Framework for generating IDS benchmarking Data sets

Stian Skjølsvik

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2007

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

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

Abstract

Benchmarking Intrusion Detection Systems, IDS, is needed for comparing different systems against each other and to determine how good a single system is. For this purpose there is a need to generate test data set, which is based on real network data. To construct data set that contains different attack profiles, special features of attacks are needed to be determined. These can be extracted from analyzing network traffic. The thesis will propose a framework for processing captured network packets and establishing connection records. We look into what features are relevant for IDS systems, by analyzing captured network packets from an academic network and trying to extract different characteristics that constitute attacks. These features can be used to uniquely identify a specific attack from all the connections. The experiment is used to determine characteristics of the constructed data set, and to determine the relevance of the extracted features.

Acknowledgements

During the work on this thesis I have received help and support in all phases. I am especially grateful to Professor Slobodan Petrović, who has provided guidance, expertise and encouragement during the work on my thesis. My fellow students also deserve thanks for providing input during discussions. I would also like to thank my family and friends for the patience and understanding they have shown. Last but not least, I would like to thank my girlfriend for her love, support and understanding during the whole period of time that went into this thesis.

Stian Skjølsvik, 25th June 2007

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic covered by this thesis

Benchmarking Intrusion Detection Systems, IDS, is needed to determine how good a system is, and to test different systems against each other. For testing these systems data sets are provided, which contain different attack profiles amongst benign traffic. Different methods have been proposed to generate such data sets, by either simulating network traffic [1] or extracting data sets based on real network flow. Both measures have advantages and disadvantages. There has been some critique of using simulated network flow, since it is not certain that it accurately represents the real world [2]. The main advantage of using simulated traffic is that we know all aspects of the environment and no unknown attacks can occur, it eliminates the problem of identifying unseen attacks when using real traffic. Sometimes it is favorable to use data that are as close to the real world as possible for generating benchmarking data sets.

This thesis will look into common attack features gathered from a real network environment, using honeypots and *tcpdump* [3]. We will deploy honeypots to gather network traffic, due to legal concerns with capturing packets from a live network. The recorded traffic will be used to develop a framework for extracting useful information from network packets and to apply data mining techniques. The focus of this thesis is construction of connection records and extracting features for detecting network attacks. We also provide an analysis of the attacks found in the data sets, and the possibility of detection these by means of the extracted features. With the analysis, we try to determine characteristics of the attacks and features that are relevant in generation of IDS benchmarking data sets.

1.2 Problem description

As mentioned above, there is a need to generate test data sets for IDS benchmarking, which are based on real data and can be shared openly between organizations [4]. In this thesis, we will look into what features of traffic are relevant for Intrusion Detection Systems, based on gathered traffic from an academic network. The challenge is to analyze network logs/traffic and try to extract different characteristics that constitute an attack. This data will be analyzed and processed to find the appropriate features that should be included in a benchmarking data set. There are at least two methods of processing that are relevant for this thesis, using heuristic or finding sequences of events to identify frequent episodes. In both cases different features will be extracted from the traffic and be included in the benchmarking data set.

This thesis will focus on extracting features by using heuristic techniques, and analyze these features and their relevance in identifying network attacks. We have used some of the features found in the KDD 99 data set, but we have also added three new content features that may be more relevant for detecting newer network attacks. The greatest challenge in this thesis is the preparation of the recorded network packets, captured from the traffic to and from the honeypots. This process also exports the captured packets to an XML file that the prototype can utilize.

1.3 Justification, motivation and benefits

Research into the field of generating IDS benchmarking data set based on real traffic is justified by the fact that there have been some critique on using data sets generated from simulated traffic. This solution is not always the best way to generate IDS benchmarking data sets. A good data set based on real traffic would be very useful for the entire IDS development community. Both practitioners in this field and researchers would benefit from having such a data set, and this is a strong motive and challenge for research. The stakeholders are both developers and users of IDS systems, because they get a common reference data set that they can use and trust. Researchers in the field can also benefit from having a new IDS test data set because there are very few good benchmarking data sets available [1, 5], and not all of them are publicly available [5]. These data sets are also needed for doing theoretical research.

1.4 Research questions

The research questions we are trying to answer in this thesis, bearing in mind the previous discussion, are the following:

1. How can common properties of attacks be extracted from a large and evergrowing set of attacks against computers networks?
2. What are the properties of an IDS benchmarking data set based on real traffic for whose generation the methods of extraction of common properties of attacks have been used?

1.5 Limitations

Because we use honeypots to gather network traffic, we do not know what kind of traffic we capture and have to limit the number of protocols. In this thesis, we will focus on extracting information from the four often used protocols *ICMP*, *IP*, *TCP* and *UDP* in the prototype. The framework proposed in this thesis is a prototype for extracting features and generate data sets, and can be used as a basis for implementing more sophisticated data mining techniques.

1.6 Definitions

A more formal definition on terms used in this master thesis:

- *Intrusion detection*, defined by [6],

  > is the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems.

- A *threat*, defined by [7],

  > in a communication network is a potential event or series of events that could result in the violation of one or more security goals.

- An *attack*,defined by [7],

  > is the actuall implementation of a threat.

- *Vulnerability*, defined by [8],

  > is a weakness that makes it possible for a threat to occur.

- *Exploit* is an actual implementation that utilizes a vulnerability to compromise a host, much of the same meaning as an attack.
- *Detection rate*, same as true positive rate, is the chance of detecting an attack.
- *False positive rate* is the probability of generating an alarm when it is not an attack.
- *False negative rate* is the probability of missing an attack, meaning no alarm is generated when an attack occurs.

2 Previous work

2.1 KDD CUP 1999

As stated in the introduction, there exist several data sets for benchmarking IDS systems. There are several problems related to these sets [2] and one of the most widely used is the KDD CUP data. This set is based on the results from the intrusion detection evaluation done by the Lincoln Laboratory in co-operation with DARPA, which took place in 1998 and 1999 [9, 10]. Other data sets are often not available for the general public, which makes it difficult for comparison.

In 1998 and 1999 the Lincoln Laboratory of MIT conducted an evaluation on certain intrusion detection systems. The two data sets used in these evaluations were training and test data set. Training data were gathered from a simulated network using network traffic and audit logs. These were then given to the participants that used them to adjust and test their algorithms using off line evaluation.

The data gathered from the test performed by MIT was used as basis for the KDD CUP data mining competition in 1999 [1, 11]. The data mining task was to build a predictive model to classify and distinguish between “bad” and “good” connections using some sort of classifiers. The attacks contained in the data set were divided into four main categories:

- DOS: denial-of-service, e.g. SYN flood.
- R2L: unauthorized access from a remote machine, e.g. by guessing a password.
- U2R: unauthorized access to local superuser (root) privileges, e.g. various buffer overflow attacks.
- Probing: surveillance and other probing, e.g. port scanning.

For these general attacks, 41 features were extracted that contained properties of the attacks in question. These were grouped into several general categories:

- Basic features related to basic *TCP* properties of individual connections (host based traffic features), e.g. protocol, flags, duration.
- Content features within a connection suggested by domain knowledge, e.g. number of failed login attempts or number of root accesses.
- Traffic features computed from a two second time window, e.g. number of connections to the same host given an interval of 2 seconds or percentage of connections that have SYN errors using a two second window.

To construct the basic features of a *TCP* connection, the connection records were sorted by destination hosts. Some probing attacks can use long period of time to complete, example scanning hosts or ports using a time interval of several minutes between requests, a time window was not used to construct these features. Instead a window of 100 connections to the same host was used to construct the host based traffic features to not miss these attacks.

Content features within a connection suggested by domain knowledge are constructed from the information in the data portion of a network packet. Looking into the unstructured data is an area that is open for research. Finding good algorithms for mining the payload of packets automatically is difficult. To construct features that looked for suspicious behavior in the packet payload domain knowledge was used together with data mining techniques. An example of a content feature is a high number of failed login attempts.

Time based features consist of same host and same service features of connection records. Same host features were constructed by analyzing connection records with connections to the same destination as the current connection in the past two seconds. Same service features were constructed using the same methodology by analyzing connection to the same service as the current in the past two seconds. Statistics related to protocol behavior, service and so on were computed and included as time based features.

2.2 Construction of features and models for intrusion detection systems by data mining

For constructing features as found in the KDD CUP 1999 data set, a data mining framework was developed for building intrusion detection models [12, 13]. This work is based on the field of knowledge discovery in databases (KDD) and can be defined as [14] the nontrivial process of identifying valid, novel, potential useful and ultimately understandable patterns in data. The KDD process involves a number of steps: interactive, iterative and user-driven [12, 15]:

- The first step is learning the application domain by doing theoretical research in an attempt to understand the data and the discovery task.
- The next step is preparation of the data. Removing noise and perform data cleansing to create a target data set. Another option is to focus on a subset of variables extracted from the whole data set. Often data reduction and projection is performed, by finding useful features to represent the data and reduce the effective number of variables to consider. This process involves dimensionality reduction and transformation of the data in question.
- After the preparation of the data, data mining techniques are applied. First the function of the data mining is established. It includes deciding the purpose of the model derived from the algorithms, e.g. classification, summarization, regression or clustering. The model depends of the data in question and the desired output from the data mining techniques. After the model is decided, appropriate algorithm(s) are chosen to search for patterns of interest.
- From the data mining, the results are interpreted, and the discovered patterns can be used to possible return to any of the previous steps to remove redundant or irrelevant information. After the results are interpreted, useful patterns are presented to the users.
- The last step is to use the discovered knowledge, e.g. incorporating this knowledge into a system or just document and report it to interested parties. In case of this thesis, this step result in generating a new data set.

For this thesis the greatest challenge is the step of data preparation, and here lie the problems. Preparing the gathered information was necessary in order to be able to extract useful features to represents attacks. The data mining [16] process is also a very important step in the KDD process and has been getting the most attention in the literature. Several algorithms are particularly relevant for the field of intrusion detection:

- Sequence analysis, models sequential patterns and these algorithms can discover what time based events are frequently occurring together.
- Link analysis, determines the relations between fields on the database records.
- Classification, classifies the data into one of several predefined categories. The algorithms normally output "classifiers" in the form of sets, e.g. decision trees or rules.

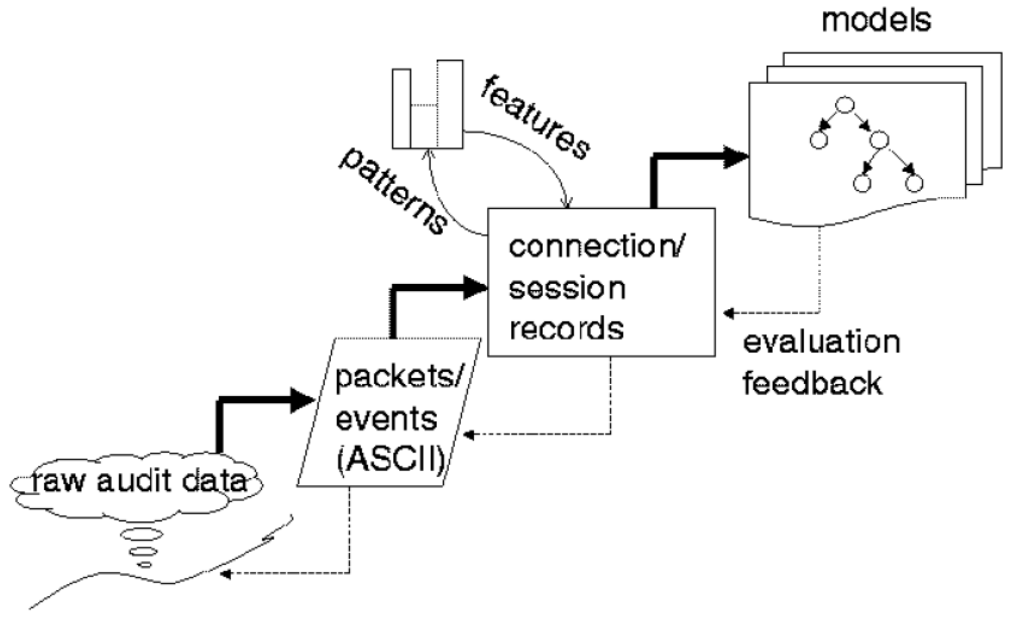


Figure 1: Feature definition/extraction procedure [12]

As it is described in [12], Figure 1, the feature definition/extracting procedure is processing raw binary audit data into ASCII network packet information. This is in turn summarized into connection records, containing a number of connection properties, e.g. *service, duration, flags* etc. Data mining programs are then applied to the connection records to compute the frequent patterns, which are analyzed to construct additional features for the connection records. Then a classification program is used to learn the detection models. This last process is of course iterative.

In this thesis, we focus on the data preparation and establishing connection records from these packet events. We focus our attention on the three first steps of the data mining framework in building the prototype. After we have generated a data set and begin to analyze the results, our focus shifts to the last two steps of the process.

2.2.1 Network connection records

In this section we give a brief review of connection records, all the terms used here will be explained more thorough in the experimental work section. A connection record contains information about one specific connection between two hosts. This connection (or host session record) can be uniquely identified by the combination of its *time* (start time), *src_host*, *src_port*, *dst_host*, and *service* (destination port) [17]. The *src_host* and *src_port* contain information about the host that initiated the connection, and the responder host is identified by the *dst_host* and *service* fields. These attributes are essential for network packets, and they are extracted from the packet headers. In addition to these, other attributes are needed for doing a network analysis. These attributes are called “intrinsic” features, and examples of these are: *src_host*, *src_port*, *dst_host*, *service*, *duration*, *src_bytes* and *dst_bytes*. The content of the network packets, which are sent in both directions between the originator (src_host) and responder host (dst_host), are also stored in the connection record.

Features of a connection are extracted/computed from the connection records to create additional attributes for each connection. These are computed from the “intrinsic” features of a connection or by the payload to the packets that belong to the connection, more details in Section 4.1.

