with Nfdump; the results are shown in Figure 4.2. The results from Nfdump show that the discovered anomaly is a port sweep done by a single host. A single IP is sending a large number or packets to different hosts, all with the destination port set as 1433. We also note that every packet has a size of 40 bytes per packet, and only one packet per flow – both signs of a port scan or port sweep. To verify that we are not dealing with a port scan, we set Nfdump to show all packets coming from source AS 4134 and source port 6000, but with an undefined destination port. If the results are the same, we are dealing with a port sweep.

```
2010-11-02 02:42:59.759     0.000 TCP     122.154.222.68:6000  ->   161.222.32.196:1433        1       40     1
2010-11-02 02:42:54.626     0.000 TCP     122.154.222.68:6000  ->   161.223.226.41:1433        1       40     1
2010-11-02 02:42:59.726     0.000 TCP     122.154.222.68:6000  ->    161.222.36.46:1433        1       40     1
2010-11-02 02:42:54.664     0.000 TCP     122.154.222.68:6000  ->  161.223.229.205:1433        1       40     1
2010-11-02 02:42:59.709     0.000 TCP     122.154.222.68:6000  ->    161.222.38.49:1433        1       40     1
2010-11-02 02:42:59.709     0.000 TCP     122.154.222.68:6000  ->    161.222.36.32:1433        1       40     1
2010-11-02 02:42:59.695     0.000 TCP     122.154.222.68:6000  ->    161.222.37.18:1433        1       40     1
2010-11-02 02:42:54.647     0.000 TCP     122.154.222.68:6000  ->   161.223.226.21:1433        1       40     1
2010-11-02 02:42:59.709     0.000 TCP     122.154.222.68:6000  ->     161.222.38.5:1433        1       40     1
2010-11-02 02:42:59.770     0.000 TCP     122.154.222.68:6000  ->   161.222.33.251:1433        1       40     1
2010-11-02 02:42:59.756     0.000 TCP     122.154.222.68:6000  ->   161.222.33.255:1433        1       40     1
Summary: total flows: 42, total bytes: 1680, total packets: 42, avg bps: 2612, avg pps: 8, avg bpp: 40
Time window: 2010-11-02 02:42:54 - 2010-11-02 02:42:59
Total flows processed: 436561, Blocks skipped: 0, Bytes read: 22701508
Sys: 0.088s flows/second: 4960695.0  Wall: 0.162s flows/second: 2686149.0
```

Figure 4.2: The anomaly from Table 4.3 viewed in Nfdump

HD + Apriori

The methodology for identifying anomalies in HD + Apriori is very similar to that of SENATUS. The main difference is understanding the output of each method. Table 4.4 shows the output of HD + Apriori, which shows a number of item sets contained in one day. We already know which time bins we ran HD + Apriori on, so we do not need this information from the output. By plotting the information from each line of the item set in Nfdump, we find that every row represents a DoS attack against a single IP contained in source AS 224. This combined effort by several hosts sending an enormous amount of packets to a single host means we have found a DDoS.

Item Sets	SrcAS	DstAS	SrcPort	DstPort	Pkts. per flow	Bytes per pkt
5	23974	224	50720	*	1	29
5	47205	224	51771	*	1	29
5	1299	224	47652	*	1	29
5	14929	224	41463	*	1	29
5	31375	224	54763	*	1	29

Table 4.4: The output of HD + Apriori

4.3 Comparison Based Ground Truth

As a third approach in evaluating the anomaly detectors, we will attempt to find the detection rate for each method, based on what the other methods detect. The ground truth we will construct is based on the idea that other anomaly detectors will flag time bins as anomalous, where SENATUS might not. E.g. from SENATUS' viewpoint, the time bins that both Entropy and HD detect that SENATUS does not detect are false negatives. This is a fair assumption to make, since no single anomaly detector is perfect, but a time bin flagged by two detectors is likely to contain an anomaly. For HD, we have an exception if all the four traffic features are flagged as suspicious. In this case, we deem it likely that the time bin contains an anomaly, even though no other anomaly detector has flagged it.

We define the rules for marking a time bin as detected as follows:

- For histogram-based detection, mark time bin as detected if
  I. flagged by all traffic features
  II. also detected by Entropy-based detection
  III. also detected by SENATUS
- For entropy-based detection, mark time bin as detected if
  I. also detected by Histogram-based detection
  II. also detected by SENATUS
- For SENATUS, mark time bin as detected if
  I. the time bin is, by manual inspection, found to be anomalous

To evaluate the detection rate DR for each detection scheme let d_x where $x \in \{\text{SENATUS, HD, ED}\}$ represent the number of detected anomalies for a system, then for each system the detection rate DR_x can be calculated as follows:

$$DR_x = \frac{d_x}{(d_s \cup d_h \cup d_e)} \quad \text{where } x = \begin{cases} s, & \text{for SENATUS} \\ h, & \text{for HD} \\ e, & \text{for ED} \end{cases} \tag{4.1}$$

5

Results

This chapter will present and evaluate the results obtained based on the different approaches described in Chapter 4. Detection rates are solely evaluated based on comparison between the three different detection schemes, while evaluation of identification rates only focus on SENATUS and HD + Apriori (for simplicity now referred to as HD).

5.1 Results From Root Cause Based Ground Truth

Identifying the root cause can only be done in SENATUS or HD. The implementation of ED lacks the ability to keep a mapping between anomalous entropy values and their respective feature values.

According to a Gartner report from 2006 [23], root cause analysis takes on average up to an hour per flagged alert. In our experience, this holds true for HD if the number of item sets are less than three. In the case of SENATUS, we see drastic improvements in the time needed to identify an anomaly. On average, we were able to identify whether an anomaly was a false or true positive in less than a minute.

As stated earlier, identification is the ability of a network administrator to classify the root cause of the anomaly. In our case the output is labeled either *"<attack type>"*, *"normal traffic"*, or *"unidentified"*.

By manual inspection of the output from each detection system (listed in appendix B) we find their respective identification rates (see table 5.1). For the identification rate I_x let i_x be the number of identified anomalies and f_x represent the number of flagged anomalies where $x \in \{\text{SENATUS, HD + Apriori}\}$. The identification rate for each system can be expressed as:

$$I_x = \frac{i_x}{f_x} \tag{5.1}$$

As we can see, there is a significant difference in the identification rate across the two systems. With SENATUS (S), we are able to identify 98.1% of all feature value sets, whereas HD has a 65.5% identification rate. Because SENATUS outputs all feature values and their respective time bins, the amount of flows matching those values are relatively few, thus easily identified. Whereas identifying item sets (generated by HD) with three or less items, which typically match hundreds of thousands of flows,

proves to be a very challenging task.

