A FASTER INTRUSION DETECTION METHOD
FOR
HIGH-SPEED COMPUTER NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET CEM TARIM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

MAY 2011

Approval of the thesis:

A FASTER INTRUSION DETECTION METHOD

FOR

HIGH-SPEED COMPUTER NETWORKS

Submitted by **MEHMET CEM TARIM** in partial fulfillment of the requirements for the degree of Master of Science in **Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, **Graduate School of Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Assist. Prof. Dr. Şenan Ece (Güran) Schmidt _____
Supervisor, **Electrical and Electronics Engineering Dept.**

Examining Committee Members

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Eng. Dept., METU

Assist. Prof. Dr. Şenan Ece (Güran) Schmidt _____
Electrical and Electronics Eng. Dept., METU

Prof. Dr. Uğur Halıcı _____
Electrical and Electronics Eng. Dept., METU

Assoc. Prof. Dr. Cüneyt Bazlamaçcı _____
Electrical and Electronics Eng. Dept., METU

Dr. Zeki Çiftçi _____
Manager, TBMM

Date: **May 09,2011**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name,Last name:Mehmet Cem TARIM
Signature :

ABSTRACT

A FASTER INTRUSION DETECTION METHOD FOR HIGH-SPEED COMPUTER NETWORKS

Tarım, Mehmet Cem

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Senan Ece (Güran) Schmidt

May 2011, 84 pages

The malicious intrusions to computer systems result in the loss of money, time and hidden information which require deployment of intrusion detection systems. Existing intrusion detection methods analyze packet payload to search for certain strings and to match them with a rule database which takes a long time in large size packets. Because of buffer limits, packets may be dropped or the system may stop working due to high CPU load. In this thesis, we investigate signature based intrusion detection with signatures that only depend on the packet header information without payload inspection. To this end, we analyze the well-known DARPA 1998 dataset to manually extract such signatures and construct a new rule set to detect the intrusions. We implement our rule set in a popular intrusion detection software tool, Snort. Furthermore we enhance our rule set with the existing rules of Snort which do not depend on payload inspection. We test our rule set on DARPA data set as well as a new data set that we collect using attack generator tools. Our results show around 30% decrease in detection time with a tolerable decrease in the detection rate. We believe that our method can be used as a complementary component to speed up intrusion detection systems.

Keywords: Intrusion Detection, Network Security, DARPA, SNORT

ÖZ

YÜKSEK HIZLI BİLGİSAYAR AĞLARI İÇİN DAHA HIZLI BİR SALDIRI TESPİT METODU

Tarım, Mehmet Cem
Yüksek Lisans, Elektrik ve Elektronik Mühendisligi Bölümü
Tez Danısmanı: Senan Ece (Güran) Schmidt

Mayıs 2011, 84 sayfa

Bilgisayar sistemlerine yapılan kötü niyetli saldırılar, saldırı tespit sistemlerinin kurulmasını gerektiren, para, zaman ve gizli bilgi kaybına neden olur. Mevcut saldırı tespit metodları belli dizileri bulmaya çalışmak ve bunları bir kural veritabanı ile eşleştirmek için paket yük kısmını inceler ve bu büyük boyutlu paketlerde çok uzun zaman alır. Geçici belleklerdeki limitlerden dolayı paketler düşürülebilir veya sistem yüksek CPU yükünden dolayı çalışmayı bırakabilir. Bu tezde, biz paket yük kısmını incelemeden sadece paket başlık bilgisine bakan imzalar ile oluşturulan imza tabanlı saldırı tespit sistemi geliştirdik. Bu amaçla, biz meşhur DARPA 1998 veri setini, böyle imzaları el ile çıkarmak, yeni bir kural seti oluşturmak ve saldırıları tespit etmek için inceledik. Biz kural setimizi popüler saldırı tespit yazılım aracı olan Snort'ta uyguladık. Buna ek olarak biz kural setimizi Snortun paket yük kısmını incelemeyen mevcut kuralları ile geliştirdik. Biz kural setimizi hem DARPA veri seti ile hem de bizim saldırı oluşturan araçları kullanarak topladığımız yeni bir veri seti ile test ettik. Bizim sonuçlarımız, tespit oranında kabul edilebilir bir azalışla tespit zamanının yaklaşık %30 azaldığını gösterdi. Biz inanıyoruz ki, bizim metodumuz saldırı tespit sistemlerininin hızını artırmak için tamamlayıcı bir unsur olarak kullanılabilir.

Anahtar Kelimeler: Saldırı Tespit, Ağ güvenliği, DARPA, SNORT

ACKNOWLEDGMENTS

First of all, I would like to thank my thesis advisor Senan Ece (Güran) Schmidt for her endless support.

I would like to thank my managers Dr. Zeki Çiftçi and Volkan Öztürk and my colleagues in TBMM for their valuable ideas.

I would also like to thank TUBITAK for its all kind of support.

Finally, I want to thank my dear wife Hatice Şule and sweet daughter İsra Nursu for their patience and giving me encouragement.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The meaning of the word intrusion is entrance by force or without permission or welcome. Another meaning is entry to another's property without right or permission. The network intrusion meaning is an unauthorized access to one or more components of a network. In computer networks, intrusions cause a serious security threat for the security of information.

In spite of the fact that the words “attack” and “intrusion” have different meanings; we usually use both words in the same meaning. Intrusions are successful attacks. They lower the confidentially of data by gaining unauthorized access and serving this data to anyone. In addition, they reduce the integrity of data by adding or deleting something in data. Finally, they reduce the availability of data by saturating the network and servers. Hence the intrusion detection systems are used to automatically scan the local area network devices and detect these network attacks.

There are lots of different classifications for intrusion detection methods. Among them, the misuse detection and anomaly detection are widely accepted and commercially available detection methods. The misuse detection systems break the network data into pieces and analyze to detect intrusions by matching the predefined rules with packet header or payload data. However they can only detect previously known intrusions. The rule set contains a signature database. The rule set has to be updated for each new type of discovered attacks. Since the signatures of the new attacks are not included in the rule set. Packet inspection takes a

long processing time and causes latency on the local area network. Moreover, this type of detection systems cause delays in the network and decrease the network traffic performance. Due to such delays the misuse based intrusion detection systems are not preferred especially in high speed computer networks. The anomaly based intrusion detection systems, first define the normal traffic condition and then compare their traffic with this normal traffic condition. If there is a different traffic from defined normal traffic, they alert to system administrator as anomaly traffic.

Detection of attacks and anomaly traffic in high speed networks require faster intrusion detection systems. Nowadays, the most of the intrusion detection systems analyze full IP packet payload and find matches with predefined set of rules to detect intrusions. This takes very long time. Also if the intrusion detection system is not fast enough, some packets pass through without inspection due to limitations of system buffers. These missed packets increase false negative rate. Therefore, it is necessary to develop faster intrusion detection methods. The hybrid based intrusion detection systems consist of two detection methods: misuse and anomaly.

In the literature there exist lots of different works to improve performance of the intrusion detection systems. Some works propose data mining techniques to select significant feature set to characterize the traffic. Some other works propose DFA or FSM to reduce memory requirement by rewriting the rules. However, for both anomaly and misuse detection approaches the characterization of the traffic (normal behavior and the attacks) is based on processing the entire IP packet with the IP and TCP headers as well as payload. This construction heavily relies on the payloads which slow down the detection process.