3 Teoretical background

3.1 Honeypot

Honeypots are decoys that serve several purposes. They can distract attackers from valuable machines on a network, can provide early warning about new attack and exploitation trends or allow an in-dept analysis of computer attacks and methods. To get early warning about new vulnerabilities we can install a honeypot. Every attempt to contact this system via the network is suspicious, and if the honeypot is compromised we can study the vulnerability that was used to compromise it.

In order to make the honeypot an attractive target, it can be configured to run any operating system and any number of services. There are two kinds of honeypots, physical and virtual. Virtual honeypots are the preferred choice, since it is possible to populate a network with hosts running numerous operating systems on one physical machine. To convince the attackers that the honeypot is running a given operating system, it needs to simulate the *TCP/IP* stack of the target operating system. This has to be done with care, in order to mislead *TCP/IP* stack fingerprint tools like *Nmap* [18].

Honeypots are divided into two general categories:

- Production honeypots help mitigate risks and are typically implemented to aid in detecting computer attacks against the organization.
- Research honeypots are used to gain information about computer attacks and to analyze the methods the attacker's use, like the identity of the attackers and what kind of tools they use.

Based on the level of interaction, honeypots can be classified by one of three categories [19]:

- Low interaction is used primarily in production honeypots. They are easy to install and can emulate very few services. An example of a low interaction honeypot is *honeyd* [20].
- Medium interaction offers the attackers more ability to interact than the low, but they have less functionality than the high interaction. They are designed to give certain predefined responses to a number of activities and provide more information about the attacker than the low interaction honeypot.
- High interaction is time consuming to build and maintain with the highest level of risk. The goal is to give the attacker access to real operating systems, where nothing is emulated or restricted. As a result, more information about computer attacks is gathered than by implementing a medium or low interaction honeypot.

3.1.1 *Honeyd*

In this thesis, we use *honeyd* [21] to simulate a network. It is less complicated to build and maintain and is useful for gather information about network probes or worm activity. Honeyd is designed to reply to network packets, whose destination *IP* address belongs to one of the simulated hosts. It can only process three of the major Internet protocols:

ICMP, *TCP* and *UDP*. Packets containing other protocols are logged and discarded. The personality engine makes the network stack of the honeypot behave as specified for any given operating system. This is achieved by simulating the behavior by changing the protocol headers of every outgoing packet to match the characteristics of the configured operating system. For each virtual operating system hosted by the honeypot, different *TCP/IP* services can be configured to be offered to an attacker. These services are emulated by scripts that listen on specified ports and interact with attackers in a predefined manner.

3.1.2 Configuring *honeyd*

In *honeyd* a virtual host is configured by creating a template of the given operating system in the configuration file called *honeyd.conf* [22]. This file defines the characteristics of the simulated host including the operating system, the port it listens to and the behavior of the emulated services. Each template is given a name and a new template is created by using the **create** command. The **set** command assigns a personality to the template; `block`, `reset` or `open` that defines the default behavior for the network protocols. When using the `block` keyword, all packets for the specified protocol are dropped by default. `Reset` indicates that all ports are closed by default for the specified protocol. `Open` means that all ports are open for the given protocol. By using the **add** command, an emulated service can be added to the template with the behavior predefined by the given script. The **bind** command is used to assign an IP address to the template.

3.1.3 Honeypots deployment and setup

In this section, we describe the setup of the honeypots we used to gather traffic for the prototype and experiments in this thesis. We used 4 computers in deploying the honeypots. In addition, we obtained 8 valid IP addresses assigned to an IDS segment, which is directly connected to the ISP of the institution [23] in other words no traffic is blocked by the institution's firewall. In order to be able to capture packets and minimize the risk of an attacker detecting it, we use transparent bridges in front of the computers running *honeyd*. This is realized by installing *OpenBSD* [24] and using the built-in bridging functionality [25]. By enabling the bridging functionality, we also had to enable packet filtering on the *OpenBSD* hosts. We created a very simple packet filter that allowed all inbound and outbound traffic to pass through the bridge. All traffic passing through the bridges is logged by *tcpdump* [3]. The network setup is presented in Figure 2.

In theory, all traffic to a honeypot is regarded as suspicious traffic, but we cannot conclude that all traffic is malicious. We need to identify the network packets that can constitute an attack. For this purpose we use an IDS to automatically report possible malicious packets, to the best of the selected system's knowledge. The traffic can contain other attacks that the IDS fails to identify, but given the volume of traffic it is very cumbersome to identify the attacks manually or by some other method.

3.2 Malicious traffic generator

In order to get different types of attack we need to use malicious traffic. There exist several vulnerability scanners that can be used for this purpose. One of the most popular scanners is *Nessus*. Another program is *the Metasploit framework* [26], which is a research project for developing, testing, and using exploit code. This program was difficult to use and we had problems with making the exploit work and capturing the packets. For this

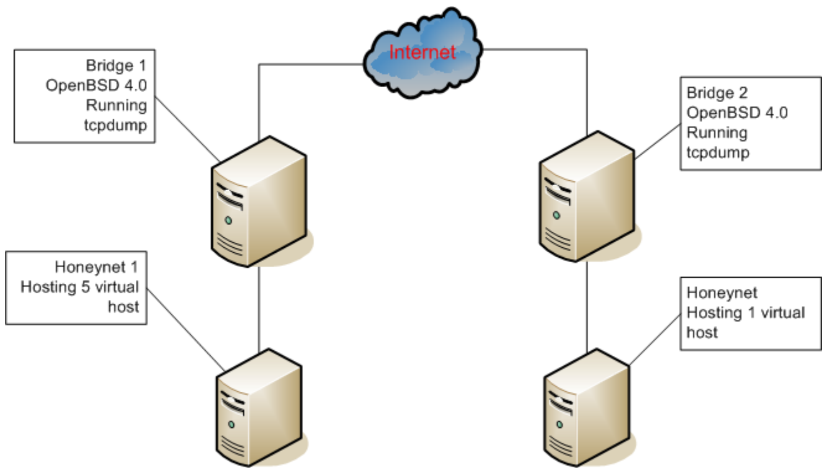


Figure 2: Honeynet deployment and network packet gathering infrastructure

thesis we have concentrated on using *Nessus* and a selection of computer attacks from it.

3.2.1 *Nessus*

In order to control the number of attacks and the attack types, we need a vulnerability scanner. *Nessus* [27] is a popular network vulnerability scanner that can be used as a penetration testing tool. It contains a number of plugins to generate different types of malicious traffic against the host to be scanned. This makes it easy to reproduce the traffic flow and it has the ability to automate the process in a simple manner. It depends on *Nmap* and performs various checks to determine which services and software packages are running on a remote host. By determining the versions of the software in question, it can determine if they are subject to known vulnerabilities. After *Nessus* is finished with scanning the remote host, it produces a report of its findings and explains what can be done to fix possible vulnerabilities.

The *Nessus* architecture consists of a client and a server. The server launches all the scans, while the client controls the process and views the report. This makes *Nessus* flexible and useful. Since the IDS segment is totally separated from the rest of the network in this organization, we need to either be on the same segment as they are or from a connection outside the organization in order to contact the honeypots. With *Nessus* both options are possible, due to the client-server architecture.

3.3 Network Protocol Analyzer and Editor

The captured network traffic is stored in a binary *tcpdump* file. All the information about network packets and payload are stored in this manner, and we need to parse this file in order to extract useful and valuable information. Reading a binary file containing network packets with different protocols is cumbersome, and to reduce the workload we need a network protocol analyzer. *Wireshark* [28] is such a tool and it is based on the source code from *Ethereal* [29].

3.3.1 *Wireshark*

Wireshark is a free network protocol analyzer for *Unix* and *Windows* systems. It can also be used as a packet sniffing application, and it uses the *pcap* library to capture packets.

In addition to capture live traffic, *Wireshark* can read many different capture file formats. *Wireshark* can capture data from numerous physical media and many network protocols are supported, including decryption support for some protocols e.g. *IPsec*, *SSL/TLS* , etc. It can also export network packets to *XML (PDML)*, *PostScript*, *CVS* or plain text files. This feature is necessary for the prototype, as it reads an *XML* file containing all the packets.

In addition to exporting a *tcpdump* file, other features of *Wireshark* are used to process the network packets. We have modified the protocol dissectors of *Wireshark* to only decode the following protocols:

- Ethernet, Protocol for the physical protocol of the transmission medium, needed for decoding higher layer protocols.
- ICMP, Internet Control Message Protocol.
- IP, Internet Protocol.
- TCP, Transmission Control Protocol.
- UDP, User Datagram Protocol.

In this thesis, we focus on these protocols and the prototype is designed with these protocols in mind, since these are a selection of the often used protocols for launching network computer attacks. Information about other protocols will not be extracted in the prototype, and therefore these protocols are not decoded in *Wireshark*.

Wireshark has also a method of filtering out packets we do not need, and this is useful in removing packets that do not belong to any connection. These packets are not included in the data set, since many of the extracted features are based on different properties of a connection. Unwanted packets must be filtered out manually and this task can be time consuming, if the number is relatively large. *Wireshark* seems to have a limitation on the length of the filtering string and we were not able to remove more than 140 packets simultaneously. To remove all unwanted packets we had to run *Wireshark* multiple times and remove 140 packets at the time. The prototype identified all packets not belonging to any connection and created a file with the filtering string for each time we had to run *Wireshark*. After all unwanted packets were removed from the tcpdump file, we exported the file to the *PDML* format and executed the prototype to verify that all unwanted packets were removed and no other packets had been removed.

Another feature of *Wireshark* we need in order to construct the data for the prototype, is the ability to merge *tcpdump* files together to one file. This allows us to mix different log files and files containing computer attacks that we have launched against the honeypots. The network packets are merged chronologically by the individual timestamp and stored in a new file.

3.4 Network Intrusion Detection System

In order to test the data set produced from the processing of captured data from the honeypots and running vulnerability scanner, we need an IDS. In this thesis, we have chosen *Snort* [30]. It can also be used in identifying computer attacks launched against the honeypots and reduce the time spent on investigating the gathered network flow.

3.4.1 *Snort*

Snort is an open source network IDS, first published in 1998 and has become one of the widest deployed IDS. It is mainly a misuse detection system, but it also has some anomaly detection capabilities. The signatures required by *Snort* are expressed by a lightweight rule description language, which is easy to understand and allow users to modify or create customized rules. New rule sets are provided by an update service and to get the freshest rules users need to subscribe. *Snort* is capable of monitoring 100Mbit networks, but experience packet loss on higher bandwidth network links. To reduce packet loss on higher bandwidth networks *Snort* can be deployed using a distributed architecture. By deploying multiple *Snort* sensors and configuring these to only look at portions of the traffic flow through a high speed link, together they can monitor the entire network traffic [31].

Snort has the ability to read from captured network flow and identify possible computer attacks in the provided *tcpdump* file. This feature is needed for analyzing and testing the generated data set in this thesis. The log file containing the findings from *Snort* is compared against the information in the data set, in order to determine the characteristics of the data set.

3.5 Selection of features for IDS benchmarking

As mentioned in previous work, the KDD 99 benchmarking data set contains 41 features for each connection. These features are grouped into four categories.

3.5.1 Basic features of individual *TCP* connections

Duration

The duration of the connection measured in seconds, continuous feature. Calculated from the time elapsed between the last and the first packet in the connection.

Protocol type

This feature indicates the type of transport protocol used in the connection, e.g. *TCP*, *UDP*.

Service

It indicates the destination service for the connection, e.g. 80 = HTTP or 22 = SSH.

Flag

Status flag of the connection. Indicates in which state the connection was ended.

Source bytes

Number of bytes sent from the source to the destination host, by summarizing all packets sent from the source.

Destination bytes

Number of bytes sent from the destination to the source host of the connection.

Land

Indicate if land attack is attempted. 1 if connection is from/to the same host or port, 0 otherwise.

Wrong fragments

The number of wrong fragments in the connection.

Urgent

The number of urgent packets in the connection.

3.5.2 Content features within a connection suggested by domain knowledge

Hot

Number of “hot” indicators. We have omitted all of the content features found in the KDD CUP 99 data set in our prototype, since these indicators are rather old and are not relevant any more.

Failed logins

The number of failed login attempts.

Logged in

Indicates if the connection was a successful login, 1 if successfully logged in or 0 otherwise.

Number of compromised

The number of “compromised” conditions.

Root shell

Indicates if root shell is obtained, 1 if true otherwise 0.

Su attempted

Indicates if the command “su root” is attempted in the connection, 1 if true otherwise 0.

Number of root accesses

A counter of the number of root accesses by the duration of the connection.

Number of file creations

The number of file creation operations by the duration of the connection.

Number of access files

The number of operations on access control files.

Number of outbound cmds

The number of outbound commands in an ftp session, given that the connection is an ftp session.

Is hot login

Indicates if the login belongs to a defined “hot” list, 1 if true otherwise 0.

Is guest login

Indicates if the user attempting to login is using a guest account, 1 if that is the case otherwise 0.

3.5.3 Time based traffic features computed using a two second time window

Count

The number of connections to the same host as the current connection in the past two seconds.

Srv count

The number of connections to the same service as the current connections in the past two seconds.

Serror rate

The rate of connections to the same host as the current connection in the past two seconds that have “SYN” errors.

Srv serror rate

The rate of connections to the same service as the current connections in the past two seconds that have “SYN” errors.

Rerror rate

Same as with “Serror rate” only with “REJ” errors instead of “SYN.”

Srv rerror rate

Same as with “Srv serror rate” only with “REJ” errors instead of “SYN.”

Same srv rate

The rate of connections to the same service in the past two seconds as the current connection.

Diff srv rate

The rate of connections to different services in the past two seconds as the current connection.

Srv diff host rate

The rate of connections to different hosts in the past two seconds using the same service as the current connection.

3.5.4 Host based traffic features computed using a window of 100 connections

Dst host count

The number of connections to the same destination host.

Dst host srv count

The number of connections to the same destination host using the same service.