Detection method	Identification rate
SENATUS	98.1%
HD	65.5%

Table 5.1: Identification rates for both SENATUS and HD.

Anomaly	SENATUS	HD
Port scan	15	1
Port sweep	91	11
DoS	18	18
Normal traffic	32	37
Unidentified	3	33
Total	159	100

Table 5.2: Anomalies detected in our dataset by SENATUS and HD.

Table 5.2 gives an overview of the malicious anomalies we are able to precisely identify. Overall, SENATUS identifies 124 malicious anomalies, 94 more than we manage to identify with the output from HD. As table 5.2 shows, SENATUS is much more likely to identify certain anomalies compared to HD. Out of the 15 port scans SENATUS detect, HD is only able to detect one. The results are fairly similar for port sweeps – while SENATUS detects 91, HD detects 11. Interestingly, we are able to identify the same amount of DoS attacks with both techniques, with the main reason being the nature of the attack. All DoS attacks identified spans several time bins, consist of 50-60 thousand flows each and with 5 feature values fixed, making it a 5-item set in the output of HD. The three anomalies we are not able to identify with SENATUS, could most likely be identified by an experienced network administrator. Two of the flagged time bins consist of several packets originating and terminating in port 110, which is used in POP3, the Post Office Protocol Version 3. Mail clients use this protocol to collect mail off server. It is possible that the traffic is non-malicious, however we do not have enough experience with the protocol, and are not able to verify this. The last unidentified anomaly has the characteristics of several hosts

replying a port scanner, but it can just as well be normal traffic. In HD there were as many as 33 anomalies we were unable to detect.

Based on the information found in Table 5.2, we find that the ratio of true positives against false positives is:

$$TP_R = \frac{\text{\# Anomalies}}{\text{\# Anomalies + \# Identified normal traffic}} \tag{5.2}$$

This ratio is shown in Table 5.3.

Detection method	Ratio of true positives
SENATUS	79%
HD	45%

Table 5.3: Identification rates for both SENATUS and HD.

5.2 Results From Comparison Based Ground Truth

Table 5.4 shows the number of respective time bins the different detection methods discover in our dataset, based on our methodology defined in Section 4.3. We calculate the intersection between SENATUS and the two other methods, the intersection between HD and ED, as well as the intersection of all the methods. We also calculate the intersection between anomalies found by SENATUS and anomalies found by the two other methods and finally the union of all the detected time bins. Based on the information in Table 5.4, we can visualize which time bins are discovered by each method with a Venn diagram as shown in Figure 5.1.

Detection method sets	Detected time bins
SENATUS (S)	104
HD	78
ED	64
S $\cap$ **HD**	15
S $\cap$ **ED**	13
HD $\cap$ **ED**	55
S $\cap$ **HD** $\cap$ **ED**	4
S $\cap$ (**ED** $\cup$ **HD**)	24
S $\cup$ **HD** $\cup$ **ED**	167

Table 5.4: Time bins discovered by the different detection methods based on our methodology

We observe that SENATUS has the highest discovery rate of the sets, but as we can see, the overlap between SENATUS and the other anomaly detectors is relatively small. This means that SENATUS discovers a largely separate set of anomalies compared to the other detection techniques. By calculation, we find that the percentage of anomalies SENATUS detects that are not detected by any of the other techniques is high:

$$\frac{S - (S \cap (ED \cup HD))}{S} = 81\% \tag{5.3}$$

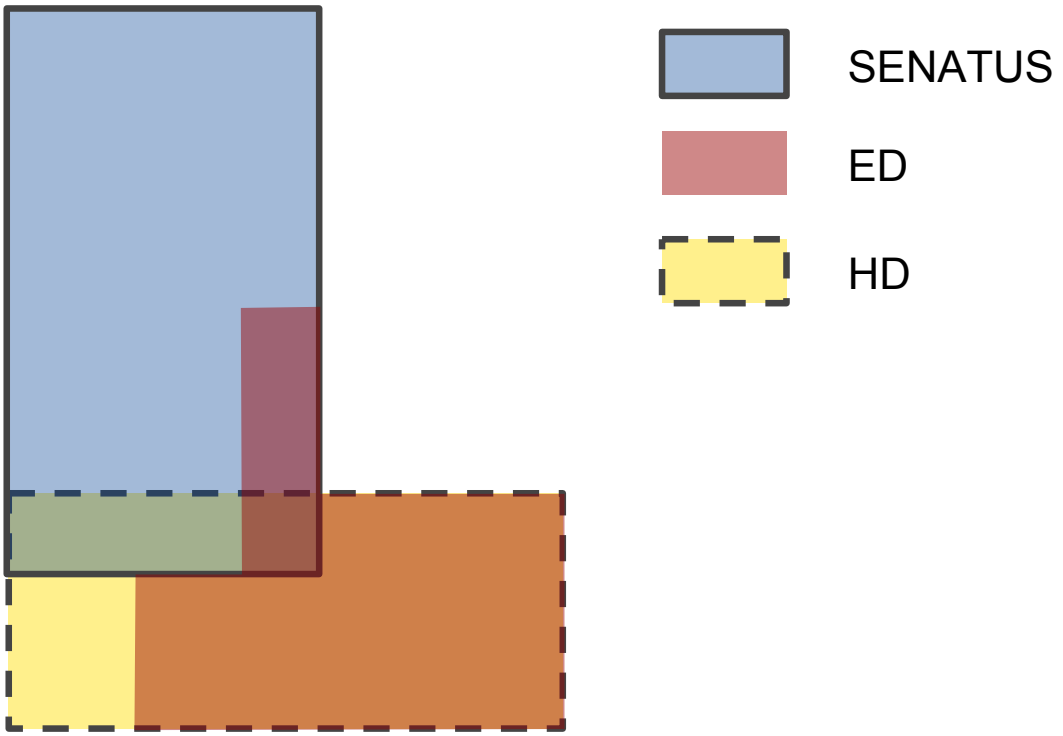

Figure 5.1: A Venn diagram showing the relationship between which time bins were detected by the different methods

Within our dataset, Entropy does not detect any anomalies that are not also detected by either SENATUS or HD. Our findings are that HD and Entropy discover many of the same anomalies; as many as 86% of the anomalies flagged by Entropy based metrics are also flagged by HD, compared to 20% for SENATUS. Table 5.5 gives a summarization of the detection rates for the different techniques.