In this thesis, we investigate traffic characterization without using any information from IP packet payloads. We implement our approach for Snort which is a signature based IDS software tool. Most of the signatures in Snort default rules contain payload information. We construct new signatures for well-known intrusions which only depend on

the IP and TCP header information and write new Snort rules for this signature set. We extract the signatures by manually investigating the well-known DARPA data set. Furthermore we modify the default Snort rules by excluding any payload dependent information and use them to enhance our new Snort rules.

We test our new approach for its detection accuracy and speed on both DARPA data set and on a new data set that we created using attack generator tools.

Our rules enhanced with Snort header rules detection rates are 98.60% for DARPA 1998 data set and 100% for our data set. Also the false alarm rates are 2.16 for DARPA 1998 and 0.23% for our data set.

Besides we compare our enhanced rule set with snort default rule set. Snort default rule set detection rates are 98.48% for DARPA 1998 data set and 100% for our data set. Also the false alarm rates are 0.07 for DARPA 1998 data set and 0.11% for our data set.

Furthermore, with our enhanced rule set Snort detection speed is increased by 28.20%. Referenced papers detection rates and false positive rates are lower than our rules performance.

We believe that our approach can be used as a "fast path" for an existing intrusion detection system which employs payload inspection. It is possible to use our approach as a prescreening tool to eliminate the attacks that can be detected by header inspection instead of applying payload inspection to every packet.

The rest of the thesis is organized as follows. In chapter 2, intrusion detection systems and their types, taxonomy and general characteristic is shown. Besides, related works and their performances are discussed. Also, an open source well known program Snort and some important plugins Base and Barnyard are described. Furthermore, a packet analyzing tool Wireshark is discussed.

In chapter 3, our fast intrusion detection approach is shown. Also the attacks in DARPA 1998 data set are explained and their IP and transport

layer header to construct our rules is introduced. Then using these headers information, our rules are constructed and formulated in Snort. The experimental evaluations are discussed in Chapter 4 and conclusion is presented in Chapter 5.

CHAPTER 2

INTRUSION DETECTION SYSTEMS (IDS)

Intrusion detection has been a very important issue of computer network security field since the 1980s. It can be determined as any actions to damage the confidentiality, integrity or availability of a data. The belief that all attacks could be detected and blocked by firewalls and access lists is wrong. There must be an Intrusion Detection Systems (IDS) with firewall to detect intrusive activities and determine their nature, origin, and seriousness. Intrusions coming from both internal and external computer network can be detected by IDS.

This chapter provides an overview of the current status of research in intrusion detection. It first provides an overview of different types of computer intrusions, and then introduces a more detailed taxonomy of intrusion detection systems with an overview of important research in the field.

Also an open source well-known NIDS Snort is introduced in this chapter. Its components are introduced. Besides its useful plugins Base and Barnyard is introduced. Also to analyze tcpdump files, Wireshark program is introduced.

2.1 History of Network Security

Network security term entered our literature when the Morris worm damaged thousands of computers in November 1988. After this damage the U.S. Defense Advanced Research Programs Agency established the CERT/CC to manage these types of network security problems [1].

The Internet was first a closed network for academics and researches only and not publicly available. Hence the security weakness of the TCP/IP protocol suite was not corrected. Security of the Internet didn't take into account. But now, the Internet is open and publicly available and the weakness of the protocol causes network attacks.

Nowadays the Internet connects all computers around the world so network security becomes a serious problem. The normal Internet user must be careful against a number of network threats such as spam, virus, worms, trojan horses, bots, spyware, and phishing. Governments and companies must protect their servers and databases against the possibility of cyber fare.

With increasing usage of Internet, networked computer systems are now playing an important role in our society. In spite of the Internet makes simpler everything, it can also be dangerous. Specifically, new attacks are created every day to threaten every internet user and computer systems. As reported by the Computer Emergency Response Team/Coordination Center (CERT/CC) [2], the number of computer attacks has increased exponentially over the years (Figure 1). Furthermore, the severity and sophistication of the attacks is also growing (Figure 2). For example, Slammer/Sapphire Worm was the fastest computer worm in history. When it began spreading throughout the Internet, it doubled in size every 8.5 seconds and damaged nearly 75,000 hosts [2].

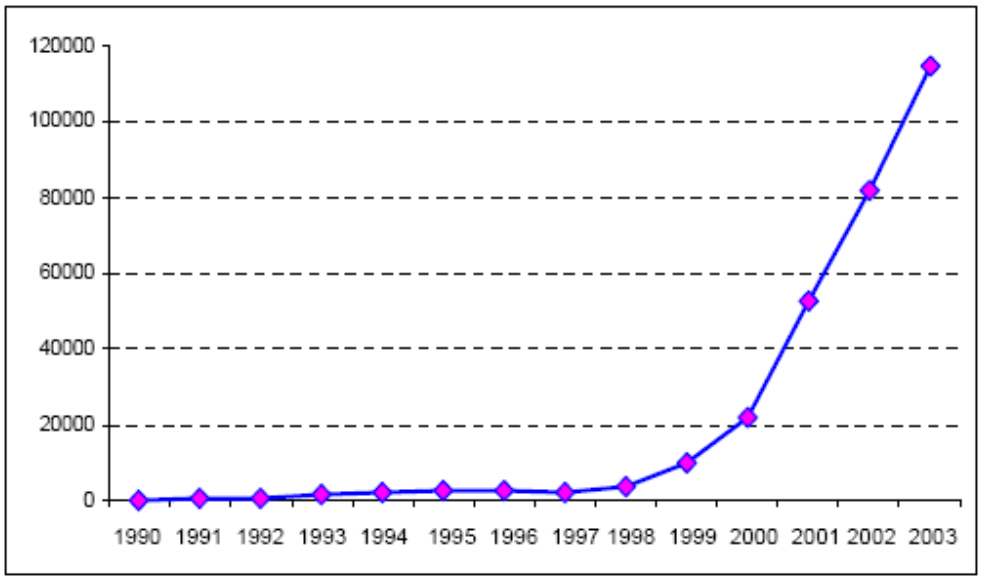

Figure 1 Growth rate of cyber incidents reported to Computer Emergency Response Team/Coordination Center (CERT/CC) [3].

In 1980s, the attackers needed whole understanding of computers and networks to launch some attacks. However, today almost anyone can generate very successful attacks with widely available attack tools [2].