Dst host same srv rate

The rate of connections to the same destination host using the same service.

Dst host diff srv rate

The rate of connections to different services on the current destination host.

Dst host same src port rate

The rate of connections to the current destination host, having the same source port.

Dst host srv diff host rate

The rate of connections to the same service coming from different hosts.

Dst host serror rate

The rate of connections to the current destination host that have an “S0” error.

Dst host srv serror rate

The rate of connections to the current destination host and the specified service that have an S0 error.

Dst host rerror rate

The rate of connections to the current destination host that have an RST error.

Dst host srv rerror rate
The rate of connections to the current destination host and with the specified service that have an RST error.

In this thesis, we concentrate on extracting most of the features found as basic features of individual *TCP* connections, and a selection of traffic features. The selection of have taken into considerations the work done by [32], were they look into the relevance of the features found in the KDD Cup 99 data set. In addition, we have created three new features that may be relevant for detecting network attacks. These are content features that are relevant for detecting newer attacks, than the attacks found in the KDD data set. The following features are extracted from each connection by the prototype:

- Duration, basic feature of time elapsed for the whole connection.
- Protocol, either: *ICMP*, *TCP* or *UDP*.
- Service, port number of the responding host in the connection.
- Src_Bytes, number of bytes, including header information, received by the responding host.
- Dst_Bytes, number of bytes, including header information, received by the originating host in the connection.
- Land, indicates if the source and destination host are the same or the port numbers are the same. This is used to create Denial of Service attack, called the Land attack.
- Urgent, number of packets in the connection with the urgent flag enabled.
- Access to passwd attempted, new content feature indicating if someone is attempting to request the *passwd* file.
- Cross site scripting attempted, new content feature indicating if cross site scripting attack is attempted.
- Directory traversal attempted, new content feature indicating if a user attempt to traverse directories on a remote host.
- Count, traffic feature indicating the number of connections to the same service as the current connection in the past two seconds.
- Same_srv_rate, rate of connections to the same service with the same host as the current connection in the past two seconds.
- Diff_srv_rate, rate of connections to different service with the same host as the current connection in the past two seconds.
- Srv_count, count of connections to the same service with the same host as the current connection in the past two seconds
- Srv_diff_host_rate, rate of connections to different host and with the same service as the current connection in the two past seconds.

Most of these features are the same as found in the KDD CUP 99 data set, but we have also added three new content features. Many of the content features found in the KDD data set are relevant for Unix systems and some of the features are difficult to determine, the content features used in the KDD CUP data set. Therefore we have excluded all content features found in KDD data set, and replaced them with three new ones. The new content features are designed with an idea of aiding to identify newer attacks found in today's network computer attacks [33]. These content features are selected based on statistics related to attacks captured by the honeypot, as well as having in mind the current hacker trends [34].

We have focused on attacks directed against web applications, which is one of the types of attacks we have obtained by deploying the honeypots. In addition to the captured attacks, we have added newer attacks on web applications. These have been chosen based on the findings from reports, such as [33, 34, 35], and the possibility of launching the attack against our honeypots. We have added cross site scripting attacks, which is one of the often used attacks against web servers according to the papers. In an attempt to identify the connections containing these attacks, we have added the "Cross site scripting attempt" features. The "Directory traversal attempt" feature has been added to identify the connections, which contains the "HTTP directory traversal" attack captured by the honeypots. The last feature has been added for detection attacks that tries to access the *passwd* file in the /**etc** directory. This attack, if successfully executed, allows an attacker to access the *passwd* file, which in turn can possible compromise user accounts on the attacked system. More details on these three new content features are provided in Section 4.1.3.

4 Experimental work

In order to process network logs, containing traffic flow to and from the honeypots, we developed a prototype for extracting useful information from individual network packets. Such a framework is essential to perform data mining techniques and experiments, using the processed network packets. The main idea was to develop a prototype to use as a framework for extracting features. It can also be used as a basis for generating frequent episodes and generalized episodes [36, 37, 38] from network connections, by looking into sequence of events.

4.1 Prototype

The prototype was developed for doing preprocessing of network packets, establishing connection records based on the previous preprocessing and calculating features for each connection to be included in the data set. The data set produced by the prototype also contains a label, to indicate if the connection contains an attack or is part of the normal traffic.

4.1.1 Preprocessing

The preprocessing of network packets, include extracting relevant and useful information about network packets. Before the prototype can perform this task, we must export the binary *tcpdump* file containing all the network packets to another format. The prototype we have developed reads the packets from a *PDML* file, which stands for *Packet Details Markup Language* and is an *XML* file containing the most important information about the fields and protocols in a packet. This file is created by *Wireshark* (for more information see the Section 3.3.1.) The prototype reads the provided *PDML* file and depending on the protocols in each packet different information is extracted.

Generic information for all packets

All the packets captured by *tcpdump* contain some general fields in the *PDML* file. From this information, some fields are extracted in order to identify each individual packet:

- Packet number. Used as an ID for each packet, a consecutive number generated by *Wireshark*.
- Size. Size of the captured packet.
- Timestamp. Time of capture for the packet. Timestamp of when the packet was captured by *tcpdump* running on the *OpenBSD* bridges in front of the honeypots.
- Protocols. Name of all the protocols in the packet, used for identifying protocols and extracting the appropriate information.

Depending on the protocols in the packet the appropriate information is extracted.

Internet Protocol

The Internet Protocol version 4 is defined in RFC 791[39] and has a minimum size of 20 octets, or 160 bits. The protocol header has the following fields:

Version

Indicates the version number, which is either 0100 for IPv4 or 0110 for IPv6. This information is not extracted as it does not provide any information in terms of detection of network computer attacks.

Internet header length

Length of header in 32 bits words. According to the specification the minimum value is five, for a minimum header length of 20 octets.

Type of service

It is a field of eight bits to provide guidance to end system and routers along the path, in selection of the actual service parameters.

Total length

This 16 bits filed indicates the total length of the *IP* packet including the header length.

Identification

A 16 bits field containing a sequence number that, together with the source address, destination address and user protocol is intended to uniquely identify a packet.

Flags

Only two bits of this three bit field are currently defined. When a packet is fragmented, the More bit indicates whether this is the last fragment in the original packet. The Don't Fragment bit prohibits fragmentation when set. This bit may be useful if it is known that the destination host does not have the capabilities to reassemble fragments. By setting this bit and the packet exceeds the maximum size permited in the network, the packet will be discarded.

Fragment offset

It is a 13 bits field that indicates where in the original packet this fragment belongs, measured in 64-bit units. This implies that fragments other than the last fragment must contain a data field that is a multiple of 64 bits in length.

Time to live

The eight bits field that specifies how long, in seconds, the packet in question is allowed to live. Every router that processes the packet must decrease this counter by at least one, so it is quite similar to a hop count.

Protocol

This eight bits field indicates which higher level protocol follows in the packet after the *IP* header.

Header checksum

The 16 bits field that contains output from an error detection code performed on the header.

Source address

This 32 bits field identifies the source of the *IP* datagram.

Destination address

This 32 bits field identifies the destination of the *IP* datagram.

Options

This is a variable size field to encode options requested by the sending user.

Padding

Variable size field used to ensure that the packet header is a multiple of 32 bits in length.

User Datagram Protocol

The User Datagram Protocol is defined in RFC 768, [40], and is designed to allow applications to exchange messages over a packet-switched computer network using a minimum number of protocol mechanisms. This protocol is connection-less, meaning no connection is initiated before the transfer of information begins. The protocol header contains the following fields:

Source port

It is a 16 bits field that identifies the source port. This field is not required, but in order to be able to receive any meaningful message it must the set. If it is not used, it must be set to zero.

Destination port

This is a 16 bits field that identifies the destination port and it is required to be set.

Length

It is a 16 bits field that specifies the length of the whole datagram in bytes. The minimum length is eight bytes, which is the minimum size of the header. The maximum theoretical length of an *UDP* datagram is 65 527 bytes.

Checksum

This is a 16 bits field used for error-checking of both the header and data of the *UDP* datagram.

All these fields are extracted from the packet, in case of a *UDP* datagram. If the datagram contains a payload, this is also extracted.

Transmission Control Protocol

The Transmission Control Protocol is defined in RFC 793 [41], and is one of the core protocols of the internet suite. The protocol is connection oriented, unlike its counterpart *UDP*, and a connection must be initiated before any data can be transmitted across the network. It supports most of the popular internet application protocols and it is also widely used for transfer of information between applications. Because *TCP* supports many application protocols and only one header version is defined, the header must include all the protocol mechanisms needed. This leads to the header being rather large, with a minimum length of 20 octets, 160 bits. The header fields are the following:

Source port

This is a 16 bits field that identifies the source port on the source host of the datagram.

Destination port

It is a 16 bits field similar to the source port, only identifying the destination port. This field is equal to the service on the destination host.

Sequence number

This is a 32 bits field with a dual role. If the SYN bit is set the initial sequence number and the first data byte is the sequence number plus one, otherwise the sequence number is the first data octet in this datagram.

Acknowledgment number

It is a 32 bits field with the sequence number of the next data octet the sender of the datagram expects to receive next. The acknowledgment flag must be set if the datagram contains an acknowledgment number.

Data offset

This four bits field specifies the length of the *TCP* header in 32 bits word. The minimum length of the header is five, giving a header length of 20 bytes.

Reserved

It is a six bits field reserved for future use, should be set to zero.

Flags

This is a six bits flags vector.

- URG: Urgent pointer field is significant.
- ACK: Acknowledgement field is significant.
- PSH: Push function.
- RST: Reset the connection.
- SYN: Synchronize the sequence numbers.
- FIN: No more data from the sender.

Window

This is a 16 bits field used for flow control. It contains the number of data octets, bytes, beginning with the one indicated in the acknowledgment field that the sender is willing to accept.

Checksum

It is a 16 bits field that contains a checksum for both the header and data used for error checking.

Urgent pointer

This is a 16 bits field that points to the last octet in a sequence of urgent data, allows the receiver to know the amount of urgent data.

Options

It is a field of variable size with different kinds of options. The length of this field must be a multiple of 32 bits.

All the previous fields are extracted from the packet descriptions, except Urgent pointer and Options. In addition the data, or payload, of the *TCP* datagram is extracted.

Internet Control Message Protocol

The Internet Control Message Protocol differs from both *TCP* and *UDP*, since it is not designed to transfer messages between user applications in the same manner as the other protocols. It is designed in mind of providing feedback about problems in the communication environment. *ICMP* messages are typically used to report errors in processing of *IP* datagrams. The protocol is defined in RFC 792 [42].

ICMP messages are sent using the basic *IP* header. The message format of the protocol is added to the data portion of the *IP* datagram. Depending on the message type, different message format is added to the payload of the *IP* datagram. The first field in the message format is the *ICMP* Type. This value determines the remaining of the message format. Usually the message format also contains a Code field to indicate a more accurate description of the *ICMP* Type. Depending on the type of *ICMP* message the following information is extracted:

- Type, what kind of message this is.
- Code, a more detailed description of the message type.
- Identifier, in aid of matching echos and replies e.g. ping.
- Data, payload of the *ICMP* message.

Some messages are responses to connection attempts made by other host. In these cases the payload of the message includes other protocols, like *TCP* or *UDP*. The prototype also extracts the necessary information about the protocol in the payload, for later identification of the right connection.

4.1.2 Establishing connection records

After all the packets have been processed and the appropriate information have been extracted, packets are gathered into connection records. This process involves checking each packet against all others to see if there are packets that belong to the same connection. The conditions for a packet to be in the same connection as others: the source and destination host must be the same and the source and destination port must be the same in case of *TCP* or *UDP* packets. A connection must contain packets in both directions, meaning each host must at least have sent one packet to the other. For *TCP* connection the next packet in the connection must have been received before a defined timeout expires. The value used is as defined in the *TCP* protocol, which should be 2 times MSL (maximum segment lifetime). On most systems MSL is 120 seconds, resulting in a timeout value of 240 seconds. Only *TCP* connections have this defined timeout, and this is not taken into account when establishing connection records for other protocols. Each connection has the following properties:

- The start time of the connection.
- The end time of the connection.
- Originating host. The host that initiated the connection.
- Originating port. The port on host that initiated the connection.
- Responding host. The host that responded to the connection.
- Service. The port on the host that responded to the connection.
- All the captured packets in the connection.

The process of creating connection records is a time consuming task that grows significantly by the number of packets needed to be examined. Therefore the prototype can load a previously saved file, which contains packets, connection records and information about attacks to label the data set.

4.1.3 Computing features for each connection

After all the packets have been gathered into connection records, the prototype can compute different features for each connection. As mentioned in Section 3.5, most of the features are related to properties of a basic *TCP* connection. Some traffic features are extracted and we have added three new content features. Since the KDD CUP 99 data set is based on attacks from the Lincoln Laboratory IDS evaluation performed in 1998 and 1999, the attacks found in this set are well known and not so much in use today. Therefore we have replaced the content features found in the KDD CUP 99 data set with three new ones, which may be more appropriate for identifying newer attack types. These content features are determined by decoding the payload of each packet in a connection. Since each payload is represented using hex numbers, the information must first be transformed to the equivalent ASCII character, if the hex value is a valid character, before it can be examined. All these new features are limited to web attacks, meaning the attack is against a host running a web server. In this thesis, all web traffic uses port 80 for the communication.

Access to passwd attempted

As mentioned in Section 3.5, this feature indicates if someone is attempting to access the passwd file by sending a request to a web server of the remote host. This feature is relevant for *Unix* based operating systems, which can store the password file in the directory **/etc**. All *Unix* based systems running a web server can be vulnerable to this attack. A positive value indicates that the payload of a packet in the connection, with the destination port 80, contains the following request: "/etc/passwd".