Detection method	Detection rate
SENATUS	62%
HD	47%
ED	38%

Table 5.5: The detection rate of each method based on our ground truth

5.3 Overview of SENATUS

SENATUS detects a wide range of anomalies with relatively small intensities. Table 5.6 is a guideline illustrating at what intensities SENATUS is able to detect and identify malicious anomalies. The intensity I is derived from the following formula:

$$I = \frac{\#\text{malicious flows}}{\#\text{total flows}} \tag{5.4}$$

Anomaly	Intensity (%)	
Port sweep	minimum	0.01
	average	0.48
Port scan	minimum	0.03
	average	0.75
DoS	minimum	8.56
	average	13.71

Table 5.6: Detection intensities for SENATUS.

Note that these values may not represent the lowest intensity of which SENATUS can detect malicious traffic. Moreover, the feature values of a malicious anomaly do not have to reside within the top-n set during its corresponding time interval – it is sufficient that they are listed at some point during the day. This is because the second phase of the initial extraction will iterate over the same data again, based on the union of all top-n sets, and therefore expose anomalies that do not have all their respective feature values within the top-n set. However, this also explains the reason why our initial attempt at creating an artificial ground truth was unsuccessful. When manually evaluating the injected attacks, we discovered that the source port value altered between 35675 and 35676. Since none of these values generated enough flows to be in the top-n set any time during that day, they simply got discarded in the first extraction phase.

For a wide range of malicious traffic detected and identified, it is possible that SENATUS only outputs a part of the attack. For example, let $A = [4314, 224, 6000, 1433]$ be a set in the output of SENATUS consisting of 42 flows. At the first glance this might appear to be a port sweep, but further investigation through manual inspection can sometimes reveal that this is just a part of a larger port scan, with a number of flows ranging in the order of hundreds.

SENATUS can be further tuned, both in the first extraction phase by varying the top-n features and by adjusting $weight$ in the robust principal component analysis (Equation 2.5). By adjusting these parameters for the same days in **November**, we have found by manual inspection of the dataset that SENATUS performs very satisfying with the parameters given in Table 5.7. As suggested by [4], we achieve at most a 20% approximation error by choosing the top 23 and 20 feature values for [Source port, Destination port] and [Source address, Destination address] respectively. So by eliminating a large ratio of the feature values, we will still represent the behaviour of the data set with at least 80% accuracy.

Figure 5.2 shows the relation between $weight$ and the false positive ratio, while figure 5.3 illustrates a proportional relationship between the amount of detected anomalies for two different values of $weight$. As expected, we observe that the amount of detected anomalies increases for lower values of lambda, however this also implies a higher amount of false positives. By setting the tuning parameter to 3, the sensitivity to abrupt variations of RPCA decreases, and SENATUS achieves a 90-100% ratio of true positives.

Parameter	Value
$weight$	3
SrcPort, DstPort	top-20
SrcAS, DstAS	top-23

Table 5.7: Parameter settings SENATUS.

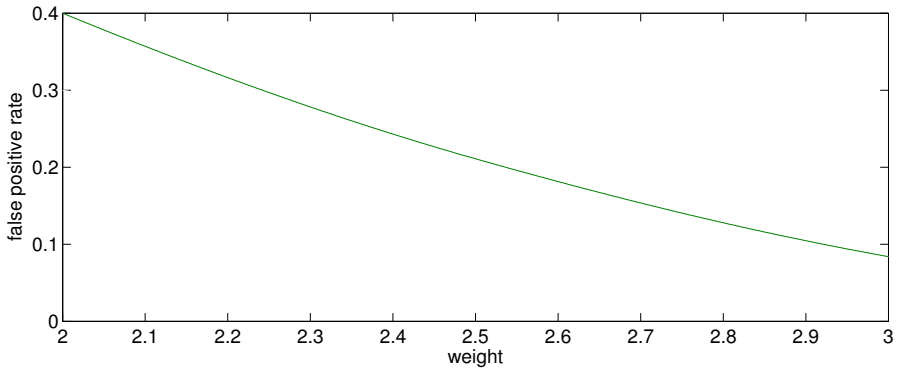

Figure 5.2: False positive rate as a function of $weight$ in robust principal component analysis.

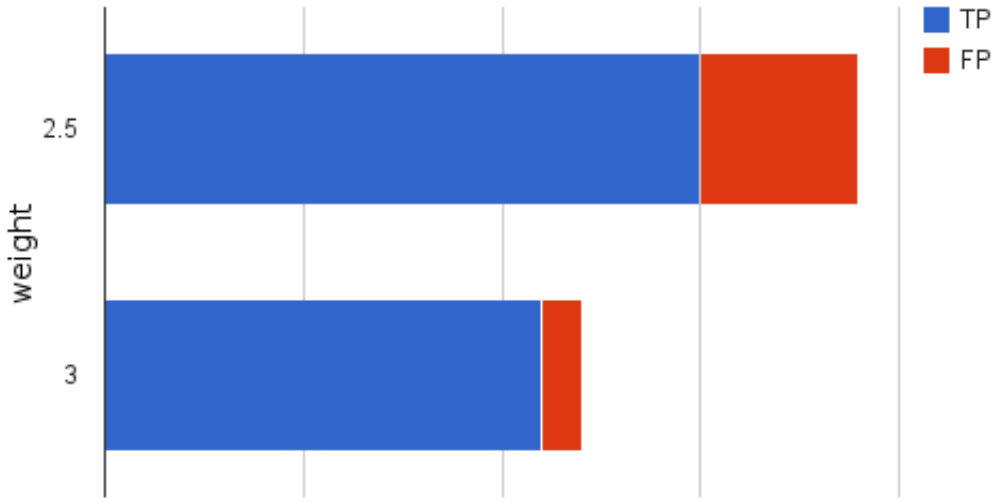

Figure 5.3: Proportinal relationship between the amount of detected anomalies for two different values of $weight$.

5.4 Evaluation and Discussion

While the results based on our ground truth gives a fairly accurate view of the detection metrics, we can never completely verify the authenticity without a thoroughly known baseline. When not knowing the total number of anomalies that resides within our dataset the need for alternative ways of evaluation rise. Having a sufficient amount of data is also important to give a statistically correct measurement. While having a relative big amount of data, we observed that there were fluctuations in our results across the different months.