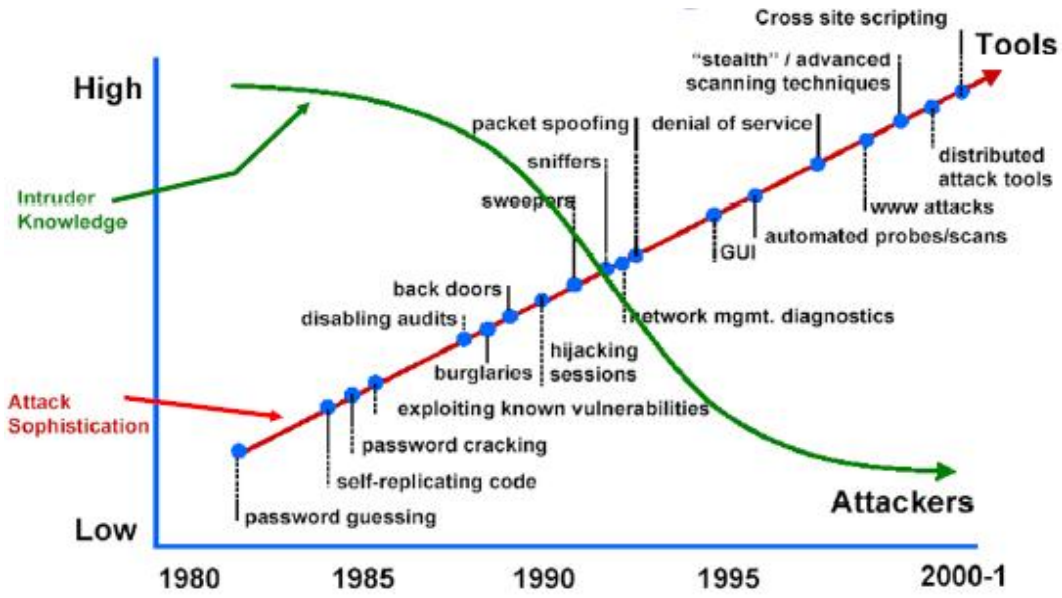

Figure 2 Attack sophistication vs. Intruder technical knowledge [3].

2.2 General Architectural Framework

Most of the Intrusion detection systems (IDSs) have general architectural framework shown in Figure 3. And this framework consists of the following components:

- **Data gathering device (sensors):** they are responsible for collecting data from the monitored system.
- **Detector (Intrusion detection analysis engine):** It analyzes the data collected from sensors and matches the data with database.
- **Knowledge base (database):** It contains signature of the network attacks.
- **Configuration device:** It gives information about the current state of the intrusion detection system (IDS).
- **Response component:** It generates an alarm when an intrusion is detected.

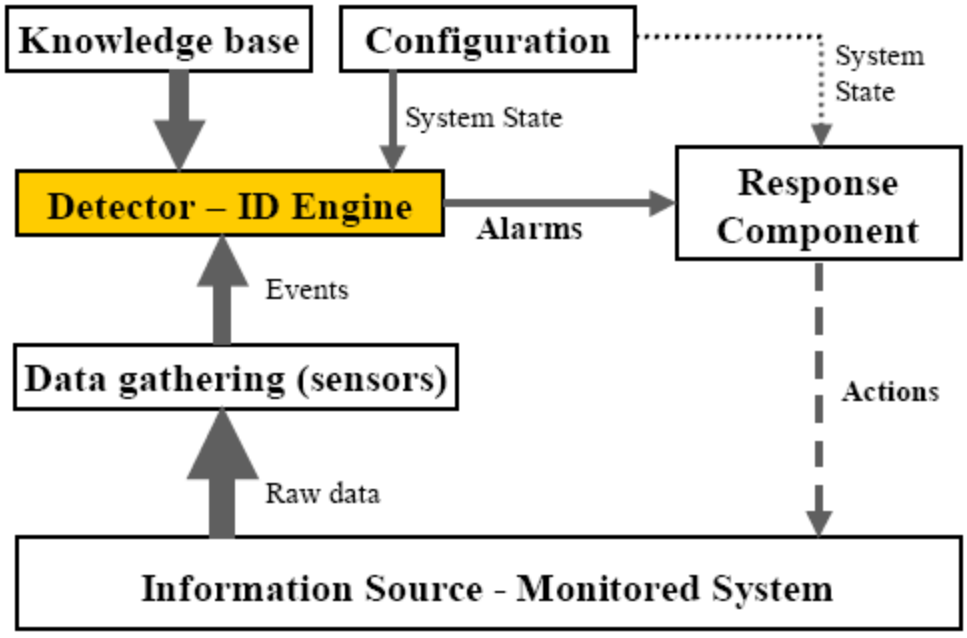

Figure 3 Basic architecture of IDS [2]

2.3 Characteristics of Intrusion Detection Systems

Intrusion detection systems have some characteristics as defined below [2]:

Detection performance: To measure the performance of an intrusion detection system, detection accuracy doesn't give correct information to us. For example, a sample data set contains some attacks which are 1% of the total data set. And a simple intrusion detection system (IDS) says that all network traffic is normal. The IDS accuracy becomes 99%. In order to measure real performance of IDS, the following definitions must be satisfied [2]:

- it must be able to correctly identify intrusions.
- it must not identify normal action as an intrusion.

Typical measures for evaluating detection performance of IDSs include detection rate and false alarm rate. Detection rate is defined as the ratio of the number of correctly detected attacks and the total number of attacks, while the false alarm (false positive) rate is the ratio of the number of normal traffic data that are misclassified as attacks and the total number of attacks. In practice, it is not easy to evaluate these two measures, because it is usually impossible to know the total number of attacks. Since detection rate and false alarm rate are often in contrast, evaluation of IDSs is also performed using ROC (Receiver Operating Characteristics) analysis. ROC curve represents a trade-off between detection rate and false alarm rate as shown in Figure 4. The closer the ROC is to the left upper corner of the graph (point that corresponds to 0% false alarm and 100% detection rate), the more effective the IDS are [2].

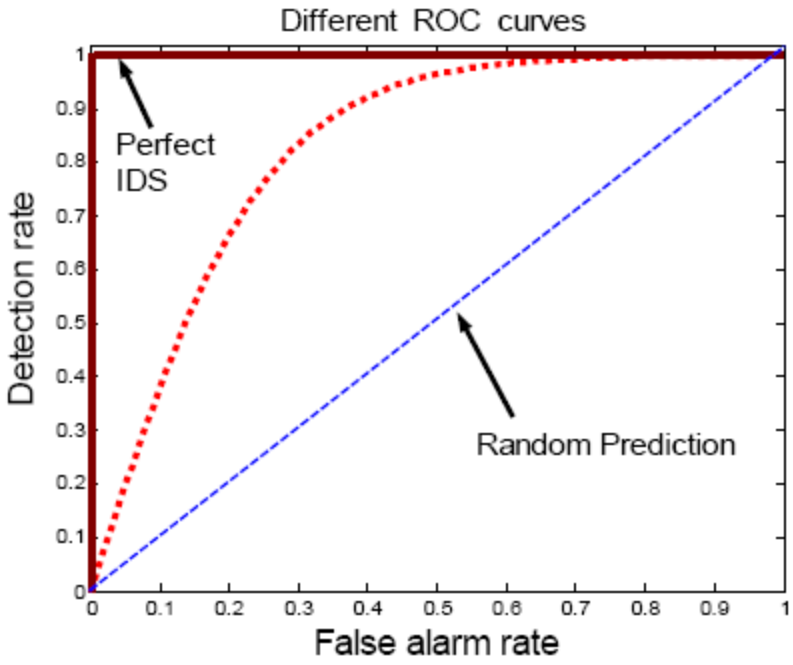

Figure 4 ROC Curves for different intrusion detection techniques [2]

Time performance: The time performance of an intrusion detection system can be measure with the total time that the IDS need to detect attacks. This time consists of the processing time and the propagation time. The processing time depends on the processing speed of the IDS, which is the rate at which the IDS process logs. If this rate is not high enough, then the real time processing of attacks may not be possible. The propagation time is the time needed for processed information to give alert to the system administrator. Both times need to be as short as possible in order to allow the system administrator sufficient time to stop an attack before much damage is being done, as well as to stop an attacker from modifying log information or altering the IDS configuration itself [2].