Cross site scripting attempted

This feature is used to indicate if someone is attempting to use cross site scripting techniques, in an attempt to inject code of their choice. By sending a specially crafted input to a web application that accepts this without filtering and use it as part of the *HTML* of a new page, an attacker can exploit this vulnerability by cross site scripting attack. The input can be designed in many different ways, but we have focused on one of the simplest types. A positive value indicates that the payload of a packet in the connection, with destination port 80, contains the following characters: "<script". Many older web servers and various web applications are vulnerable to this attack, and the attack is relative simple to launch as automated exploit code exists. This feature can generate false positives, if a web application is designed to accept input that contain scripts. False positives are not generated in this thesis, as all connections with a positive value do contain an attack.

Directory traversal attempted

This is the last content feature in the data set, it indicates if someone is attempting to traverse directories by sending a request in the payload of a packet. This technique can also be part of a lager attack, e.g. someone attempting to obtain a native shell by using directory traversal vectors. A positive value indicates that a packet in a connection, with destination port 80, contains characters for traversing directories, e.g. "..\" or "../". These attack vectors are very simple to launch and do not require any exploit software. Bad links in web pages can contain such a sequence, and can therefore generate false positives with this feature. All the connections in our data sets, which contain a positive value for this feature, contain some sort of attack vectors to attempt directory traversal and therefore we have no false positives concerning this feature.

In addition, 12 other features are extracted as listed in Section 3.5, of the total 15 extracted features for any given connection.

4.1.4 Identifying attacks

The task of identifying all attacks in the captured traffic log is cumbersome. This can be done manually, by inspecting all the individual packets gathered from the honeypots. The other option is to use an IDS, like *Snort*, to inspect the log file and use the classification of attacks that it produces as a means of identification. We have chosen the latter option, since it is less time consuming and manually inspecting packets is not really an option when we have such high amounts of packets to investigate. To produce the alert file from *Snort*, we have used the official *Snort* rule set for registered user as 10.01.2007 and the following options:

```
Snort.exe -e -l c:\Snort\log\<directory to save log>
-c c:\Snort\etc\snort.conf -r <tcpdumpfile to replay>
```

For identifying connections that contain computer attacks, the alert file is processed and necessary information for uniquely identifying the network packet that contains the attack is extracted. This information is stored in an attack file, where we also can add new attacks if needed. The prototype processes the attack file to find the time of attacks. The timestamp of the attack is used together with other relevant information from the alert. To successfully identify an attack in a connection, several conditions must be fulfilled. The first one is related to time of the attack. Since there are no discrepancies between timestamps of packets and alert time, the packet in the connection has to have the same timestamp as the alert. All the packets in the connection that match the timestamp of the alert and have the same source and destination address as the attack will be further analyzed. There are several other factors that identify the right packet:

- Sequence and Acknowledgement number, in case of *TCP* packet.
- Source and Destination ports.
- ICMP Code and Type.

In order to identify an attack, the timestamp, source and destination and one of the factors in the previous list must match with a packet. Since a connection consists of several packets, in theory one connection can contain several attacks of the same type. We do not distinguish on the number of attacks in a connection, only the classification of the connection is important in this thesis.

4.1.5 Generating the data set

After packets have been gathered into connection records, the prototype can generate the data set. This process involves identifying all the attacks found in the connections and calculate features for each connection. Each of the connections obtains a label indicating if it is a part of the normal traffic or it contains one or more packets that is an attack. The data set is stored on a text file for further analysis.

4.2 Program for comparing *Snort* alerts with attacks found in the data set

We have also developed a small program to aid in the experiments on the generated data set. The purpose of the program is to compare the alerts generated by *Snort* when processing the *tcpdump* file containing the data set, with the attacks we have identified. This way we can determine the detection rate to *Snort* when using the benchmarking data set. *Snort* stores the alert (alarms) in the form of an alert file, which is processed in the same way as with identifying attacks in the prototype. The program compares the time of the alert, reported by *Snort*, and finds the corresponding attack. Attacks that *Snort* fails to identify are stored in a file for manual analysis.

4.3 Experiments

The prototype generates a labeled data set, represented by the extracted features. This data set is based on the traffic gathered at the honeypots, but we have also added our own attacks to the data set. From analysis of the traffic captured by the honeypots, we found that it did not contain a broad range of attacks, and the attacks consisted of automatic scans. As a result, we have added some other types of attacks to generate a more versatile data set. We have used *Nessus* for this task and have generated two data sets, which are the basis for the experiments. One contains only the network packets and attacks gathered by the honeypots, while the other is a modified version where we have added other attacks to the data set.

4.3.1 Methodology

In order to analyze and test the generated data sets, we need to perform some experiments. We have concentrated on two methods of analyzing the data sets:

1. Compare the attacks found in the data sets with alarms from Snort to determine detection rates.
2. Analyze the three new content features, and their abilities to identify attacks.

The first method is an experiment for determining the detection rate of *Snort* on the data sets. Since the first data set only contains attacks already identified by *Snort*, we expect that all attacks are classified as attacks. In the modified data set we are interested in comparing the results from running *Snort* with the knowledge about the generated data set. The purpose of this experiment is to determine some properties of the generated data set in relation to benchmarking of IDS, in this case *Snort*.

The second part is more an analysis of the new features, and their relevance in detecting network attacks. Since these are new features, there is a need for determining their characteristics. By determining their characteristic, we can determine if they are useful content features to identify attack. The main goal is to analyze which attacks can be identified by these features, and determine a value for the detection rate of these attacks compared to normal traffic.

4.3.2 The attacks found in the data sets

In the data sets, several different attack types are included. Ideally the data set should only contain attacks captured by the transparent bridges in front of the honeypots. After analyzing the logged network traffic we decided to launch some additional attacks in order to get more attack types and improve the attack characteristics of the data set. Different attack types are needed for benchmarking network intrusion detection systems and to determine detection rate on the system in question.

Since we have deployed honeypots to gather network traffic, the number of different attack types is rather limited. The main reason is the use of low interaction honeypots that limits the possibility of capturing more sophisticated attacks. Most of the captured attacks are results of automatic scans, rather than designated attacks that target the honeypots directly. This is the reason for adding more attacks to the data set. These attacks have been mixed with captured traffic in the same time period, in order not to create any bias in the data set. Most of the attacks found in the data set have been classified according to the classification name *Snort* uses for the attack. The attacks can therefore have another name than the one used in this thesis. We have chosen to use the *Snort* classification name, because it simplifies the process of determining which of the attacks are detected when running *Snort* on the data set.

Authorization basic overflow attempt

This attack is an exploit attempt on a known vulnerability in the *Oracle* database server. There are multiple buffer overflows in the *XML* database functionality that may provide remote execution of arbitrary code with the user privileges as the user running the database. An alert is generated based on an attempt to exploit the *HTTP Basic Authorization* mechanism for *Oracle Web Services* by means of a buffer overflow. The exploit can allow local users to cause denial of service or hijack user sessions [43]. This attack has been added to the generated traffic and is only present in the modified data set.

Chunked-Encoding transfer attempt

This attach is directed at a web server, and exploiting a known vulnerability and possible gaining administrative access to the server. The exploits and vulnerabilities depend on the web server in question, but the basis for the attack is the same. Some web applications do not perform a stringent check when validating the credentials of the host connection to the web service. This can lead to unauthorized access and data stored on the web server can be compromised.

The honeypots are emulating virtual host running *Microsoft Internet Information Server*, so the actual exploit found in the data set is an attempt to cause a buffer overflow in the chunked encoding transfer mechanism. This can cause denial of service or allow an attacker to execute arbitrary code [44].

Cross site scripting attempt

This is an attack technique, rather than an attack on a known vulnerability. Cross site scripting targets users of a web site. Malicious code can be inadvertently executed by running scripts written by an attacker when users follow untrusted links on webpages, emails etc. User can also be vulnerable to cross site scripting when viewing dynamically created content based on some content provided by users. A successful attack can steal a user's cookie or session ID, which can be used to impersonate the user. The attacker may also get access to the information submitted on the targeted web site [45]. This attack has been added and is only present in the modified data set.

HTTP directory traversal

The purpose of this attack is to gain unauthorized access to information on a web server or web application. This is possible on applications that do not check that the path of the request is correct, and allowing a remote user to traverse directories. An attacker can use directory traversal technique to browse directories outside the designated directory, which can disclose useful information that may be used to further attack the host [46]. This attack is present in both data sets, while we have added more connections containing this attack in the modified data set.

ICMP Ping NMap

This attack is a probing scan originating from *Nmap*, using the *ICMP* protocol. The attack is used to gather information about host on a given network. This can be part of a full *Nmap* port scan, to determine what services are running [18]. This attack is present in both data sets.

ICMP Superscan echo

This attack is also a probing scan that originates from a *Windows* based network scanner from Foundstone named *Superscan*. By default configuration the scanner sends an *ICMP Echo Request* message before proceeding with the scan. This packet has a payload of eight bytes with the character zero. Other tools can use the same payload as *Superscan*, and can therefore be labeled as *Superscan*, but the classification name has very little impact on the general characteristics of the data set [47]. This attack is present in both data sets.

IIS SAM attempt

This attack is an attempt to access the *Windows Security Accounts Manager*, *SAM*, the password file, using a web request. The password file contains *Windows* login information, which is stored as *NTLM* or *LANMAN* hashes on *Windows NT, 2000 or XP* host. If an attacker is successful in obtaining the password file, the login information can be cracked in minutes or hours if the passwords are weak. Then the attacker can use the local administrator login to compromise the host further [48]. The attack is present in both data sets.

IIS view source via translate header

The attacker crafts a special URL containing the keyword “Translate: f” in an attempt to view the source of the requested file. The affected system is *IIS version 5* that allows remote attackers to view source code of files and other scripts that normally is not available for viewing. Some *Microsoft* applications use the “Translate: f” header, but since we do not use such application there are no false positives relating to this attack [49]. This attack is present in both data sets and we have added some more occurences in the modified set.

MS SQL worm propagation attempt

This attack is a part of the “Slammer” worm, with the goal to compromise *MS SQL Server 2000* or *Microsoft Desktop Engine* [50]. The worm contains self propagation code that attempts to exploit a vulnerability in the *Resolution Service* of the two products. The exploit consists of sending too many bytes in the version request, resulting in a buffer overflow and can allow an attacker to cause denial of service or execution of arbitrary code [51]. This attack is present in both data sets.

SYN Scan

SYN Scan is a probing attack to determine what kind of services are running on a remote host. It is one of the most popular *TCP* scanning techniques and relies on generating own *IP* packets and monitor the responses, rather than using the operating system network functions. The scan never opens a full *TCP* connection and closes the connection before the handshake is completed, therefore it is more stealthy than a connect scan. The downside of using a SYN scan is that it can lead to denial of service attack since it does not terminate the connection properly. The benefit is that it works well through firewalls and normally do not generate many alarms on IDS or firewall, as it looks like a failed connection attempt [52]. This attack had been added to the modified data set, and is only present there.

Traceroute ICMP

This is a part of the Traceroute attack, and contains the *ICMP* messages issued in the scan. The reason for splitting the attack is that *Snort* only detected the *ICMP* packets, and not the whole scan. For more details see Traceroute UDP. Both Traceroute connection types have been added, and are only present in the modified data set.

Traceroute UDP

This attack is used to determine the physical layout of network devices of an organization. The purpose is to determine the route taken by packets across the network. It uses *UDP* packets on *Unix* and *Linux* systems with modifying the Time to live value to determine which host it travels through. Traceroute can be used by hackers to map and identify a company's network architecture. This information can identify vulnerable nodes or hosts, which the attacker can exploit to gain access to the network [52].

Web /etc/passwd attempt

This attack is similar to the "IIS SAM", that an attacker attempts to access the passwd file located on *Unix* and *Linux* host in the **/etc** directory. If shadow password files are not enabled, an attacker can gain valid login information for the host in question by using cracking tools on the obtained passwd file. This attack has been added to the modified data set.

WebDAV search acces

This attack exploit vulnerabilities in the *Web Distributed Authoring and Versioning (WebDAV)* search of a misconfigured *IIS* with *Index Server* enabled. The attacker can receive a complete listing of the web server root directory, and can potentially launch a denial of service attack if the provided search string is too long [53]. This attack is present in both data sets.

4.3.3 Traffic captured by the honeypots

We have gathered traffic from both honeypots in the period from the 23th of February to the 27th of March 2007, in the form of a *tcpdump* file. There are approximately 177 thousand individual network packets with the following protocol hierarchy:

- Internet Protocol (100 %)
  - Transmission Control Protocol (79,73 %)
  - User Datagram Protocol (6,36 %)

- Internet Control Message Protocol (13,91 %)

There has been some preprocessing of the captured data, with removing packets that did not belong to any connection, as mentioned above. To perform the experiment, the binary *tcpdump* file is replayed by *Snort* to identify computer attacks, in the same manner as it would with processing live traffic. The result from *Snort* is compared with the knowledge of the attacks.

4.3.4 Attacks found in the captured network packets

The captured traffic has been analyzed automatically with *Snort* to identify all the attacks, to the best of our knowledge. There is no way to guarantee that we have identified all the possible attacks, since there can be never before seen attacks in the network logs. The attacks captured by the honeypots after removing unwanted packets, classified by *Snort* are presented in Table 1.

Attack name	**Attack type**	**Number of occurences**	**Number of attacks detected by Snort**	**DR**
Chunked-Encoding transfer attempt	U2R	1	1	100 %
HTTP directory traversal	Information disclosure	9	9	100 %
ICMP Ping NMap	Probing	8	8	100 %
ICMP Superscan echo	Probing	6	6	100 %
IIS SAM attempt	R2L	12	12	100 %
IIS view source via translate header	Information disclosure	13	13	100 %
MS SQL worm propagation attempt	U2R	765	765	100%
WebDAV search access	DOS	1	1	100 %

Table 1: Attacks found in the captured traffic

As seen in Table 1, all the attacks present in the data set are identified by *Snort*. This is as we expected, since we also use *Snort* to classify the network packets captured by the honeypots. We have used the same attack groups as the KDD CUP 99 data set, with one addition. None of the groups fit the class of attacks, which has the goal of gathering information on a remote system. This process is quite different from probing attack, whose goal is surveillance of a remote system. We have added *information disclosure* as its own group, where the goal is to attack a remote system in order to receive some information the attacker normally would not see or obtain. An example is to try to obtain the source code or files on a web server that are normally not available for public viewing (*IIS* view source via translate header.)