HD

When evaluating the implementation of histogram-based detection, we found it challenging to identify the root cause of the anomalous time bins. This difficulty was mainly introduced in the process of generating meta-data for further pre-filtering. At certain days, the majority of all feature values were flagged as anomalous (potentially 65535 ports and ASs, both source and destination) and we found it infeasible to create the combination of all flagged feature values in terms of computing power. In general we observed that the overall runtime increased significantly when the number of flagged feature values increased substantially. Sometimes the run times could become so long that it was not practically feasible to perform the calculations. E.g. we had four traffic feature vectors, each containing N elements. To make the permutation of these elements we needed four for-loops. If $N = 100$, the corresponding number of iterations is 100^4. As can be seen the number of iterations grows exponentially, and eventually becomes too high for our setup to run in a reasonable time frame.

The aforementioned problem lead to another challenge of using HD, which appeared when manually inspecting the output of Apriori. The challenge was to properly

identify whether an item set represented a malicious activity or not. We experienced that most of the item sets could be classified as normal activity or as unidentified. The latter because the number of items contained in each set was too low to draw any conclusions (typically l-item sets where $l < 3$). A better approach might have been to adjust the implementation of HD to accept IP addresses instead of ASs. This would have made the identification process smoother. However, capturing IPs would have led to a tremendous increase in the dimensionality.

SENATUS

SENATUS has a very good identification rate since it always lists a full set of feature values and their respective time bin.

SENATUS can potentially ignore anomalies due to their nature. This can occur e.g. when a DoS has a BPP > 100, which can be the case for an ICMP flood. If this is the case for many anomalies, the detection rate for SENATUS will drop. Another limitation in the current version of SENATUS is cross detection of anomalies. An example of this is DoS backscatter, where the victim of the DoS attack sends reply packets to the attacker. This backscatter has a signature that can very easily be falsely identified as a scan. In fact, the scan detector part of SENATUS has a higher detection rate of DoS attacks than the DoS detector. For the purpose of this report, we have not considered cross detection as misidentification – we look upon each part of SENATUS as a whole.

6

Conclusion

It is not an easy task to evaluate and compare different anomaly detection systems. The main challenge in our work was to establish a well defined baseline for further analysis and comparison. Our first attempt at evaluating SENATUS, was based on injecting anomalies through the backbone network of Uninett. When analyzing the results, we discovered that the anomalies did not produce enough flows to be properly detected by SENATUS. This was due to both the configuration of the anomaly injector and the maximum intensity – which was too low.

The second approach was root cause analysis of alarms generated by the anomaly detectors. Root cause analysis does not include possible false negatives – anomalies

that should have been detected, but were not. To attempt to rectify this problem, we developed a third method to further prove the validity of our results. This approach was based on a comparison between SENATUS and the other detection methods. Time bins which were flagged by the other detection methods, but not by SENATUS, were viewed as false negatives. Our results were based on the last two methods: the root-cause analysis based ground truth and the comparison based ground truth.

Based on the comparison based ground truth, we have seen that the detection performance of SENATUS is highly satisfying compared to both histogram-based detection (HD) and entropy-based detection (ED). SENATUS had a 62% detection rate, while HD and ED had a detection rate of 47% and 38% respectively. ED and HD detected mostly the same anomalies, while we found that SENATUS detected a largely separate set of anomalies.

When evaluating the performance of SENATUS and HD with the root cause analysis, we found that we could identify 98.1% of the anomalies flagged by SENATUS. Histogram-based detection had an identification rate of 65.5%. Among the anomalies identified by each system, SENATUS achieved a ratio of 79% true positives (TP), while HD had a ratio of 45%.

We found that identifying whether an anomaly is benign or malicious, took drastically less time with SENATUS compared to the other detection methods.

6.1 Future Work

Future work should involve optimization of the different detection schemes, preferably an implementation in a low-level programming language with highly developed mathematical facilities such as c, Matlab, or Fortran. This will lead to great improvements in terms of execution time, thus obtaining more results and increasing the accuracy of the performance metrics, e.g. detection and identification rate.

To obtain more accurate results in terms of statistics, a larger data set should be used. A labeled dataset where we have the knowledge of all present anomalies, gives us the possibility to take into account false negatives – true anomalies that were not detected by any method.

Another very interesting topic open for further research is the deployment of pre-configured extraction methods in gateway routers. The implementation of all three detection methods in this paper involves an initial extraction phase of NetFlow data. This is a time consuming process due to the huge amount of information and dimensionality in the dataset. A pre-configured extraction method for a detection scheme could potentially avoid most of the initial extraction, hence decrease the overall runtime significantly.

Bibliography

[1] Nmap reference guide. http://nmap.org/book/man.html, June, 2011. Accessed June 26, 2011.

[2] A. D. Robbins. GAWK: Effective AWK Programming. http://www.gnu.org/software/gawk/manual/gawk.pdf, April, 2009. Free Software Foundation, Inc. Accessed June 12, 2011.

[3] A. Abdelkefi and Y. Jiang. "Robust Traffic Anomaly Detection with Principal Component Pursuit". ACM CoNEXT Student Workshop, November, 2011.

[4] A. Abdelkefi and Y. Jiang. "Compressible Traffic Features". ACM SIGCOMM'11, August, 2011.

[5] R. Agrawal, T. Imielinski, and A. Swami. "Mining Association Rules Between Sets of Items in Large Databases". pages 207–216, SIGMOD Conference 1993.

[6] R. Agrawal and R. Sirkant. "Fast Algorithms for Mining Association Rules". pages 487–499, Proceedings of 20th International Conference on Very Large Data Bases, Santiago, Chile, September 12-15, 1994.

[7] A. Banerhee, V. Chandola, V. Kumar, and J Srivastava. Anomaly Detection: A Tutorial [Power Point presentation]. www.siam.org/meetings/sdm08/TS2.ppt, SIAN conference on Data Mining, April 2008. Accessed May 28, 2011.

[8] P. Barford, J. Kline, D. Plonka, and A. Ron. "A Signal Analysis of Network Traffic Anomalies". Proceedings of ACM SIGCOMM Internet Measurement Workshop, Nov, 2002.

[9] D. Brauckhoff, X. Dimitripoulos, K. Salamatian, and A. Wagner. "Anomaly Extraction in Backbone Networks using Association Rules". *IMC'09*, November, 2009.