2.4 Taxonomy of IDS

Although there is no generally accepted taxonomy of IDSs, we can summarize that it has four categories as illustrated in Table 1. These are

misuse based, anomaly based, network based and host based. Also their advantages and disadvantages are summarized in the following.

Table 1 Summary of IDS types

Types of IDS	**Advantages**	**Disadvantages**
Host-Based	Independent from network topology	Limited view of entire network topology.
Network-Based	Monitor entire network	Increase network loads
Anomaly-Based	Novel attacks may be detected.	High false positive rate.
Misuse-Based	Low false positive rate. High detection rate.	Poor detection for novel attacks. Frequent update needed.

2.5 Related Works

In the rest of this section we present the relevant previous work on intrusion detection. All the works use the Defense Advanced Project Agency (DARPA) data sets and KDD Cup 1999 to test their systems. KDD Cup 1999 data set is derived from DARPA 1998 data set. In some works, they also construct their data set to support their DARPA results. DARPA intrusion detection data set is constructed on the simulated military network environment with different attacks. The victim machines operating systems are Linux, SunOSTM, and SolarisTM.

In [4] a hybrid procedure for developing rules by combining signature analysis with automated techniques (such as machine learning and statistical techniques) to achieve high detection rates with low false alarms is proposed. They develop rules for only two remote-to-local (R2L) attacks: warezmaster and warezclient. They test their rules on the KDD Data set built from DARPA 1998 data set. The proposed rules

achieve only 53.08% detection and 0.005% false alarm rates for the two R2L attacks in the KDD testing data set. They only develop two rules and their rules detection rate performance is much lower than ours. Besides, there is no other information about throughput of the system. We increase the system speed by 30%.

In [5] a Hidden Markov Model (HMM) to detect only eight mimicry attacks (eject, ffb, loadmodule, format, ftp-write, warezclient, satan, ipsweep) is proposed. They use system calls of host machine and the Hidden Markov Model Toolkit (HTK) to detect these attacks. They test their rules on the DARPA 1998 Data set. The results show average 93% detection and 3% false alarm rates for these attacks in the DARPA data set. They develop rules for limited intrusions. Also they use host intrusion detection and host machine system calls. This detection method must be installed all computer in local area network (LAN). Because attacks which are not destined to these installed computers, are not detected. The detection rate is a bit lower than ours. Also there is no information about system detection speed. With our rules the system speed is increased by 30% also the detection rate remain high enough nearly 95%.

In [6] [7], data mining based IDSs are proposed. In [6], a subset of significant feature set is selected. In [7], Support Vector Machines data mining techniques are used. The performance of the proposed systems is tested with DARPA 1998 data set. Detection rates for different attacks types are given. They are between 84%-100% for different attack types. They are similar with our rules result. However, there is no information about packet processing capacity. They don't show that their system works faster. But in our proposed system we show that the system speed is increased by 30%.

In [8], neural network based IDS is proposed. Besides, a feature selection algorithm is proposed. The proposed system looks for matches with packet header only. It is an anomaly based IDS. The performance of the proposed system is tested with KDD Cup 1999 data set constructed using DARPA 1998 data set. Detection rate is between 90-93% for different attack types, and this result is not better than our system

performance. They don’t show that their system works faster. But in our proposed system we show that the system speed is increased by 30%.

In [9], DFA based IDS is proposed. Rule patterns are rewritten in order to reduce memory requirements. They test their proposed method with DARPA 2000 data set. They claim that their system achieved a factor of 12 to 42 performance improvement compared with other DFA based systems. However, the detection rate and false positive rates do not mention. It is important to increase detection speed while keeping the detection rate high enough. With our rules the system speed is increased by 30% also the detection rate remain high enough nearly 95%.

In [10], a hybrid IDS is proposed. In anomaly part they use Self Organizing Map structure to find normal behavior. In misuse part they define signatures by using some packet header information. The proposed system is tested with KDD Cup 1999 data set and detection rate is 99.90%. The result is a bit higher than our results. However there is no other information about throughput of the system. We increas the system speed by 30%.

In [11], anomaly based IDS is proposed. In order to define abnormal activity Support Vector Machines approach is used. Besides, some packet header information is used in order to define normal traffic. The proposed system performance is tested with DARPA 1999 data set. The result is worse than our proposed system. The detection rate is below 90% and also there is no evidence about the improvement of throughput.

In [12] [13], data mining based IDSs are proposed. They both use Support Vector Machines algorithm in order to reduce feature set. The proposed systems are tested with KDD Cup 1999. The detection rate of the proposed system is similar to ours. They are between 68-100% for different attack types in [12] and overall nearly 95% in [13]. But there is no information about throughput of the systems. With our rules the system speed is increased by 30% also the detection rate remain high enough nearly 95%.

In [14], anomaly and network based IDS is proposed. To classify the traffic Support vector machines method is used. The proposed system is tested with DARPA 1998 data set. DOS and probe types of attacks are detected with high rate upper than 90% but the overall performance is lower than 70%. The detection performance of the system is much lower than our proposed system. Also there is no information about throughput of the systems. With our rules the system speed is increased by 30% also the detection rate remain high enough nearly 95%.

In [15] [16], rule based IDSs are proposed. In [15], automatic tuning approach is proposed. In [16], they propose genetic based machine learning to dynamically update rules. The proposed systems are tested with KDD Cup 1999 data set. The overall detection rates are 95% and 92% respectively. The detection performances are nearly same with our proposed system performance. But there is no information about throughput of the proposed systems.

In [17], a false positive rate is decreased while filtering the alerts. Snort rule set and DARPA 1999 data set are used to test system performance. In results, they claim that false positive rate is reduced by 63%. There is no other information about detection capability and system throughput capacity.

In [18], packet header information is analyzed. They say that the packet payload inspection in heavy traffic condition is unnecessary. They compare their system with Snort rule set. They claim that their proposed system time performance is increased by 6.5%. However there is no information about detection rate. With our rules the system speed is increased by 30% also the detection rate remain high enough nearly 95%.