In addition to the attacks mentioned in Table 1, there are also some network packets that have truncated *TCP* options. It is the *Snort* decoder that generates these alerts on packets where a certain *TCP* option was set, but the corresponding length was not used or the reported length was incorrect. These alerts do not report an attack, and are therefore not included in the table. Snort can also be configured to not report this alert by modifying the “snort.conf” file and including this line: “config disable_tcpopt_alerts”.

4.3.5 The modified data set

This data set is based on the previous set, but we have added other attacks to make a benchmarking data set for IDS. Adding attacks was necessary as the captured network traffic did not contain as many attacks as we had hoped for. The time period is the same as the previous set, but the total number of packets is increased to 222 462 network packets, divided on the following protocols:

- Internet Protocol (100 %)
    - Transmission Control Protocol (83,83 %)
    - User Datagram Protocol (5,08 %)
    - Internet Control Message Protocol (11,09 %)

4.3.6 Attacks in the modified data set

In the modified, data set we have added additional attacks to the ones already present. These attacks have been added manually, without *Snort* first classifying the packets. Here we expect to find some discrepancy in the total number of attacks and the result from *Snort*, as a result of using *Nessus* to generate some of the additional attacks. The packets sent from *Nessus* may not in fact be the intended attack, but more like a probing attack if the vulnerability is present on the remote host(s). This fact is taken into consideration when analyzing the result from the experiment. As with the previous set, only packets that can be used in establishing a connection are added to the data set. The following attacks are present in the modified data set:

Attack name	**Attack type**	**Number of occurences**	**Number of attacks detected by Snort**	**DR**
Authorization basic overflow attempt	U2R, DOS	4	4	100 %
Chunked-Encoding transfer attempt	U2R	1	1	100 %
Cross site scripting attempt	Information disclosure	7	7	100 %
HTTP directory traversal	Information disclosure	14	14	100 %
ICMP Ping NMap	Probing	8	8	100 %
ICMP Superscan echo	Probing	6	6	100 %
IIS SAM attempt	R2L	12	12	100 %
IIS view source via translate header	Information disclosure	15	15	100 %
MS SQL worm propagation attempt	U2R	768	765	100 %
SYN Scan	Probing	22295	0	0 %
Traceroute ICMP	Probing	1	1	100 %
Traceroute UDP	Probing	8	0	0 %
Web /etc/passwd attempt	Information disclosure	7	7	100 %
WebDAV search access	DOS	1	1	100 %

Table 2: Attacks found in the modified data set

Only two attacks were not detected by *Snort*, when processing the data set. SYN scan as mentioned above, is a probing attack. To do a complete SYN Scan on a remote host, results in the creation of many individual *TCP* connections. Each port scanned has its own *TCP* connection, and in the data set we have scanned five hosts with this attack. Resulting in 22 295 connections being added to the data set. We have used the SYN Scan plugin in *Nessus* to perform this attack. This plugin has been removed in the new version of *Nessus*.

Traceroute UDP packets are also not detected by *Snort*. We used the traceroute plugin in *Nessus* to perform this attack. *Snort* only detected the *ICMP* packets issued by the scan, which is not directly a part of the scan.

4.3.7 The generated data set

The second part of the experiment is an analysis of the new content features we have added to the data set. In this analysis, we use the modified data set that contains several attack types, which are not present in the captured traffic from the honeypots. We have included a sample of the generated data set, but because of size limitations we had to split the data set into two tables, Table 3 and Table 4. As we can see from the tables each connection is given an ID, which is only the packet number assigned by *Wireshark* to the packet that initiated the connection. Packet numbers are assigned based on the timestamp of the packets, meaning the network packets are sorted chronologically. This ID is not part of the features of the data set, merely for identification purposes.

The modified data set consists of 42 841 connections constructed from 222 462 captured network packets. From all these connections, 23 144 have one or more packets that contain a specific network attack. Based on the modified data set, we have calculated some statistics for each of the features.

4.3.8 General statistics from the modified data set

Basic features

Duration

The longest duration found in the set is 2 599 376,16 seconds, with the *ICMP* protocol where there is no defined time out for the connection. The shortest duration is zero seconds, meaning the connection only lasted less than a millisecond. By summarizing all the durations, we calculated the average to be 6 404,7 seconds.

Protocol

There are three protocols in the data set spread on the following number of connections:

- ICMP with 5 233 connections, 12,2 % of the total number of connections.
- TCP with 32 145 connections, 75 % of the total number of connections.
- UDP with 5 463 connections, 12,8 % of the total number of connections.

Service

As described in the Section 3.5, this feature represents the port number on the host that responded to the connection attempt. In case of connections that use the *ICMP* protocol, they do not use port numbers, with the result of the service field being zero in the data set. The smallest value of the service field is one, while the largest recorded is 65 534. The most often used service port is 80, belonging to 4 716 connections.

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
1	1,12	TCP	80	752	1474	False	0	True	False	False
14	1,169	TCP	80	750	1474	False	0	True	False	True
19	1,197	TCP	80	744	1474	False	0	True	False	True
24	1,21	TCP	80	713	1474	False	0	True	False	True
53	1,12	TCP	80	758	1474	False	0	False	True	False
58	1,374	TCP	80	714	1474	False	0	False	True	False
63	1,008	TCP	80	810	1474	False	0	False	True	False
68	1,23	TCP	80	700	1474	False	0	False	True	False
73	1,372	TCP	80	717	1474	False	0	False	True	False
94	1,366	TCP	80	718	1474	False	0	False	True	False
99	1,072	TCP	80	714	1474	False	0	False	False	False
144	0,388	TCP	80	758	3068	False	0	False	False	False
171	1,372	TCP	80	1040	3068	False	0	False	False	True
198	1,302	TCP	80	1032	3068	False	0	False	False	True
217	0,368	TCP	80	758	3068	False	0	False	False	False
234	1,314	TCP	80	1030	3068	False	0	False	False	True
279	1,225	TCP	80	1024	3068	False	0	False	False	True
289	1,088	TCP	80	1069	3068	False	0	False	False	True
333	1,406	TCP	80	1208	3068	False	0	True	False	True
343	1,342	TCP	80	1083	3068	False	0	True	False	False
370	1,296	TCP	80	1031	3068	False	0	True	False	False
414	0,095	TCP	80	1632	3068	False	0	False	False	False
441	0,094	TCP	80	1632	3068	False	0	False	False	False
468	0,35	TCP	80	4508	3308	False	0	False	False	False
501	0,176	TCP	80	4508	3308	False	0	False	False	False
534	1,967	TCP	2967	372	360	False	0	False	False	False
546	3,372	UDP	10112	284	280	False	0	False	False	False
550	3,297	UDP	10112	284	280	False	0	False	False	False
554	0	UDP	10112	142	140	False	0	False	False	False
558	3,144	UDP	10112	284	280	False	0	False	False	False

Table 3: A smal sample of the generated data set, part 1

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
1	1	0	1	0	Web /etc/passwd attempt
1	1	0	1	0	Web /etc/passwd attempt
2	1	0	1	0,5	Web /etc/passwd attempt
3	1	0	2	0,333333333	Web /etc/passwd attempt
1	1	0	1	0	Cross site scripting attempt
2	1	0	2	0	Cross site scripting attempt
3	1	0	1	0,666666667	Cross site scripting attempt
4	1	0	2	0,5	Cross site scripting attempt
5	1	0	3	0,4	Cross site scripting attempt
6	1	0	3	0,5	Cross site scripting attempt
7	1	0	4	0,428571429	Cross site scripting attempt
1	1	0	1	0	IIS view source via translate header
1	1	0	1	0	HTTP directory traversal
2	1	0	2	0	HTTP directory traversal
2	1	0	1	0,5	IIS view source via translate header
3	1	0	2	0,333333333	HTTP directory traversal
1	1	0	1	0	HTTP directory traversal
2	1	0	2	0	HTTP directory traversal
1	1	0	1	0	Web /etc/passwd attempt
2	1	0	1	0,5	Web /etc/passwd attempt
3	1	0	2	0,333333333	Web /etc/passwd attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	normal
1	1	0	1	0	normal
2	1	0	2	0	normal
3	1	0	3	0	normal
4	1	0	4	0	normal

Table 4: A smal sample of the generated data set, part 2

Src_bytes

As described in Section 3.5, this feature is the number of bytes sent from the source host of the connection. The lowest value of bytes being sent from the originating host is 60 bytes, in 22 559 connections. The largest volume of bytes being sent is 74 204. The average source bytes for the data set are 212,9 bytes.

Dst_bytes

This feature is the opposite as Src_bytes, and is the number of bytes sent from the responder host of the connection. The lowest value of bytes sent from the responder host is 60 bytes, in total 23 073 connections. The largest value of bytes sent is 15 028, in two connections. The average destination bytes for the data set are 346,1 bytes.

Land

This feature is used to indicate if some packets in the connection contain the land attack. This attack was more popular before, and is not so relevant any more. since manufacturers of operating systems and other network equipment have patched vulnerable systems. This fact is also present in the data set, as only 11 connections have been labeled with true for the land feature. About half of these connections have been identified to belong to the Traceroute UDP attack, five connections. These connections contained *UDP* packets that originated from running the Traceroute plugin in *Nessus*.

Urgent

This is the last basic feature that is calculated for each connection. It indicates the number of packets in the connection with the Urgent pointer enabled. In the generated data set, no connections contained packets with the Urgent pointer enabled. This feature cannot be used for identifying any network attacks in this data set, since for all connections the value is zero.

Traffic features

Count

This feature is the number of connections to the same host as the current connection, calculated by using a time window of two seconds. The lowest value in the data set is one and the largest value is 3 290. The calculated average value is 1 634, 6.

Same_srv_rate

It is the rate of connections to the same service as the current connection in the past two seconds. The lowest value is 0,0015 and the highest value is one, with a calculated average of 0,47.

Diff_srv_rate

This feature is the rate of connections to different services in the past two seconds as the current connection. The lowest value is zero and the highest value is 0,998, with the calculated average of 0,530.

Srv_count

It is the number of connections to the same service as the current connection in the past two seconds. The lowest value is one and the highest value is 58, with an computed average of 3,1.

Srv_diff_host_rate

This feature is the rate of connections to different host in the past two seconds using the same service as the current connection. The lowest value is zero and the highest value is 0,994, with an computed average of 0,552.

All the previous features are the same features as found in the KDD CUP 99 data set. In the next section, we analyze the properties of the new content features, and try to determine if they can be used for identification of network attacks.

4.3.9 Statistics of the new content features

Access to passwd attempt

This feature indicates if one or more packets in the connection have attempted to access the *passwd* file. As with all the content features we have added in this thesis, they are based on traffic to a web server, connection with port 80 in the service field. In the modified data set, seven connections have been marked with true for this feature. All the other connections have the value false. "Access to passwd attempted" managed to identify all the occurrences of the attack with the classification: "Web /etc/passwd attempt."

Cross site scripting attempt

As with the previous feature, this is also based on web traffic. It indicates if one or more packets in the connection contain a cross site scripting attack. In the data set we find six occurrences of connections with a positive value, while all other connections have a negative value. The connections with a positive value have all been labeled as a "Cross site scripting attempt."

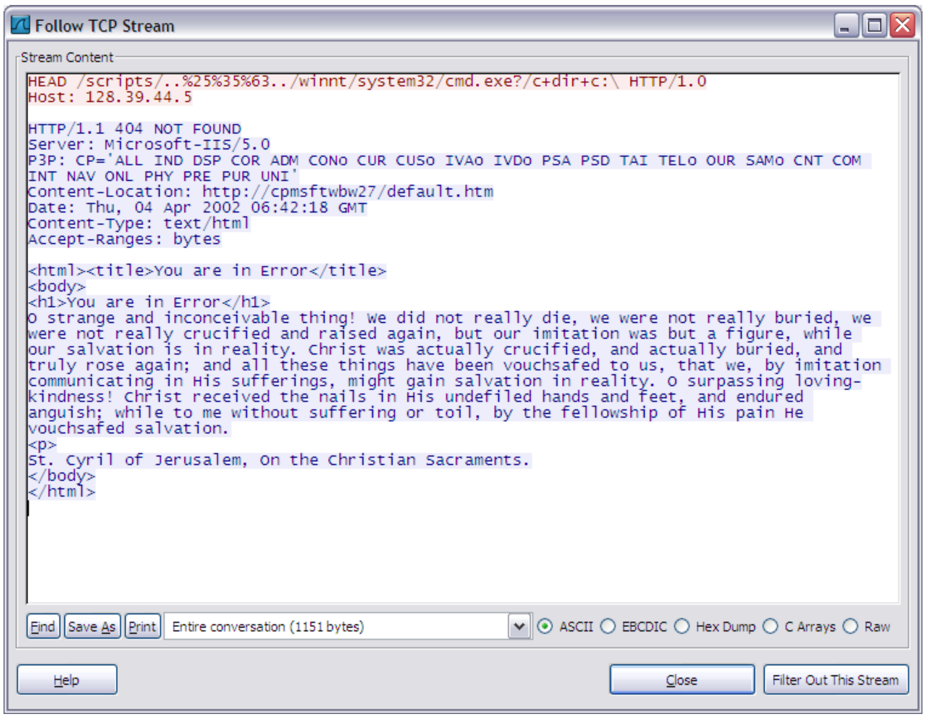
Directory traversal attempt

The goal of this feature is to try to identify attacks that can lead to information disclosure by traversing directories in an attempt to access files or information. The data set contains 1 879 connections with a positive value, and not all of these connections are labeled as an attack. Only 30 of these connections are computer attacks, and the attack types are: "HTTP directory traversal", "IIS SAM attempt" and "Web /etc/passwd attempt." The rest of these connections are, as far as we know, normal traffic. After deeper analysis of the traffic labeled as normal by *Snort*, we have found that it is in fact some old attacks on *IIS*. The attack target is *FrontPage Extension*, using old attack vectors that in past versions of *IIS* offered a method for executing commands using the native shell in *Windows*. An example payload of the whole connection is presented in Figure 3.