[10] D. Brauckhoff, K. Salamatian, and Martin May. "Applying PCA for Traffic Anomaly Detection:Problems and Solutions". IEEE INFOCOM, April, 2011.

[11] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. "Impact of Packet Sampling on Anomaly Detection Metrics". IMC'06, October 25-27, 2006.

[12] John F. Burns and Ravi Somaya. "Hackers Attack Those Seen as WikiLeaks Enemies". http://www.nytimes.com/2010/12/09/world/09wiki.html?_r=1, December, 2010. Accessed June 8, 2011.

[13] C. Ramey, and B. Fox. Bash Reference Manual. http://www.gnu.org/software/bash/manual/bash.pdf, December, 2009. Free Software Foundation, Inc. Accessed May 26, 2011.

[14] E. B. Claise. Cisco systems netflow services export version 9. RFC 3954, Cisco Systems, 2004. http://www.ietf.org/rfc/rfc3954.txt. Accessed May 21, 2011.

[15] David Dittrich. "The stacheldraht distributed denial of service attack tool". http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt, December, 1999. Accessed June 8, 2011.

[16] Free Software Foundation. Sed, a stream editor. http://www.gnu.org/software/sed/manual/sed.html, August, 2010. Free Software Foundation, Inc. Accessed May 26, 2011.

[17] A. Kind, M. Stoecklin, and X. Dimitropoulos. "Histogram-Based Traffic Anomaly Detection". IEEE Transactions on Network and Service Management, vol. 6, no. 2, June 2009.

[18] A. Lakhina, M. Crovella, and C. Diot. "Mining Anomalies using Traffic Feature Distributions". ACM SIGCOMM'05, August, 2005.

[19] F. Lau, S.H. Rubin, M.H. Smith, and L. Trajkovic. "Distributed denial of service attacks". *Systems, Man, and Cybernetics, 2000 IEEE International Conference*, August, 2002. Accessed June 25, 2011.

[20] Jonathan Lemon. "Resisting SYN flood DoS attacks with a SYN cache". http://people.freebsd.org/~jlemon/papers/syncache.pdf, December, 2001. Accessed June 9, 2011.

[21] G. Nychis. "An Empirical Evaluation of Entropy-based Anomaly Detection", May, 2007.

[22] Perl.com. The Perl Programming Language. http://www.perl.com/, May, 2011. Accessed May 17, 2011.

[23] P. E. Proctor. "Marketscope for Network Behaviour Analysis". *Gartner Research Report G00144358, Gartner Inc.*, November, 2009.

[24] RFC2828. Internet Security Glossary. http://tools.ietf.org/html/rfc2828, May, 2000. Accessed June 12, 2011.

[25] F. Silveira, C. Diot, N. Taft, and R. Govindan. "ASTUTE: Detecting a Different Class of Traffic Anomalies", August, 2010.

[26] H. Skrautvol and M. Ask. "Internet Attack Simulation - Empirical evaluation of the entropy boundaries for network anomaly detection". Technical report, The Norwegian University of Science and Technology, January, 2010.

[27] Praetox Technologies. "LOIC, low orbit ion cannon". http://sourceforge.net/projects/loic/, June, 2011. Accessed June 8, 2011.

[28] TheFreeDictionary.com. Definition of anomaly. http://www.thefreedictionary.com/anomaly. Accessed June 27, 2011.

[29] J. D. Tisdall. Beginning Bioinformatics. http://www.perl.com/pub/2002/01/02/bioinf.html, January, 2002. Accessed May 17, 2011.

[30] UNINETT AS. Drift Uninett - Status for nett og tjenester. http://drift.uninett.no/kartg/last/uninett/norge/geo/nuh, May, 2011. Accessed May 15, 2011.

[31] UNINETT AS. Om UNINETT. http://www.uninett.no/, May, 2011. Accessed May 15, 2011.

[32] Wikipedia. "The anonymous hacker group". http://en.wikipedia.org/wiki/Anonymous_(group), June, 2011. Accessed June 8, 2011.

[33] Wikipedia, the free encyclopedia. Entropy (information theory). http://en.wikipedia.org/wiki/Entropy_(information_theory), June, 2011. Accessed May 24, 2011.

[34] Wikipedia, the free encyclopedia. Kullback-Leibler divergence. http://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence, April, 2011. Accessed May 20, 2011.

A Code

This appendix contain parts of our implementation of histogram-based detection.

A.1 histogram-based detection

The following scripts is our contribution to the implementation of the histogram-based detection scheme.