Table 2 Summary of related works

Ref.	Feature	Methodology	Data set	Accuracy (%)	Header or Payload
[4]	Misuse and machine learning and statistical techniques are used.	Hybrid based	KDD Cup 1999	53.08%	Header
[5]	Host machine system calls are used.	Host based	DARPA 1998	93%	Payload
[6]	A subset of significant feature set is selected.	Data Mining based Misuse based	DARPA 1998	Normal: 100 Probe:100 DOS:100 U2R: 84 R2L: 99,47	Header
[7]	Artificial Neural Network and Support Vector Machines techniques are used.	Data Mining based Network based	DARPA 1998	99,80-99,2	Header
[8]	It is a feature selection algorithm.	Neural network based Anomaly based	KDD Cup 1999	90,94-93,48	Header

[9]	In order to reduce memory requirements, rule patterns are rewritten.	DFA based Misuse based	DARPA 2000	-	Payload
[10]	To find normal behavior, Self Organizing Map structure is used.	Hybrid based	KDD Cup 1999	99,90	Header
[11]	To detect abnormal activity Support Vector Machines approaches is used.	Anomaly based	DARPA 1999	87,74	Header
[12]	Decision Trees and Support Vector Machines are combined.	Data Mining based Hybrid based	KDD Cup 1999	Normal: 99,70 Probe:100 DOS:99,92 U2R: 68 R2L: 97,16	Payload
[13]	In order to eliminate unimportant feature set, feature selection algorithm SVM is used.	Data Mining Misuse based	KDD Cup 1999	DOS: 99,53 Probe:97,55 U2R:19,73 R2L:28,81 Overall:95,72	Payload

[14]	In order to classify the traffic Support Vector Machines method is used.	Anomaly based Network based	DARPA 1998	Normal: 95 Probe:91 DOS:97 U2R: 23 R2L: 43 Avg: 69,8	Payload
[15]	Automatic tuning approach is proposed.	Misuse based	KDD Cup 1999	95	Payload
[16]	Rules are dynamically updated which called genetic-based machine learning.	Misuse based	KDD Cup 1999	92,03	Payload
[17]	Filtering the alert, false positive rates are decreased. It is compared with Snort rule set.	Network based	DARPA 1999	63% reduced false positive rates.	Payload
[18]	Packet header information is analyzed. It is compared with Snort rule set.	Misuse based	DARPA 1999	The system performance is increased by 6.5%	Header

2.6 Network-based IDS (NIDS) and Snort

2.6.1 Network-based IDS (NIDS)

Network based intrusion detection systems (NIDS) analyze network traffic to identify unauthorized access and abnormal traffic. NIDS collects packets in a given network and matches it with predefined attack signature database. If the system finds matches, give an alert to the system administrator. A well-known example of NIDS is Snort.

2.6.2 Snort

Snort is a free and open source network intrusion detection system (NIDS) which is developed by Sourcefire. In 2009, Snort entered InfoWorld's Open Source Hall of Fame as one of the "greatest open source software of all time." [19]

In 2006, Gartner's Magic Quadrant for Network Intrusion Prevention System Appliances listed Sourcefire as one of 5 leaders in this network security market sector. The others consist of 3com, Tipping Point, IBM, McAfee, and Juniper Networks [20].

In 2010, Sourcefire was placed as a leader in the Gartner report "Magic Quadrant for Network Intrusion Prevention Systems". Also only three companies out of 14 received this recognition from Gartner [21].

Snort is the most popular and widely used NIDS in worldwide. There are about 300,000 registered users. Also Snort has been downloaded approximately 15 million times so far [22].

Developers of Snort chose their tool to be open source project influencing from Richard Stallman who is founders of open source movement. Open source means that all software should have source code available and can be developed by some communities of interested developers. With supports of open source community, Snort is improved and become the leader in some reports in intrusion detection area [23].

There exists thousands of qualified programmers reviewing and testing the functionality of the Snort engine and rule sets. All bugs and other

problems can be detected and solved easily from Snort user community in worldwide in contrast to “closed” programs and projects [23].

Therefore, Snort is chosen as a framework and used for analysis and tests in our intrusion detection approach in this thesis.

2.6.2.1 Components of snort

Snort consists of some important components. These are shown in the following [24]:

- Packet Decoder
- Preprocessors
- Detection Engine
- Logging and Alerting System
- Output Modules

Figure 5 shows how these components are organized. Packet coming from the Internet first goes to packet decoder part of the Snort and then travels through preprocessor, detection engine and logging and alerting system. And then, after analyzed the packet either dropped or cause to generate an alert.

In Table 3, each components function is summarized.

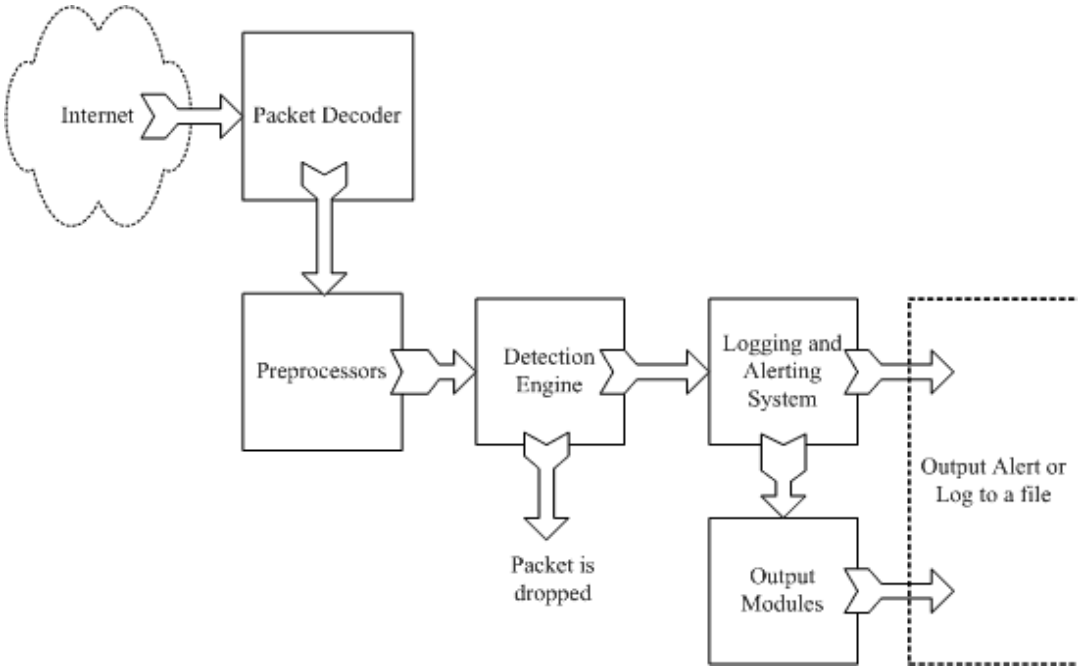

Figure 5 Components of Snort [24]

Table 3 Definitions of Snort components

Name	Description
Packet Decoder	Prepares packets for processing.
Preprocessors or Input Plugins	Used to normalize protocol headers, detect anomalies, packet reassembly and TCP stream reassembly.
Detection Engine	Matches rules with packet data.
Logging and Alerting System	Generates alert and log messages.
Output Modules	Process alerts and logs and generate final output.

The detailed information about Snort and its modes of work, alert output, high performance configuration and rules structure is in Appendix C.

2.6.2.2 Base and Barnyard

BASE means the Basic Analysis and Security Engine. The former name of the project was the Analysis Console for Intrusion Databases (ACID). This program gives a very user friendly web interface to analyze Snort alerts. It has a user authentication and role-base system. It is also an open source project so it is supported by open source community users [25].

Some screenshots is shown in Figure 6 and Figure 7.