After manual inspection of the payload for the connections with a positive value of the "Directory traversal attempt", we have concluded that these connections are all old attack vectors. There are some variations in the vector, but all have the string "../winnt/system32/cmd.exe ?/c+dir+c:\", which is an attempt to get access to the native shell on *Windows* systems. According to a post on SecurityFocus forum [54], these attack vectors are about 3-4 years old and the vulnerabilities have been removed several years ago.

We have chosen not to modify the classifications for the connections in questions, since these attack vectors are old and the system has been patched. This is probably the reason why *Snort* does not generate a report on these occurrences. The result of the feature is that all the connections with a positive value are attacks, even though not all are labeled as attacks in the data set.

Some occurrences of the "Web /etc/passwd attempt" attack trigger both the "Access to



```
HEAD /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.0
Host: 128.39.44.5

HTTP/1.1 404 NOT FOUND
Server: Microsoft-IIS/5.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo OUR SAMo CNT COM
INT NAV ONL PHY PRE PUR UNI'
Content-Location: http://cpmsftwbw27/default.htm
Date: Thu, 04 Apr 2002 06:42:18 GMT
Content-Type: text/html
Accept-Ranges: bytes

<html><title>You are in Error</title>
<body>
<h1>You are in Error</h1>
O strange and inconceivable thing! We did not really die, we were not really buried, we
were not really crucified and raised again, but our imitation was but a figure, while
our salvation is in reality. Christ was actually crucified, and actually buried, and
truly rose again; and all these things have been vouchsafed to us, that we, by imitation
communicating in His sufferings, might gain salvation in reality. O surpassing loving-
kindness! Christ received the nails in His undefiled hands and feet, and endured
anguish; while to me without suffering or toil, by the fellowship of His pain He
vouchsafed salvation.
<p>
St. Cyril of Jerusalem, On the Christian Sacraments.
</body>
</html>
```

Figure 3: A sample payload for a normal connection with an positive value for the "Directory traversal attempt" feature.

passwd attempted" and "Directory traversal attempted" features. Four of the total seven connections with the attack have positive values for both features. In these connections, the remote host using directory traversal techniques in an attempt to access the *passwd* file on the target host. No other attacks trigger multiple content features.

4.3.10 Results

Snort managed to detect all attacks in the original data set, as we expected, by failed to identify two attacks in the modified data set. The detection rate, true positive rate, is for the original data set 100 %. For the modified data set the number of connections marked as some sort of attack is: 23 144. The SYN scan contains 22 295 of these attacks, and is the attack type with the most occurrences (96,3 %). *Snort* managed to detect 841 connections with attacks, and giving it a detection rate of 3,6 %. The main reason is that *Snort* failed to identify the SYN scan; if we exclude it from the computation we get a detection rate of 99,1 %. The only attack that is not detected is eight connections that contain the Traceroute UDP scan. The base rate, rate of attacks, for the original data set is 4,0 %, while the base rate for the modified data set is 54 %. Excluding the SYN scan from the data set, give a base rate of 4,1 %.

Based on the general statistics of the modified data set, we analyze the features and the attacks in order to try to determine some characteristics of the attacks. Identifying features that are relevant for identifying each of the attacks is the main goal. The basis for these analyses is the modified data set. Tables for all attacks found in the data set is provided in the appendix, and for a few attacks we only provide an excerpt as the attack consist of too many connections to include all in the report.

4.3.11 Relevant features for detecting a given attack type

Authorization basic overflow attempt

In the data set, four connections contained this attack type, and after analyzing these connections we have found they have some features in common. The service field is 80, meaning HTTP traffic with the three traffic features: "Count", "Same_srv_rate" and "Srv_count". These three features have all the same value, one, and with the service field they uniquely identify only the connections with this attack type. No other connections have the same combination of feature values, giving this combination a detection rate of 100 %.

Chunked-Encoding transfer attempt

Only one connection contains this attack type. As a consequence, we cannot conclude which features are best suited for detecting this attack. We need more occurrences of this attack to do a more thorough analysis.

Cross site scripting attempt

In the data set we find seven occurrences of this attack. Six of these have a positive value, true, for the "Cross site scripting attempted" feature. After analyzing the connection with a negative value for this feature, we found that the character '<' had been replaced with its equivalent hex number '%3C'. This is a technique to avoid detection by IDS and we had not programmed the prototype to deal with this event in checking the payload. By fixing this in the prototype all the seven connections with cross site scripting attack, will have a positive value of the feature. No other connections have a positive value of the feature, and it is very useful in detection of this attack with a detection rate of 100 %.

HTTP directory traversal

The data set contains 14 connections with this attack type, but the directory traversal technique can be used as a part of a more sophisticated attack. The features with the same values these connections all have in common are "Directory traversal attempt" and "Same_srv_rate". All connections with this attack trigger the new content feature, and they also have a "Same_srv_rate" of one, which is the highest obtainable value. This combination is not unique for this specific attack and 1 855 connections are identified in the data set. The other connections contain one of the following attack types: "IIS SAM attempt" and "Web /etc/passwd attempt", with the remaining connections labeled as "normal" traffic. But as mentioned in Section 4.3.9, these connections are in fact old attack vectors on *FrontPage Extensions* in older version of *IIS*. Since these attacks use directory traversal techniques, the combination of features can be used to detect this specific attack, and also other attacks that use this technique as part of their attack strategy. The detection rate of the "HTTP directory traversal" connections is 0,75 %, and the rest of the connections identified contain another attack type that uses the directory traversal technique as part of their attack.

ICMP Ping NMap

This probing attack occurs in eight connections in the data set. Determining the values for the features that uniquely identify these connections is a more complicated task, comparing to the previous attacks. With the value of the "Protocol" feature equal to *ICMP*, the "Srv_diff_host_rate" value greater or equal to 0,94 and the "Src_bytes" and "Dst_bytes" equal to 60, we managed to identify seven out of the eight "ICMP Ping NMap" attacks.

The last attack contained the value 440 for the “Src_bytes” and “Dst_bytes” features. The reason for this discrepancy compared to the other attacks is that since the *ICMP* protocol does not have a timeout value, multiple *Nmap* scans can be stored in the same connection. A remote host has most likely scanned a host multiple times and as a result the connection contains all these scans, as long as the *IP* and port numbers are the same. The combination of features has also identified 11 connections labeled as “normal”, resulting in a detection rate of 38,9% for this data set. The false positive rate is 61,1% and the false negative rate is 12,5 %.

ICMP Superscan echo

This is also a probing attack and it occurs six times in the data set. The attack consists of sending a payload of eight bytes with zeros, but after analyzing the attack we see that five of the six connections have twice the total size as it should have been. The simple reason is that for the five connections, two requests have been sent instead of only one. To successfully identify these attacks, we have based the combination of features with the five connections in mind. The combination of features is the same as the previous attack: “Protocol”, “Src_bytes” , “Dst_bytes”, and we have added “Count”. The protocol is “ICMP” but the “Src_bytes” and “Dst_bytes” equals 120. For the “Count” feature, we have used the range [50 , 60>. This combination identifies five out of the six occurrences of this attack. The last occurrence is not identified because the connection only consists of one request. The detection rate of this combination is 100 % and the false negative rate is 16,67 %.

IIS SAM attempt

We find 12 connections with this attack in the data set. After analyzing these connections we cannot find any combination of features that only identifies these attacks. The combination of “Directory traversal” equal to true, “Same_srv_rate” equal to one and the “Duration” value between the range [0,157 , 0,215], identifies 1574 connections in the data set. Most of these connections is labeled as “normal” and only 12 are the attack. The attack has close resemblance to normal traffic, and we conclude that this attack is hard to filter out based on the features we have calculated in this thesis. Adding more content features to the data set, could possibly filter out more of the normal traffic and only identify the attack. Since the attack is based on sending a special payload, most of the basic and traffic features offer little help in uniquely identifying it. The detection rate of the proposed combination is 0,76 %, and the false positive rate is 99,24 %.

IIS view source via translate header

The data set contains 15 connections with this attack type. This attack is as the previous one content based, since the attacker crafts a special payload to send. We had also the same problem as with the previous attack to uniquely identify only the connections containing the attack in the data set. The attack resembles normal traffic when analyze the features in the data set, and only with adding more content features to the data set can we possible differentiate between the attack and normal traffic. The best combination we obtained, included the “Protocol” feature equal to *TCP*, the “Srv_count” and “Same_srv_rate” features equal one, all content features equal false and the “Count” value between the range [122 , 124]. This combination identifies 35 connections, which four contains the attack. The resulting detection rate is 11,43 % and the false positive rate is 88,57 % for the combination. The false negative rate is in this case 73,33 %.

MS SQL worm propagation attempt

This attack is present in 765 connections and the attack is simple to identify, when using the “Service” feature and the value 1 434. This identifies all occurrences of this attack, and additional five connections containing the “SYN Scan” attack are identified. By including the “Protocol” feature with the value UDP, we only identify the connections containing this attack. No normal connection is identified by this value for the feature, giving it a detection rate of 100 %.

SYN Scan

This is the attack type that has the most occurrences in the data set, and to successfully identify these we use a combination of basic and traffic features. The purpose is to identify as many as possible without obtaining “normal” connections. By using the “Duration” feature in the range [0 , 0,016] with equal “Src_bytes” and “Dst_bytes” value in combination with the traffic feature “Diff_srv_rate” in the range [0,5 , 1], we identify 22 292 out of the total 22 295. No other connections are identified and we only missed three connections. The detection rate for this combination is 99, 99 % and the false negative rate is 0,013 ‰.

Traceroute ICMP

This connection is not the essential part of the Traceroute, but it is issued in order to ping host(s). Traceroute uses *TCP* or *UDP* packets to determine the layout of a network, and they constitute the attack. But since the *ICMP* packets were also issued when the scan was executed, they are included as a connection in the data set. All the *ICMP* packets being stored in one connection and we cannot use the features to try to find a combination to identify this connection, we require at least two connections for this task.

Traceroute UDP

The connections issued by launching this scan, by using *Nessus*, can be divided into two groups. One group is connections that contain the destination service netbios-ns (137). These connections are probably not a direct part of the Traceroute, merely packets for determining NetBIOS name of the host. To identify these connections, we can use a combination of the “Protocol” and “Land” features. With the “Protocol” equal to *UDP* and the “Land” value equal to true, we identify the five connections with the destination service equal 137.

The second group contains connections with the “Service” feature in a much higher port range. Normally Traceroute uses the port range 33 434 to 33 534, but in this data set the “Service” field contains the value 32 768. The port range Traceroute uses can be manually configured, as we assume is the case for these connections. In order to identify these, we use the combination of the “Protocol” feature and the “Service” feature. The “Protocol” value is *UDP* and the “Service” value is 32 768, in this special case. Normally we would have used the port range mentioned above. This combination identifies all three connections and no other, giving it a detection rate of 100 %.

Web /etc/passwd attempt

In the data set there are seven connections, which contain this attack type. The attack is, as some other mentioned above, a content attack. The attacker crafts a special payload in an attempt to disclose information. To uniquely identify these connections, we use the content feature “Access to passwd” with the value true. This identifies all the seven connections and no normal connections, giving the feature a detection rate of 100 %.

WebDAV search access

The last attack type is the “WebDAV search access”, and only occurs in one connection in the data set. This makes it impossible to determine a combination of features to uniquely identify this attack from the normal traffic.

4.3.12 Discussion regarding relevance of the features

After analyzing how to identify different attack types by using a combination of the features found in the data set, we determine the relevance of the chosen features. We also discuss the possibility of adding other features, that can be better suited for identification of network attacks. We plot the attacks and features in the same table to determine their relevance in detecting different attacks, see Table 5. All considerations are based on the modified data set.

	Attack													
Feature	Authorization basic overflow attempt	Chunked-Encoding transfer attempt	Cross site scripting attempt	HTTP directory traversal	ICMP Ping NMap	ICMP Superscan echo	IIS SAM attempt	IIS view source via translate header	MS SQL Worm propagation attempt	SYN Scan	Traceroute ICMP	Traceroute UDP	Web /etc/passwd attempt	WebDAV search access
Duration										x				
Protocol					x	x			x			x		
Service	x								x			x		
Src_bytes					x	x				x				
Dst_bytes					x	x				x				
Land												x		
Urgent														
Access to passwd													x	
Cross site scritping			x											
Directory traversal				x										
Count	x													
Same_srv_rate	x			x										
Diff_srv_rate										x				
Srv_count	x													
Srv_diff_host_rate					x	x								

Table 5: The relevance of features in identifying different attack types

The only feature that cannot be used to identify any of the attacks found in the data set, is the “Urgent” feature. None of the packets in any of the connections in both data sets have the Urgent flag enabled. As a consequence, it can be removed from the data sets without causing any alterations to the characteristics of the data sets.

We did not find any combinations of features that uniquely identified the attacks: “IIS SAM attempt” and “IIS view source via translate header.” Many normal connections would also be identified by a combination of the features found in the data set. In order to reduce the number of normal connections identified by the current feature combination, we need more content features. As mentioned in a previous section, these attacks are content based and closely resemble normal traffic, web traffic in these cases. Both basic and traffic features are not designed with this type of attack in mind, and offer little aid in detecting these attacks. Adding other content features can reduce the number of normal connections being identified and increase the detection rate of these attacks.

In the group of basic features, the “Protocol” feature is used, in combination with others, to identify four attacks. It is the most frequent feature used to identify attacks, and three out of four are probing attacks. “Service”, “Src_byte” and “Dst_byte” are all used to identify three attacks. The “Service” feature is used to identify two of the total three U2R attacks, while both “Src_byte” and “Dst_byte” only identifies probing attacks.