A.1.1 anoflows.m

Anoflows.m is used to identify the time bins flagged by all traffic feature, and then to aid in the creation of candidate flows.

```
function [] = anoflows(day,month, year)
% This script does the following:
%
% 1.Creating suspicious flows by
%   combining all the different features
%   during one day.
% 2.Create the union of anomalous timebins
%   foe each feature.

clc;
```

```
clearvars -except s day month year;

% Add a zero if day or month is less than 10
if(day<10)
    day=strcat('0',num2str(day));
else
    day=num2str(day);
end
if(month<10)
    month=strcat('0',num2str(month));
else
    month=num2str(month);
end

year=num2str(year);

% Load anomalous time bins.
mat = 'anomalousbin.mat';
sas = ['Mat/' year '/' day month '/srcas/' mat];
das = ['Mat/' year '/' day month '/dstas/' mat];
srcp = ['Mat/' year '/' day month '/srcport/' mat];
dstp = ['Mat/' year '/' day month '/dstport/' mat];

srctime = importdata(sas);
dsttime = importdata(das);
srcporttime = importdata(srcp);
dstporttime = importdata(dstp);


% Generate the union of timebins and put it into a cell array.

intersectmat = union(union(union(srctime,dsttime),srcporttime),dstporttime);

for j=1:size(intersectmat,2)
    pre = floor((intersectmat(j)*5)/60);
    post = mod(intersectmat(j)*5, 60);

    if(post==0)
        post = 55;
        pre = pre-1;
    else
        post = post-5;
```

```
    end

    if(pre<10)
        pre=strcat('0',num2str(pre));
    else
        pre=num2str(pre);
    end
    if(post<10)
        post=strcat('0',num2str(post));
    else
        post=num2str(post);
    end

    time = strcat(num2str(pre), num2str(post));
    uniontimebin(j) =  {time};
end

% Load suspicious features
mat = 'suspicious.mat';
sas = ['Mat/' year '/' day month '/srcas/' mat];
das = ['Mat/' year '/' day month '/dstas/' mat];
srcp = ['Mat/' year '/' day month '/srcport/' mat];
dstp = ['Mat/' year '/' day month '/dstport/' mat];

srcas = importdata(sas);
dstas = importdata(das);
srcport = importdata (srcp);
dstport = importdata (dstp);

% Check if the feature is present in the union of timebins, ↩
   → if it is
% we want to keep it.
n1=1;
n2=1;
n3=1;
n4=1;
for j=1:size(intersectmat,2)
    for i1=1:size(srcas,2)
        if (srcas(2,i1)==intersectmat(j))
            srcasf(1,n1)=srcas(1,i1);
            srcasf(2,n1)=intersectmat(j);
            n1=n1+1;
        end
    end
```

```
    for i2=1:size(dstas,2)
        if (dstas(2,i2)==intersectmat(j))
            dstasf(1,n2)=dstas(1,i2);
            dstasf(2,n2)=intersectmat(j);
            n2=n2+1;
        end
    end
    for i3=1:size(srcport,2)
        if (srcport(2,i3)==intersectmat(j))
            srcportf(1,n3)=srcport(1,i3);
            srcportf(2,n3)=intersectmat(j);
            n3=n3+1;
        end
    end
    for i4=1:size(dstport,2)
        if (dstport(2,i4)==intersectmat(j))
            dstportf(1,n4)=dstport(1,i4);
            dstportf(2,n4)=intersectmat(j);
            n4=n4+1;
        end
    end
end

   srcasf = unique(srcasf);
   dstasf = unique(dstasf);
   srcportf = unique(srcportf);
   dstportf = unique(dstportf);

% Generate all the combinations of traffic features based on the intersection of
% timebins.
n=1;
size(srcasf,1)
for j=1:size(srcasf,1)
    for k=1:size(dstasf,1)
        for l=1:size(srcportf,2)
              for m=1:size(dstportf,2)
                 if (flows(:,n-1)~=[srcasf(j) dstasf(k) srcportf(l) dstportf(j)])
                   flow(:,n)= [srcasf(j,1) dstasf(k,1) srcportf(1,l) dstportf(1,m)];
                   n=n+1;

                 end
```

```
            end
        end
    end
end

flow = uniqueCols(flow);

% Write out potentially anomalous flows
save = ['Text/' year '/' day month '/entapflows2.txt'];
dlmwrite(save,flow,'delimiter','\t');

% ...and the union of suspicios timebins to .txt file
save = ['Text/' year '/' day month '/aptimebin.txt'];
fid = fopen(save, 'wt');
fprintf(fid, '%s\t', uniontimebin{:});
fclose(fid);
end
```

A.1.2 mining.sh

Mining.sh processes the candidate flows and extract flows that matches the feature values from the original data set.

```
#!/bin/bash

# INPUT VARIABLES
day=$2
month=$1
year=$3
file="$year$month$day"
dir1="$HOME/master/anomalydetection/Text/$year/$day$month"

echo $dir1

# OUTPUT VARIABLES
dir2="$HOME/master/anomalydetection/apriori/$year/ ↩
   → $day$month"
output="entap_suspicious.txt"

# NFDUMP & APRIORI VARIABLES
format="fmt: %sas %das %sp %dp %pr %pkt %byt %ts %td"
gawk="gawk -f $HOME/master/formatting.awk"
sed='sed /S/d;'

```

```
21 if [ -e $dir1 ]
22 then
23
24 no=`cat $dir1/entap.txt | awk 'END{print NF}'`
25
26 for (( i=1;i<=$no;i+=1 ))
27 do
28
29 timebin=`cat $dir1/entap.txt | awk '{print $'$i'}'`
30 echo $timebin
31
32 features=`gawk 'END{print NF}' $dir1/entapflows.txt`
33 echo $features
34
35 y=100
36 cnt=1
37 while [ $cnt -le $features ]
38 do
39
40 if [ $cnt -eq $y ]
41 then
42 echo "Cnt: "$cnt
43 let y=$y+100
44 fi
45
46 cd $dir1
47
48 srcas=`gawk '{if(NR==1) print $'$cnt'}' entapflows.txt`
49 dstas=`gawk '{if(NR==2) print $'$cnt'}' entapflows.txt`
50
51 cd /data/netflow/oslo_gw/$year/$month/$day/
52
53 nfdump -r nfcapd.$file$timebin -o "$format" 'src as '$srcas
   → ' and dst as '$dstas''  | $sed  | $gawk  >> $dir2/
   → ap_suspicious$day$month.txt
54
55 let cnt++
56
57 done
58 done
59 fi
```

A.1.3 formatting.awk

Formatting.awk is applied to the output from the mining process to make it comatible with Apriori.

```
#Script to convert timeformat to POSIX and to align the values the preferable format.
BEGIN{
    OFS=","
    FS=" "
}

function toposix(d,t ){
split(d, dates, "-")
split(t, times, ":")
return mktime(dates[1]" "dates[2]" " dates[3]" " times[1]" " times[2]" "times[3])
}

{

if ($5 ~ /TCP/){
    print $1,$2,$3,$4,"6",$6,$7,toposix($8,$9),$10
}

if ($5 ~ /UDP/){
    print $1,$2,$3,$4,"17",$6,$7,toposix($8,$9),$10
}

if ($5 ~ /ICMP/){
    print $1,$2,$3,$4,"1",$6,$7,toposix($8,$9),$10
}

if ($5 ~ /GRE/){
    print $1,$2,$3,$4,"47",$6,$7,toposix($8,$9),$10
}

if ($5 ~ /IPv6/){
    print $1,$2,$3,$4,"41",$6,$7,toposix($8,$9),$10
}
}

END{
}
```

B

Identification rates for HD and SENATUS

The following 7 pages list the output for both detection schemes. Page 88-92 is the output from SENATUS and page 93-94 is the output from HD + Apriori. The output has been used for identification and root cause analysis.