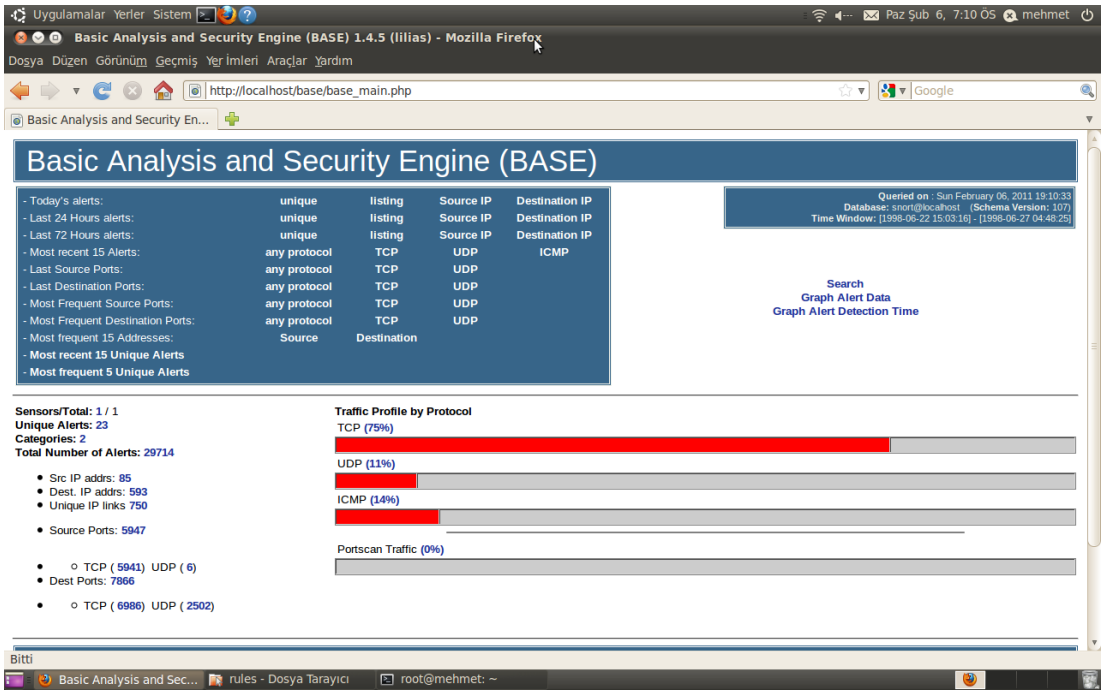

Figure 6 BASE main screen

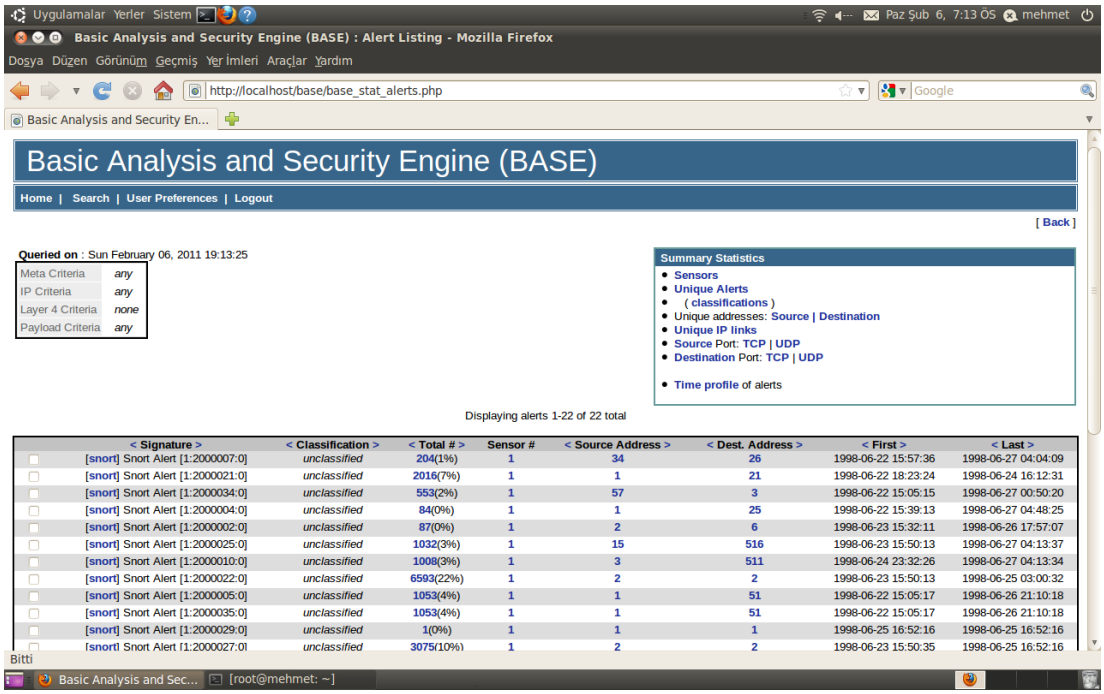

Figure 7 BASE alert screen

Barnyard is an output plugin for Snort. It reads the alert log file created from Snort with binary format and writes the data to the database [26].

This is really useful plugin. Because Snort can log rapidly in binary format and while Barnyard writing these data to database, Snort spends time processing network traffic data. Thus the performance of snort increases [27].

2.6.3 Wireshark

Wireshark is also a free and open source project which is used to analyze network packets. Its former name was Ethereal.

Wireshark supports almost all protocol types used in computer network such as TCP, UDP, IP, ICMP, DHCP, ARP, DNS. Nowadays, it is so popular that it was the Source Forge Project of the Month in August 2010 [28].

The screenshot of the wireshark program is shown in Figure 8.

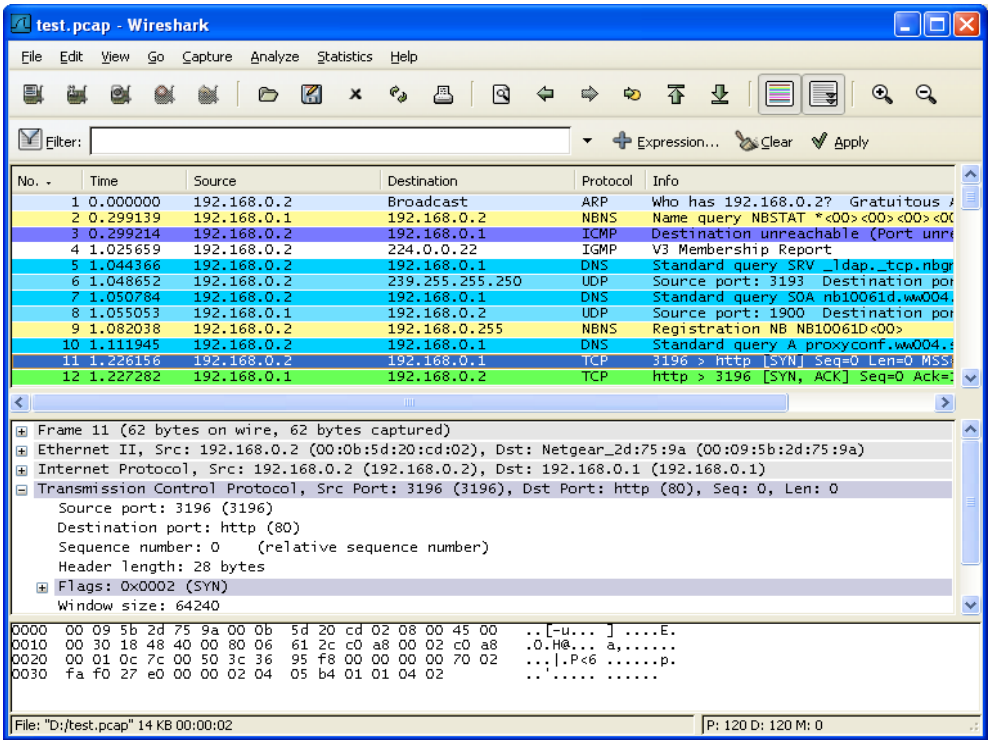

Figure 8 Wireshark screen [29]

CHAPTER 3

OUR FAST INTRUSION DETECTION APPROACH

In this chapter, we present developing steps of our fast intrusion detection method. First, the attack types in the DARPA Data set are analyzed and their characteristics, which are IP and transport layer header information, are listed in Table 4. And then by using Snort program, we construct attack signatures based on these headers information and formulate with Snort rules to detect these attacks. Our goal is speed up the intrusion detection system by decreasing the detection time and to keep detection rate high enough and false alarm rate low enough.