Most of the traffic features are only used once to identify attacks, a part from “Same_srv_rate” and “Srv_diff_host_rate.” “Srv_diff_host_rate” is used to identify two probing attacks, while “Srv_diff_host_rate” is used in identification of one information disclosure and one U2R/DOS attack.

The Table 6 contains the detection rate, as well as both false positive rate and false negative rate for the best combination of features to uniquely identify a specific attack. Three of the attacks were only present in one connection, making it impossible to de-

Attack	**DR, TPR**	**FPR**	**FNR**
Authorization basic overflow attempt	100 %	-	-
Chunked-Encoding transfer attempt	NA	NA	NA
Cross site scripting attempt	100 %	-	-
HTTP directory traversal	0,75 %	*	*
ICMP Ping NMap	38,9 %	61,1 %	12,5 %
ICMP Superscan echo	100 %	-	16,67 %
IIS SAM attempt	0,76 %	99,24 %	-
IIS view source via translate header	11,43 %	88,57 %	73,33 %
MS SQL worm propagation attempt	100 %	-	-
SYN Scan	99,99 %	0,013 ‰	-
Traceroute ICMP	NA	NA	NA
Traceroute UDP	100 %	-	-
Web /etc/passwd	100 %	-	-
WebDAV search access	NA	NA	NA

Table 6: Detection rate, false positive rate and false negative rate for the attacks found in the modified data set

termine appropriate features to uniquely identify the specific attack from all the other connections in the data set. Because of this fact, we neither could compute detection rate, false positive rate and if possible false negative rate. These three attacks have been marked with the value ‘NA’, not available, for all the rates. The detection rate for the “HTTP directory traversal” attack is very low, but we have not computed false positive rate or false negative rate. The best combination of features we obtained also identified other attacks than the specific “HTTP directory traversal” attack. The reason is that the combination detects all occurrences of the directory traversal technique, which can be a part of an larger attack. These other attacks are not normal traffic, and cannot be used in computing the false positive rate. The definition of the false positive rate, is the chance of generating an alarm caused by normal traffic and not an attack. Therefore it would be wrong to compute false positive rate for combination of features that are used identify the “HTTP directory traversal” attack, so we use ‘*’ to instead. For all other attacks, the character ‘-’ means that the value is zero or does not exist for the attack in question. An attack with a detection rate of 100 %, has not a false positive rate.

Three of the attacks in the data set are represented only once, making it impossible to determine appropriate features for identification. More occurrences of these attacks are needed in order to determine, if a combination of features can uniquely identify the attack in question. There is some uncertainty with the combination of features for identifying some of the cases of attacks, those with a low number of occurrences in the data set. More attacks of the same type would reduce this, as the probability of discovering other features that can be more relevant in detecting a specific attack would be reduced. Based on the features and attacks found in the modified data set, only one feature does not contribute any information in relevance to detecting the attacks, and this feature can therefore be removed without any consequence. As mentioned above, adding other content features can aid in the task of identifying some of the attacks.

5 Conclusions

We have constructed a framework for processing network packets, stored in the *XML* format, to construct connection records. These connection records can be labeled and different statistics computed for each one. The statistics are used to construct features, which present various characteristics of these connections and the relationship between them in the provided data. We have deployed honeypots, using *honeyd*, for capturing traffic, since it is difficult to capture traffic on the institution's network due to legislation and laws. This solution has not provided a wide range of network attacks, and we have been forced to generate attacks to include in the data set.

One of the main reasons for generating a new benchmarking data set is to include more updated attacks. We managed to add some new attack techniques, which are not present in the KDD CUP data set from 1999. The selection of new network attacks is rather limited mostly due to the fact that we have used a low interaction honypots to capture the network traffic. More sophisticated attacks usually require more interaction with the target host, making it harder if not impossible to launch the attack against the honeypots. All of the attacks added to the data set, was created with running *Nessus*. *Nessus* is not the best tool to generate malicious traffic, since its purpose is a vulnerability scanner and not an attack generator. There exist other applications that are possibly more suited for this task, like *the Metasploit framework*, but their use is very complicated and time consuming.

The prototype has some limitations regarding processing of network packets. Not all network protocols are processed, only the protocols that are frequently used in network communications and on the Internet. Other protocols are not that important when it comes to IDS, as the bulk of the traffic that is relevant for such systems are the *TCP* and *UDP* transport protocol.

The constructed prototype generates a data set, based on processing of network packets and establishing connection records. As mentioned in Section 2.2, the greatest challenge in this thesis has been the preparation of the data mining process. This process has consumed most of our time and construction of a framework for this task is essential in order to be able to utilize KDD techniques. The algorithm we chose for the data mining task was a form of classification. All network packets were gathered into connection records, which were classified as either normal traffic or a specific network attack. The interpretation step included, computing statistics and extracting features based on the connection records. And the last step in the KDD process is to use the discovered knowledge, in this case creation of an IDS benchmarking data set.

We return to the research questions, to summarize the result for this thesis:

1. How can common properties of attacks be extracted from a large and evergrowing set of attacks against computers networks?

   We have constructed a framework for processing network packets captured by *tcpdump*, to extract common properties for all connections. Many of these properties are closely related to computer network attacks and they can be used to successfully identify these attacks.

2. What are the properties of an IDS benchmarking data set based on real traffic for whose generation the methods of extraction of common properties of attacks have been used?

   To answer this question we have performed some experiments on the modified data set and analyzed the results as well as an analysis of the data set in general. We have determined the detection rate for *Snort*, by running it on both data sets, and calculated statistics for each feature and attack type. The relevance of each feature has also been determined, in regards to detection of network attacks. The detection rate for each attack by the best combination of features is also determined, as well as false positive rate and false negative rate if possible.

6 Further work

We have developed a framework for processing captured network packets and using these to construct connection records. There are several aspects that can be improved as further work:

- Add more attack types, gather more sophisticated attacks that are present in the wild. This can be done by utilizing a honeypot with a higher level of interaction than *honeyd*. Other attack tools can also be used to generate other attacks that are representative for today's threats.
- Add more web traffic to the data set, by hosting multiple sites. A possible solution can be to use a higher level interaction honeypot.
- Determine frequent episodes by finding sequences of events for feature generation and extraction. This process can yield other appropriate features, given the sequential nature of attacks.
- Add more content features to the data set that can better identify newer types of network attacks.
- Test the benchmarking data set against other intrusion detection systems.

Bibliography

[1] Stolfo, S. J., Fan, W., Lee, W., Prodromidis, A., & Chan, P. K. KDD CUP data set from the knowledge discovery and data mining tools competition. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [Last visited March 2007].

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

[3] tcpdump. http://www.tcpdump.org [Last visited May 2007].

[4] Mell, P., Hu, V., Lipmann, R., Haines, J., & Zissman, M. An overview of issues in testing intrusion detection systems.

[5] Rieck, K. & Laskov, P. DIMVA 2006. PESIM 2005 data set from - detecting unknown network attacks using language models.

[6] Bace, R. G. 2000. *Intrusion detection*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA.

[7] Schäfer, G. 2003. *Security in Fixed and Wireless Networks*. Wiley.

[8] Bishop, M. 2003. *Computer Security - Art and Science*. Addison Wesley.

[9] Haines, J., Lippmann, R., Fried, D., Zissman, M., Tran, E., & Boswell, S. 1999 darpa intrusion detection evaluation - design and procedures. Technical report, Lincoln Laboratory - MIT, 2001.

[10] Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunningham, R., & Zissman, M. 2000. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA. IEEE Computer Society Press.

[11] Salvatore J. Stolfo, Wei Fan, W. L. Cost-based modeling for fraud and intrusion detection results from the jam project.

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

[13] Lee, W., Stolfo, S. J., & Mok, K. W. 1999. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, 120–132.

[14] Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. 1996. From data mining to knowledge discovery: An overview.

[15] Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. 1996. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11), 27–34.

[16] Mannila, H. 1996. Data mining: Machine learning, statistics, and databases. In *Statistical and Scientific Database Management*, 2–9.

[17] Lee, W., Stolfo, S., & Mok, K. 1999. Mining in a data-flow environment: Experience in network intrusion detection. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, Chaudhuri, S. & Madigan, D., eds, 114–124.

[18] Insecure.org. Nmap. http://www.insecure.org/nmap [Last visited May 2007].

[19] Spitzner, L. 2002. *Honeypots: Tracking Hackers*. Addison Wesley Professional.

[20] Provos, N. Honeyd. http://www.honeyd.org [Last visited January 2007].

[21] Provos, N. 2004. A virtual honeypot framework.

[22] Sadasivam, K., Samudrala, B., & Yang, T. A. 2005. Design of network security projects using honeypots. *J. Comput. Small Coll.*, 20(4), 282–293.

[23] UNINETT - The Norwegian Research Network. http://www.uninett.no [Last visited November 2006].

[24] OpenBSD. http://www.openbsd.org [Last visited January 2007].

[25] Artymiak, J. 2003. *Building Firewalls with OpenBSD and PF*.

[26] The Metasploit Project. Metasploit framework. http://www.metasploit.com [Last visited February 2007].

[27] Nessus - the network vulnerability scanner. http://www.nessus.org [Last visited May 2007].

[28] Combs, G. Wireshark - network protocol analyzer. http://www.wireshark.org [Last visited March 2007].

[29] Combs, G. Ethereal - a network protocol analyzer. http://www.ethereal.com [Last visited November 2006].

[30] Snort - open source network intrusion prevention and detection system. http://www.snort.org [Last visited May 2007].

[31] Northcutt, S. & Novak, J. 2002. *Network Intrusion Detection - Third Edition*. New Riders.

[32] Kayacik, H. G., Zincir-Heywood, A. N., & Heywood, M. I. 2005. Selecting features for intrusion detection: A feature relevance analysis on kdd 99. In *PST*.

[33] Symantec. Internet Security Threat Report, Trends for July - December 06. Technical report, Symantec, 2007.

[34] SANS Institute. 2006. Sans top-20 internet security attack targets (2006 annual update). https://www.sans.org/top20/ [Last visited May 2007].

[35] OWASP. 2007. The ten most critical web application security vulnerabilities 2007. http://www.owasp.org/index.php/Top_10_2007 [Last visited May 2007].

[36] Mannila, H., Toivonen, H., & Verkamo, A. I. 1995. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Fayyad, U. M. & Uthurusamy, R., eds, Montreal, Canada. AAAI Press.

[37] Mannila, H., Toivonen, H., & Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3), 259–289.

[38] Mannila, H. & Toivonen, H. 1996. Discovering generalized episodes using minimal occurrences. In *Knowledge Discovery and Data Mining*, 146–151.

[39] Postel, J. September 1981. RFC 791 - Internet Protocol Specification version 4.

[40] Postel, J. August 1980. RFC 768 - User Datagram Protocol.

[41] Postel, J. September 1981. RFC 793 - Transmission Control Protocol.

[42] Postel, J. September 1981. RFC 792 - Internet Control Message Protocol.

[43] Cve-2003-0727 - multiple buffer overflows in Oracle database. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0727 [Last visited March 2007].

[44] CVE-2002-0079 - IIS remote buffer overflow. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0079 [Last visited March 2007].

[45] Carnegie Mellon University. CERT Advisory CA-2000-02 Malicious HTML tags embedded in client web requests. http://www.cert.org/advisories/CA-2000-02.html [Last visited March 2007].

[46] Hoglund, G. & McGraw, G. 2004. *Exploiting Software - How to break code*. Addison-Wesley.

[47] Foundstone. Superscan - a free tcp port scanner, pinger, resolver. http://www.foundstone.com [Last visited March 2007].

[48] CIAC. Windows NT SAM permission vulnerability. http://www.ciac.org/ciac/bulletins/h-45.shtml [Last visited March 2007].

[49] CVE-2000-0778 - Microsoft IIS 5.0 "Translate: f" source disclosure vulnerability. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0778 [Last visited March 2007].

[50] Carnegie Mellon University. CERT Advisory CA-2003-04 MS-SQL Server worm. http://www.cert.org/advisories/CA-2003-04.html [Last visited March 2007].

[51] CVE-2002-0649 - Microsoft SQL Server 2000 Resolution Service vulnerabilities. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0649 [Last visited March 2007].

[52] McClure, S., Scambray, J., & Kurtz, G. 2005. *Hacking Exposed - Network Security Secrets & Solutions*. McGraw-Hill.

[53] CVE-2000-0951 - Microsoft IIS 5.0 Indexed Directory disclosure vulnerability. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0951 [Last visited March 2007].

[54] SecurityFocus. http://www.securityfocus.com [Last visited March 2007].