Day	Timebin	Flows (total)	Event	BPP	Flows (anomaly)	%	Identification	False positive
11/1/2010								
	445	740154	DoS	29	97634	13.19%	1	0
			DoS	29	99667	13.47%	1	0
			DoS	29	63383	8.56%	1	0
			DoS	29	80940	10.94%	1	0
	450	1168271	DoS	29	116711	9.99%	1	0
			DoS	29	209232	17.91%	1	0
			DoS	29	205328	17.58%	1	0
			DoS	29	131906	11.29%	1	0
			DoS	29	162185	13.88%	1	0
	455	1166945	DoS	29	208473	17.86%	1	0
			DoS	29	204712	17.54%	1	0
			DoS	29	130888	11.22%	1	0
			DoS	29	162055	13.89%	1	0
	500	1134591	DoS	29	114391	10.08%	1	0
			DoS	29	206572	18.21%	1	0
			DoS	29	204343	18.01%	1	0
			DoS	29	103129	9.09%	1	0
			DoS	29	160252	14.12%	1	0
	1045	1269425	Port sweep	47	909	0.07%	1	0
	1235	1342635	Port sweep	47	894	0.07%	1	0
	1555	1169256	Port sweep	47	1880	0.16%	1	0
	1800	1064741	Port sweep	47	583	0.05%	1	0
	1850	1066061	POP3	45	4070	0.38%	0	
	1900		normal HTTPS/SSL traffic	146			1	1
	1905	1037947	Port sweep	40	934	0.09%	1	0
	2000	976905	Port sweep	47	535	0.05%	1	0
	2105	992800	POP3	45	1442	0.15%	0	
11/2/2010								
	5	749995	Port scan	40	377	0.05%	1	0
	240	436561	Port sweep	40	42	0.01%	1	0
	1100	1291805	Port sweep	40	527	0.04%	1	0
	1135	745707	Normal SSL traffic				1	1
	1140		Normal traffic				1	1
	1330	1328863	Port sweep	40	518	0.04%	1	0
	1405	1297121	Port sweep	40	427	0.03%	1	0
			Port sweep	40	1506	0.12%	1	0
	1435	1331311	Port sweep	48	554	0.04%	1	0
	1825	1056754	Port sweep	40	562	0.05%	1	0
	1840	1109792	Port sweep	40	594	0.05%	1	0
	1850		normal ICMP traffic				1	1
	1855		No clear anomalous pattern				1	1
	1900	773748	Port Sweep	40	490	0.06%	1	0
	2110		Gaming (XBOX live ++)				1	1

Day	Timebin	Flows (total)	Event	BPP	Flows (anomaly)	%	Identification	False positive
11/12/2010								
	625	345577	Port sweep	48	637	0.18%	1	0
	630	341894	Port sweep	48	1938	0.57%	1	0
	850		Random traffic				1	1
	855		Random traffic				1	1
	1335	1207997	FP	83			1	1
	1440	1166679	Port scan	40	737	0.06%	1	0
	1555	1064469	Port scan	40	9262	0.87%	1	0
	1825	915361	Port scan	44	1314	0.14%	1	0
			Random traffic				1	1
	1835	903375	Port scan	44	709	0.08%	1	0
	1850	912427	Port scan	44	3187	0.35%	1	0
	1900	907999	Port scan	44	3053	0.34%	1	0
	1935	882901	Port sweep	40	2457	0.28%	1	0
	2025	850759	Port sweep	40	637	0.07%	1	0
	2135	840105	Port sweep	40	646	0.08%	1	0
	2345	758524	Port sweep	47	656	0.09%	1	0
12/12/2010								
	200	868023	Port sweep	48	1938	0.22%	1	0
	535	718357	Port scan	40	668	0.09%	1	0
	1110	999965	Gaming (Brother in Arms +)			0.00%	1	1
	1135	1067878	Port sweep	48	3244	0.30%	1	0
	1140	1072468	Port sweep	40	3235	0.30%	1	0
	1255		ICMP req/ack				1	1
	1405	1161915	Port sweep	40	2021	0.17%	1	0
	1740		Random traffic				1	1
	1750		Random traffic				1	1
	1800	1205250	Port sweep	48	1600	0.13%	1	0
	1835		Random Traffic				1	1
	1840		Random Traffic				1	1
11/14/2010								
	125	646028	Port sweep	48	2527	0.39%	1	0
	350	417728	SSH bruteforce	84	3578	0.86%	1	0
	740	299041	Port sweep	40	963	0.32%	1	0
	750	299814	Port sweep	40	973	0.32%	1	0
	1100	613538	Port sweep	40	661	0.11%	1	0
	1440	943247	Port sweep	40	574	0.06%	1	0
	1655		Random Traffic	85			1	1
	1805		Random Traffic	94			1	1
	1835		Random Traffic	96			1	1

Day	Timebin	Flows (total)	Event	BPP	Flows (anomaly)	%	Identification	False positive
	1845		Random Traffic	264			1	1
	1850		Random Traffic	170			1	1
	1900		Random Traffic	96			1	1
	1940	1050601	Port sweep	40	1937	0.18%	1	0
	2040	1025786	ICMP req/ack	155			1	1
	2130	1052698	Port sweep	40	1154	0.11%	1	0
12/18/2010								
	155	433820	Port sweep	40	649	0.15%	1	0
	210	410520	Port sweep	40	1072	0.26%	1	0
	220	399264	Port sweep	40	1907	0.48%	1	0
	230	389529	Port sweep	40	1165	0.30%	1	0
			Port sweep	40	678	0.17%	1	0
	340	324827	Port sweep	40	1053	0.32%	1	0
	425	299168	Port sweep	47	1970	0.66%	1	0
			Port scan	40	331	0.11%	1	0
	430	299630	Port sweep	40	2117	0.71%	1	0
	525	274330	Port sweep	40	1097	0.40%	1	0
	850	322784	Port sweep	40	2443	0.76%	1	0
			Port sweep	40	516	0.16%	1	0
	1155	574986	Port sweep	40	1132	0.20%	1	0
	1200	589714	Port scan	40	65	0.01%	1	0
	1220	600258	Possible backscatter from a previous port sweep. Not very recognisable	179	584	0.10%	0	
	1315	623311	Port sweep	47	417	0.07%	1	0
	1405	656448	Port sweep	48	2611	0.40%	1	0
			Port sweep	48	570	0.09%	1	0
	1455	676810	Port sweep	40	1004	0.15%	1	0
	1610	685496	Port sweep	40	992	0.14%	1	0
	1615	684208	Port sweep	40	604	0.09%	1	0
			Port sweep	40	1593	0.23%	1	0
	1625	695192	Port sweep	40	1005	0.14%	1	0
			Random traffic	264			1	1
	1710		Random traffic	398			1	1
	1820	686921	Port sweep	40	1057	0.15%	1	0
	1825		Random Traffic	215				1
	2020	693056	Port sweep	40	1087	0.16%	1	0
	2145	685444	Port sweep	40	717	0.10%	1	0
12/21/2010								
	25		Random Traffic	540			1	1
	155	338724	Port sweep	40	515	0.15%	1	0