3.1 Our Approach

In order to construct our approach we follow the steps shown below.

1. The tcpdump files of DARPA 1998 data sets are downloaded and analyzed in Wireshark program.
2. Common packet header features for all type of attacks are noted.
3. Using these features, rules are constructed and formulated with Snort rule type.
4. After we change the Snort configuration files, we test our constructed rules.
5. If the test gives high false alarms or poor detection performance, further analysis is needed.

There are lots of different attacks in DARPA 1998 data set but some of them are obsolete today. So we generate some attacks by using some attack tools to test and update our constructed rule set.

In order to detect attacks we use IP and transport layer header information which is shown in Table 4.

Table 4 Used IP and Transport Layer Header fields

Source port	Port numbers ranges from 0 to 65536.
Destination port	Port numbers ranges from 0 to 65536.
Tos	Type of Service value.
Total length	Packet total length in bytes.
IP flags	Do Not Fragment (D), More Fragment (M)
Ttl	Time to live value.
TCP flags	Ack(A), Push(P), Reset(R), Syn(S), Fin(F)
Window	The tcp window scale option.
ICMP type	Type of icmp packet.
ID	Ip id number

In the rest of this chapter we present the list of attacks that we extract from the DARPA data set and their respective signatures that only depend on the packet headers.

3.1.1 Denial of Service Attacks

3.1.1.1 Back

It is a denial of service attack destined to web server port TCP 80. It includes many back slashes in the packet payload. The web servers were vulnerable and used to stop working in old times [30].

Table 5 Attack Signatures of Back Attack

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
Any	80	0	>1400	D	64	PA	-	-

3.1.1.2 Land

These attacks occur when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [30].

Table 6 Attack Signature of Land Attack

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
-	-	-	-	-	-	-	-	-

3.1.1.3 Neptune

It is a SYN Flood denial of service attack. The attacker sends a SYN packet and then the victim machine reply with SYN ACK and waits for ACK packet to establish a TCP connection. But the attacker does not send an ACK packet and continue to send SYN packet to fill the TCP half open connection buffer of victim machine. So the victim machine become out

of service and does not accept any new connection attempts. So the system becomes out of service.

Table 7 Attack Signature of Neptune Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
Any	any	-	<40	-	255	S	242	-

3.1.1.4 Ping of death

The Ping of Death is a denial of service attack that has larger packet size than 65535 bytes. Sending the large ping packets may crash the victim machine [31].

Table 8 Attack Signature of Ping of death Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
-	-	-	1480	-	-	-	-	-

3.1.1.5 Syslog

It is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. The Solaris operating systems are not vulnerable now [30].

Table 9 Attack Signature of Syslog Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
514	514	-	<10		64	-	-	-

3.1.1.6 Teardrop

It is a denial of service attack that the attacker sends malformed IP fragments with overlapping [30].

Table 10 Attack Signature of Teardrop Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type	Id
Any	any	-	28	M	-	-	-	-	242

3.1.2 User to root attacks

3.1.2.1 Eject

The Eject attacks destined to Solaris operating systems. The attacker has a user account and gain to root in victim machine by using eject program [32].

Table 11 Attack Signature of Eject Attack

Source port	Destination port	Tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
Any	23	16	58	D	64	PA	32120	-

3.1.2.2 Ffbconfig

The Ffbconfig attack is also destined to Solaris operating system. It use ffbconfig program to generate the attack [30].

Table 12 Attack Signature of Ffbconfig Attacks

Source port	Destination port	Tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	any	0	71	D	254	PA	8760	-

3.1.2.3 Format

The Format attack is also destined to Solaris operating systems. It use format program to generate the attack. The format program formats diskettes and PCMCIA memory cards [30].

Table 13 Attack Signature of Format Attacks

Source port	Destination port	Tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
any	any	0	71	D	254	PA	8760	-

3.1.2.4 Loadmodule

The Loadmodule attack is destined to SunOS systems. It use the loadmodule program within SunOS. With this attack unauthorized users can gain root access on the local machine [30].

Table 14 Attack Signature of Loadmodule Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
any	23	-	Between 40 and 50	D	64	PA	32696	-

3.1.2.5 Perlmagic

The Perlmagic attack uses Suidperl script to generate the attack in some Perl implementations. With this attack anyone with access to an account on the system can gain root access [30].

Table 15 Attack Signature of Perlmagic Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	23	-	19	D	64	PA	32120	-

3.1.2.6 Rootkit

A rootkit is software that enables continued privileged access to a computer while actively hiding its presence from administrators by subverting standard operating system functionality or other applications. An attacker installs a rootkit on a computer after first obtaining root-level access, either by exploiting a known vulnerability or by obtaining a password. Once a rootkit is installed, it allows an attacker to mask the ongoing intrusion and maintain privileged access to the computer [33].

Table 16 Attack Signature of Rootkit Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	any	-	Between 50 and 100	-	63	-	-	8
>1000	>1000	-	Between 4 and 40	-	-	-	-	-

3.1.3 Remote to user attacks

3.1.3.1 Dict

The Dict attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords [30].

Table 17 Attack Signature of Dict Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
20,23,110, 143,513	>1000	-	Between 17 and 20	-	-	PA	-	-

3.1.3.2 Ftp-write

The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of

these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system.

Table 18 Attack Signature of Ftp-write Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	Window	ICMP type
20,21	>1000	0	-	D	254	S	24820	-
Between 1000 and 1100	20,21,513	0	-	D	63	S	512	-

3.1.3.3 Guest

The Guest attack is a variant of the Dictionary attack. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [30].

Table 19 Attack Signature of Guest Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window
20,23,110,143,513	>1000	-	Between 17 and 20	-	-	PA	-

3.1.3.4 Imap

The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [30].

Table 20 Attack Signature of Imap Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	Window	ICMP type
Any	143	-	>1000	D	-	PA	32120	-

3.1.3.5 Phf

The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [30].

Table 21 Attack Signature of Phf Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	80	-	Between 50 and 60	D	64	PA	32120	-

3.1.3.6 Spy

Spy attacks are multi-day scenario attacks in which a user breaks into a machine with the purpose of finding some important information where the user tries to avoid detection [32].