A Tables for each attack type

A.1 Authorization basic overflow attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
414	0,095	TCP	80	1632	3068	False	0	False	False	False
441	0,094	TCP	80	1632	3068	False	0	False	False	False
468	0,35	TCP	80	4508	3308	False	0	False	False	False
501	0,176	TCP	80	4508	3308	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt
1	1	0	1	0	Authorization basic overflow attempt

Table 7: Authorization basic overflow attempt

A.2 Chunked-Encoding transfer attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site script.**	**Directory traversal**
168406	4,988	TCP	80	5760	7094	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
63	1	0	2	0	Chunked-Encoding transfer attempt

Table 8: Chunked-Encoding transfer attempt

A.3 Cross site scripting attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
53	1,12	TCP	80	758	1474	False	0	False	True	False
58	1,374	TCP	80	714	1474	False	0	False	True	False
63	1,008	TCP	80	810	1474	False	0	False	True	False
68	1,23	TCP	80	700	1474	False	0	False	True	False
73	1,372	TCP	80	717	1474	False	0	False	True	False
94	1,366	TCP	80	718	1474	False	0	False	True	False
99	1,072	TCP	80	714	1474	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
1	1	0	1	0	Cross site scripting attempt
2	1	0	2	0	Cross site scripting attempt
3	1	0	1	0,666666667	Cross site scripting attempt
4	1	0	2	0,5	Cross site scripting attempt
5	1	0	3	0,4	Cross site scripting attempt
6	1	0	3	0,5	Cross site scripting attempt
7	1	0	4	0,428571429	Cross site scripting attempt

Table 9: Cross site scripting attempt

A.4 HTTP directory traversal

	BASIC FEATURES							CONTENT FEATURES		
ID	Duration	Protocol	Service	Src_Bytes	Dst_Bytes	Land	Urgent	Access to passwd	Cross site script.	Directory traversal
171	1,372	TCP	80	1040	3068	False	0	False	False	True
198	1,302	TCP	80	1032	3068	False	0	False	False	True
234	1,314	TCP	80	1030	3068	False	0	False	False	True
279	1,225	TCP	80	1024	3068	False	0	False	False	True
289	1,088	TCP	80	1069	3068	False	0	False	False	True
43608	424,122	TCP	80	1766	4422	False	0	False	False	True
53458	0,162	TCP	80	586	1474	False	0	False	False	True
64416	0,178	TCP	80	1172	3068	False	0	False	False	True
66882	0,211	TCP	80	1172	2948	False	0	False	False	True
81816	65,346	TCP	80	2596	5896	False	0	False	False	True
92959	0,166	TCP	80	1172	2948	False	0	False	False	True
101146	0,162	TCP	80	1038	2948	False	0	False	False	True
108775	0,171	TCP	80	1170	2948	False	0	False	False	True
119615	0,16	TCP	80	1172	2948	False	0	False	False	True

TRAFFIC FEATURES					
Count	Same_srv_rate	Diff_srv_rate	Srv_count	Srv_diff_host_rate	Status of connection
1	1	0	1	0	HTTP directory traversal
2	1	0	2	0	HTTP directory traversal
3	1	0	2	0,333333333	HTTP directory traversal
1	1	0	1	0	HTTP directory traversal
2	1	0	2	0	HTTP directory traversal
160	1	0	40	0	HTTP directory traversal
210	1	0	37	0,588888889	HTTP directory traversal
207	1	0	39	0,551724138	HTTP directory traversal
228	1	0	39	0,638888889	HTTP directory traversal
209	1	0	42	0,528089888	HTTP directory traversal
182	1	0	20	0,677419355	HTTP directory traversal
154	1	0	16	0,529411765	HTTP directory traversal
144	1	0	8	0,666666667	HTTP directory traversal
132	1	0	7	0,416666667	HTTP directory traversal

Table 10: HTTP directory traversal

A.5 ICMP Ping NMap

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
150851	2628,717	ICMP	0	444	444	False	0	False	False	False
162135	0,001	ICMP	0	60	60	False	0	False	False	False
173428	0,002	ICMP	0	60	60	False	0	False	False	False
173430	0	ICMP	0	60	60	False	0	False	False	False
173432	0,001	ICMP	0	60	60	False	0	False	False	False
173434	0,001	ICMP	0	60	60	False	0	False	False	False
173436	0	ICMP	0	60	60	False	0	False	False	False
173438	0	ICMP	0	60	60	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
160	0,5	0,5	1	0,992647059	ICMP PING NMAP
171	1	0	1	0,993464052	ICMP PING NMAP
58	1	0	1	0,941176471	ICMP PING NMAP
58	1	0	1	0,941176471	ICMP PING NMAP
59	1	0	1	0,944444444	ICMP PING NMAP
60	1	0	1	0,947368421	ICMP PING NMAP
61	1	0	1	0,95	ICMP PING NMAP
62	1	0	1	0,952380952	ICMP PING NMAP

Table 11: ICMP Ping NMap

A.6 ICMP Superscan echo

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site script.**	**Directory traversal**
12556	0,001	ICMP	0	60	60	False	0	False	False	False
12610	0,001	ICMP	0	120	120	False	0	False	False	False
12614	0,001	ICMP	0	120	120	False	0	False	False	False
12618	0	ICMP	0	120	120	False	0	False	False	False
12622	0	ICMP	0	120	120	False	0	False	False	False
12626	0	ICMP	0	120	120	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
52	1	0	1	0,923076923	ICMP Superscan echo
52	1	0	1	0,923076923	ICMP Superscan echo
53	1	0	1	0,928571429	ICMP Superscan echo
54	1	0	1	0,933333333	ICMP Superscan echo
55	1	0	1	0,9375	ICMP Superscan echo
56	1	0	1	0,941176471	ICMP Superscan echo

Table 12: ICMP Superscan echo

A.7 IIS SAM Attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
41225	0,163	TCP	80	554	1474	False	0	False	False	True
41253	0,164	TCP	80	554	1474	False	0	False	False	True
41281	0,159	TCP	80	554	1474	False	0	False	False	True
41309	0,163	TCP	80	554	1474	False	0	False	False	True
46686	0,157	TCP	80	1108	2948	False	0	False	False	True
46741	0,157	TCP	80	1108	2948	False	0	False	False	True
46797	0,158	TCP	80	1108	3068	False	0	False	False	True
46855	0,206	TCP	80	1108	2948	False	0	False	False	True
47523	0,215	TCP	80	1108	2948	False	0	False	False	True
47627	0,158	TCP	80	1108	2948	False	0	False	False	True
47685	0,157	TCP	80	1108	2948	False	0	False	False	True
47741	0,158	TCP	80	976	2948	False	0	False	False	True

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
137	1	0	17	0	IIS SAM Attempt
137	1	0	17	0	IIS SAM Attempt
137	1	0	17	0	IIS SAM Attempt
138	1	0	18	0	IIS SAM Attempt
179	1	0	17	0,711864407	IIS SAM Attempt
180	1	0	18	0,7	IIS SAM Attempt
181	1	0	19	0,68852459	IIS SAM Attempt
182	1	0	20	0,677419355	IIS SAM Attempt
193	1	0	18	0,75	IIS SAM Attempt
194	1	0	18	0,753424658	IIS SAM Attempt
193	1	0	18	0,75	IIS SAM Attempt
192	1	0	19	0,732394366	IIS SAM Attempt

Table 13: IIS SAM Attempt

A.8 IIS view source via translate header

	BASIC FEATURES							CONTENT FEATURES		
ID	Duration	Protocol	Service	Src_Bytes	Dst_Bytes	Land	Urgent	Access to passwd	Cross site script.	Directory traversal
167996	63,147	TCP	80	599	1414	False	0	False	False	False
144	0,388	TCP	80	758	3068	False	0	False	False	False
217	0,368	TCP	80	758	3068	False	0	False	False	False
30438	69,66	TCP	80	533	1414	False	0	False	False	False
30478	67,598	TCP	80	1198	2828	False	0	False	False	False
30482	67,316	TCP	80	1066	2828	False	0	False	False	False
30528	0,257	TCP	80	682	268	False	0	False	False	False
30538	0,288	TCP	80	682	268	False	0	False	False	False
149324	5,693	TCP	80	733	2276	False	0	False	False	False
151013	185,242	TCP	80	599	1414	False	0	False	False	False
160874	156,867	TCP	80	599	1414	False	0	False	False	False
170864	0,664	TCP	80	539	1414	False	0	False	False	False
170879	0,68	TCP	80	539	1414	False	0	False	False	False
170893	0,19	TCP	80	341	134	False	0	False	False	False
170899	0,196	TCP	80	341	134	False	0	False	False	False

TRAFFIC FEATURES					
Count	Same_srv_rate	Diff_srv_rate	Srv_count	Srv_diff_host_rate	Status of connection
66	0,5	0,5	1	0	IIS view source via translate header
1	1	0	1	0	IIS view source via translate header
2	1	0	1	0,5	IIS view source via translate header
122	1	0	1	0	IIS view source via translate header
123	1	0	1	0,5	IIS view source via translate header
124	1	0	1	0,666666667	IIS view source via translate header
124	1	0	1	0,666666667	IIS view source via translate header
124	1	0	1	0,666666667	IIS view source via translate header
132	1	0	1	0	IIS view source via translate header
164	1	0	1	0	IIS view source via translate header
169	1	0	1	0	IIS view source via translate header
39	1	0	1	0,5	IIS view source via translate header
41	1	0	1	0,666666667	IIS view source via translate header
42	1	0	1	0,75	IIS view source via translate header
38	1	0	1	0	IIS view source via translate header

Table 14: IIS view source via translate header

A.9 MS-SQL Worm propagation attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
656	0	UDP	1434	836	140	False	0	False	False	False
678	1079007,107	UDP	1434	6688	1120	False	0	False	False	False
696	709226,038	UDP	1434	3344	560	False	0	False	False	False
732	0	UDP	1434	836	140	False	0	False	False	False
792	801431,163	UDP	1434	2508	420	False	0	False	False	False
976	0,001	UDP	1434	836	380	False	0	False	False	False
980	0	UDP	1434	836	380	False	0	False	False	False
1099	1702213,853	UDP	1434	3344	70	False	0	False	False	False
1112	0	UDP	1434	836	140	False	0	False	False	False
1164	427509,639	UDP	1434	3344	560	False	0	False	False	False
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
1	1	0	1	0	MS-SQL Worm propagation attempt
1	1	0	1	0	MS-SQL Worm propagation attempt
2	1	0	1	0,5	MS-SQL Worm propagation attempt
3	1	0	1	0,666666667	MS-SQL Worm propagation attempt
3	1	0	1	0,666666667	MS-SQL Worm propagation attempt
4	1	0	1	0,75	MS-SQL Worm propagation attempt
4	1	0	1	0,75	MS-SQL Worm propagation attempt
4	1	0	1	0,75	MS-SQL Worm propagation attempt
5	1	0	1	0,8	MS-SQL Worm propagation attempt
5	1	0	1	0,8	MS-SQL Worm propagation attempt
⋮	⋮	⋮	⋮	⋮	⋮

Table 15: MS-SQL Worm propagation attempt

A.10 SYN scan

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site script.**	**Directory traversal**
175438	0,002	TCP	1774	60	60	False	0	False	False	False
175394	0,001	TCP	1774	60	60	False	0	False	False	False
175404	0	TCP	1774	60	60	False	0	False	False	False
175406	0,001	TCP	1774	60	60	False	0	False	False	False
175407	0,001	TCP	1773	60	60	False	0	False	False	False
175431	0,003	TCP	20203	60	60	False	0	False	False	False
175426	0,002	TCP	1772	60	60	False	0	False	False	False
175427	0,002	TCP	1771	60	60	False	0	False	False	False
175429	0,002	TCP	20203	60	60	False	0	False	False	False
175398	0,001	TCP	20203	60	60	False	0	False	False	False
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
74	1	0	1	0,8	SYN scan
49	1	0	1	0	SYN scan
54	1	0	1	0,5	SYN scan
56	0,5	0,5	1	0,666666667	SYN scan
56	0,5	0,5	1	0,5	SYN scan
73	0,2	0,8	1	0,75	SYN scan
73	0,2	0,8	1	0,75	SYN scan
73	0,2	0,8	1	0,75	SYN scan
73	0,2	0,8	1	0,75	SYN scan
53	0,2	0,8	1	0	SYN scan
⋮	⋮	⋮	⋮	⋮	⋮

Table 16: SYN scan

A.11 Traceroute ICMP

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
220024	13,992	ICMP	0	70	240	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
50	1	0	1	0,954545455	Traceroute ICMP

Table 17: Traceroute ICMP

A.12 Traceroute UDP

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site script.**	**Directory traversal**
220019	0	UDP	32768	60	70	False	0	False	False	False
219994	3,025	UDP	137	276	70	True	0	False	False	False
219996	3,026	UDP	137	276	360	True	0	False	False	False
219997	3,025	UDP	137	276	140	True	0	False	False	False
219998	3,025	UDP	137	276	140	True	0	False	False	False
219999	3,025	UDP	137	276	140	True	0	False	False	False
220021	0	UDP	32768	60	70	False	0	False	False	False
220023	0,013	UDP	32768	120	70	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
54	0,5	0,5	1	0	Traceroute UDP
50	1	0	1	0,5	Traceroute UDP
50	1	0	1	0,5	Traceroute UDP
53	1	0	1	0,8	Traceroute UDP
53	1	0	1	0,8	Traceroute UDP
53	1	0	1	0,8	Traceroute UDP
55	1	0	1	0,5	Traceroute UDP
49	1	0	1	0	Traceroute UDP

Table 18: Traceroute UDP

A.13 Web /etc/passwd attempt

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site scripting**	**Directory traversal**
1	1,12	TCP	80	752	1474	False	0	True	False	False
14	1,169	TCP	80	750	1474	False	0	True	False	True
19	1,197	TCP	80	744	1474	False	0	True	False	True
24	1,21	TCP	80	713	1474	False	0	True	False	True
333	1,406	TCP	80	1208	3068	False	0	True	False	True
343	1,342	TCP	80	1083	3068	False	0	True	False	False
370	1,296	TCP	80	1031	3068	False	0	True	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
1	1	0	1	0	Web /etc/passwd attempt
1	1	0	1	0	Web /etc/passwd attempt
2	1	0	1	0,5	Web /etc/passwd attempt
3	1	0	2	0,333333333	Web /etc/passwd attempt
1	1	0	1	0	Web /etc/passwd attempt
2	1	0	1	0,5	Web /etc/passwd attempt
3	1	0	2	0,333333333	Web /etc/passwd attempt

Table 19: Web /etc/passwd attempt

A.14 WebDAV search access

	BASIC FEATURES							CONTENT FEATURES		
ID	**Duration**	**Protocol**	**Service**	**Src_Bytes**	**Dst_Bytes**	**Land**	**Urgent**	**Access to passwd**	**Cross site script.**	**Directory traversal**
168399	20,941	TCP	80	74204	4414	False	0	False	False	False

TRAFFIC FEATURES					
Count	**Same_srv_rate**	**Diff_srv_rate**	**Srv_count**	**Srv_diff_host_rate**	**Status of connection**
64	0,333333333	0,666666667	1	0	WebDAV search access

Table 20: WebDAV search access