Day	Timebin	Flows (total)	Event	BPP	Flows (anomaly)	%	Identification	False positive
	250	312033	Port sweep	48	21039	6.74%	1	0
	255	309286	Port sweep	48	22056	7.13%	1	0
	300	314573	Port sweep	47	22167	7.05%	1	0
	305	310715	Port sweep	47	22024	7.09%	1	0
	310	299115	Port sweep	48	21418	7.16%	1	0
	315	294292	Port sweep	48	21418	7.28%	1	0
	555	226598	Port sweep	40	3507	1.55%	1	0
	600	230102	Port sweep	40	2917	1.27%	1	0
	605	222749	Port sweep	40	3849	1.73%	1	0
	755	353397	Port sweep	40	561	0.16%	1	0
	935	622203	Port scan	40	8109	1.30%	1	0
	940	622662	Port scan	40	9514	1.53%	1	0
	1050		Random traffic				1	1
	1305		Random Traffic				1	1
			Random Traffic				1	1
	1345	730410	Port sweep	40	545	0.07%	1	0
	1525	668097	Port sweep	40	1458	0.22%	1	0
	1655	541561	Port sweep	47	692	0.13%	1	0
	1700	542459	Port sweep	47	2102	0.39%	1	0
	1745	523577	Port sweep	47	667	0.13%	1	0
	1750	524168	Port sweep	47	2014	0.38%	1	0
	1935	521171	Port sweep	48	2619	0.50%	1	0
			Port sweep	48	2263	0.43%	1	0
	2050	523456	Port sweep	40	599	0.11%	1	0
	2230	518478	Port sweep	48	624	0.12%	1	0
	2240	513160	Port sweep	48	647	0.13%	1	0
	2300	486119	Port sweep	40	1198	0.25%	1	0
3/1/2011	2125		Random traffic				1	1
3/25/2011								
	45	121764	Port sweep	40	110	0.09%	1	0
	830	121764	Port sweep	40	340	0.28%	1	0
	1635	158855	Port sweep	64	204	0.13%	1	0
	1705	153513	Port sweep	40	119	0.08%	1	0
3/26/2011								
	705	50255	Port sweep	40	57	0.11%	1	0
	1850	155158	Port sweep	40	95	0.06%	1	0
	1925	161392	Port sweep	84	2176	1.35%	1	0
	1935	159503	Port Sweep (Looks like a port sweep, but the packet sizes are pretty random)	84	2957	1.85%	1	0
	1945		Random traffic	484				1

Day	Timebin	Flows (total)	Event	BPP	Flows (anomaly)	%	Identification	False positive
	2155	145258	Port sweep	40	206	0.14%	1	0
	2310	131057	Port sweep	40	262	0.20%	1	0
3/28/2011								
	1545	220964	Port scan	40	2202	1.00%	1	0
	1550	217892	Port scan	40	2258	1.04%	1	0
	1705	197522	Portsweep	48	168	0.09%	1	0
	2035	196523	Port scan	40	315	0.16%	1	0

	#	Intensity(min)	Intensity(avg)			Identification		
Port scan	88	0.01%	0.48%					
Port sweep	15	0.03%	0.75%		Time bins identified	154		
DoS	18	8.56%	13.71%		Time bins not identified	3		
					Identification rate	98.09%		
TOTAL	121							

Day	Timebins	Flows (avg)	Event	BPP	Flows (anomaly)	%	Identification	False positive
28-Mar-2011								
	0515, 0600, 0650, 0720, 0725	70232	Port sweep	56	610	0.87%	1	0
			Random traffic				1	1
25-Mar-2011								
	0230,0245	66,619	Random traffic				1	1
			Random traffic				1	1
			Not feasible to verify				0	
			Random traffic				1	1
			Not feasible to verify				0	
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Port sweep	40	645	0.97%	1	0
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
26-Mar-2011								
	0200,0205, 0800,0820	73,951	Port scan	29	1050	1.42%	1	0
			Random traffic				1	1
			Port sweep	46	790	1.07%	1	0
			Random traffic				1	1
			Random traffic				1	1
			Not feasible to verify				0	
			Random traffic				1	1
			Port sweep	47	740	1.00%	1	1
1-Mar-2011								
	1300,1305, 1420,1425	70,000	Not feasible to verify				1	0
			Random traffic				1	1
12-Dec-2010								
	0720,0735, 0740	691,649	Random traffic				1	1
			Port sweep	46	3500	0.51%	1	0
			Random traffic				1	1
			ICMP traffic				1	1
			Random traffic				1	1
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Port sweep	47	2400		1	0
21-Dec-2010								
	0040,0250, 0255,0300	935,892	Port sweep	48	15,055	1.61%	1	0
			Port sweep	41	8833	0.94%	1	0
			Random traffic				1	1
			Not feasible to verify				0	
			Not feasible to verify				0	
			Random traffic				1	1
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
1-Nov-2010								
	0445,0450, 0505,0500	950,053	DoS	29	83,433	8.78%	1	0
			DoS	29	108,908	11.46%	1	0
			DoS	29	83,088	8.75%	1	0
			DoS	29	87512	9.21%	1	0

Day	Timebins	Flows (avg)	Event	BPP	Flows (anomaly)	%	Identification	False positive
			DoS	29	86,987	9.16%	1	0
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Not feasible to verify				0	
18-Dec-2010								
	0030,0035, 0700,0720, 0800		Not feasible to verify				0	
			ICMP traffic				1	1
			ICMP traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
2-Nov-2010								
	1135,1140, 1905,1920		Random traffic				1	1
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Random traffic				1	1
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
14-Nov-2010								
	0755,0810, 0900,0910	330,198	Port sweep		4,705	1.42%	1	0
			Random traffic				1	1
			Port sweep		6,640	2.01%	1	0
			Random traffic				1	1
			Port sweep		3,649	1.11%	1	0
12-Nov-2010								
	0645,0650, 0705,0700		Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Not feasible to verify				0	
			Random traffic				1	1
	#	Intensity (min)	Intensity(avg)		Identification			
Port sweep	11	0.51%	1.15%		58			
Port scan	1	1.42%	1.42%		32			
DoS	18				0.64444444444444			
TOTAL	30							
Unidentified	33		TP	17				
Normal traffic	37		FP	41				
			TPR	0.29				
TOTAL	100							