Table 22 Attack Signature of Spy Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
>10000	23	16	Between 45 and 50	-	64	PA	32120	-

3.1.3.7 Warez

In warez attack, user logs into anonymous FTP site and creates a hidden directory [32].

Table 23 Attack Signature of Warez Attacks

Source port	Destination port	tos	Total length	IP flags	ttl	Tcp flags	window	ICMP type
Any	21	0	<30	-	64	PA	32120	-

3.1.3.8 Warezmaster

In warezmaster attacks, anonymous FTP user uploads of illegal copies of copywrited software onto FTP server [27].

Table 24 Attack Signature of Warezmaster Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
20	>1000	-	<40	-	254	S	24820	-

3.1.3.9 Warezclient

In warezclient attacks, users download illegal software which was previously posted via anonymous FTP by the warezmaster [32].

Table 25 Attack Signature of Warezclient Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
>1000	21	-	<20	-	64	S	512	-

3.1.4 Probes

3.1.4.1 Ipsweep

An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines [30].

Table 26 Attack Signature of Ipsweep Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	<100(except 80,25)	-	-	-	64	S	512	-
Any	Any	-	<30	-	-	-	-	8

3.1.4.2 Nmap

Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of port scans—options include SYN, FIN and ACK scanning with TCP and UDP, as well as ICMP (Ping) scanning. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order [30].

Table 27 Attack Signature of Nmap Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	any	0	<40	-	254	F	2048	-
137,138	137,138	-	<250	-	63	-	-	-
Any	Any	-	-	-	63	-	-	8

3.1.4.3 Portsweep

In portsweep attacks, attacker sends packet to every port by using some tools to detect open services of victim machine [32].

Table 28 Attack Signature of Portsweep Attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
>1000	any(except 80)	-	<40	-	-	S	-	-

3.1.4.4 Satan

It is a network probing tool which looks for well-known weaknesses [32].

Table 29 Attack Signature of Satan attacks

Source port	Destination port	tos	Total length	IP flags	Ttl	Tcp flags	window	ICMP type
Any	any	-	<20	-	-	-	-	-
>1000	any(80 not included)	-	<40	-	64	S	-	-
any(53 not included)	any(53 not included)	0	<25	-	64	-	-	-

3.2 Rule set

By using both information in Section 3.1 and Snort tool, we construct our rules presented in Appendix A.

CHAPTER 4

EXPERIMENTAL EVALUATION

We compare our rule set with Snort default rules in two data sets. First data set is DARPA 1998 data set which is widely used training and testing intrusion detection system performance. Second data set is our own data set which is constructed in our experimental setup.

At first, DARPA 1998 data set is downloaded from the website. There are both tcpdump file for testing data set and tcpdump list file which shows the attack types and attack times. The data set has 16.5 GB of tcpdump file consists of seven weeks tcpdump files, 27 types of attacks and totally almost 300000 attacks.

Then we prepare our experimental environment. We use our notebook Acer emachines D730G. It has Intel Core i5-430M and 3 GB DDR3 Memory and the operating system is Windows 7.

Then a virtual machine software VMware is installed. With two virtual machines are installed on this VMware. One of the virtual machines has Ubuntu 10.04 operating system. And the other one is Backtrack which consist of very popular attack tools for hackers [34].

In Ubuntu 10.04 virtual machine, Snort and Base are installed to test rules performances. Also, the wireshark program is installed to inspect the downloaded data packet contains intrusion activity and to save attack traffic that we create. The entire rule set is constructed and improved by inspecting the data sets tcpdump file.

Our first data set is DARPA 1998 data set and in this data set first week has the lowest intrusions among the other weeks. And sixth week has the highest intrusions among the other weeks. The distribution of number of intrusions over the seven weeks is shown in Figure 9.

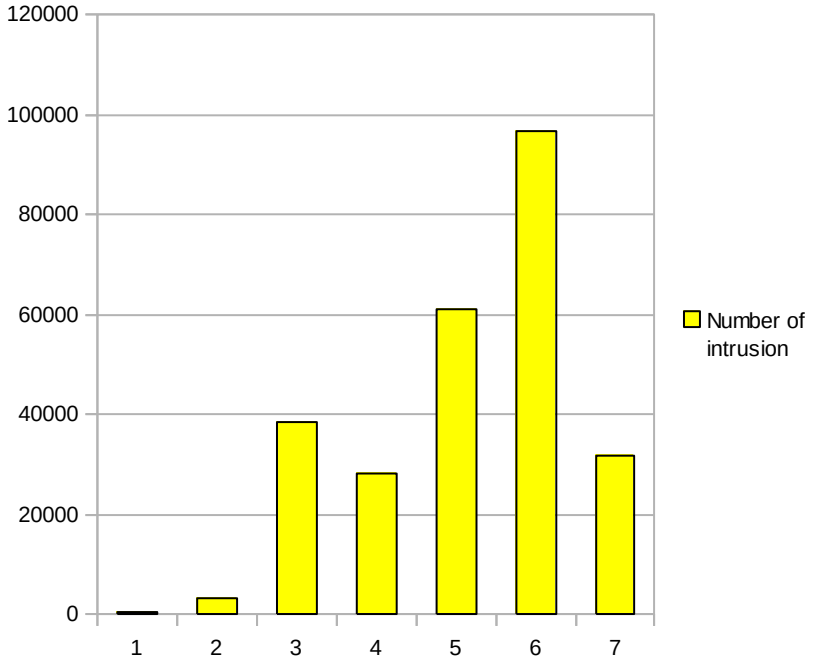

Figure 9 The distribution of number of intrusions in DARPA 1998

In this part we want to show the performance of our rule set for DARPA 1998 data set. The followings are the attacks in the DARPA 1998 data set:

Back: In the test data set there are 4381 back attacks. Our constructed rules detect all the back attacks, i.e. detection rate of back is 100%. Also our constructed rules detected 581 normal packets as a back attack, i.e. number of false positive is 581. False positive rate of back is 13.26%. This rate is not low enough because our rules detect a large packet destined to tcp port number 80 as a back attack and tcp port number 80 is commonly used for http.

Dict: In the test data set there are 882 dict attacks. And our constructed rules detected 631 of them, i.e. detection rate of dict is 71.54%.

Remaining 251 attacks are not detected, i.e. our rules miss these attacks. The detection rate of dict attack can be increase further by setting the threshold. In order to guess password of victim machines, this type of attack is repeated. We detect this type of attack by looking the victim machine response. When the attacker enters wrong password, the victim machine respond with a login failure message. This failure message came from tcp port number 20, 23, 110, 143, 513 and repeated a much in a limited time period.

Eject: The test data set consists of 11 eject attacks and 6 of them is detected and the remaining 5 is missed. So detection rate of eject attack is 54.54%.

Ffb: The test data set consists of 10 ffb attacks and all of them is detected, i.e. the detection rate is 100% but also our rules detect extra 24 attacks as a ffb attacks, i.e. a number of false positive is 24. We surprised that our rules really give a huge amount of false positive rate of ffb attack but thanks to all the attacks is detected. There are no missed attacks.

Format: Ffb and format types of attacks have same characteristics so we use one rule to detect these attacks. In test data set, there are 8 attacks and our rules detect all attacks so detection rate is 100%. And also there exist 5 false positive attacks of format. The false positive rate of format is lower than the ffb.

Ftp-write: There are 8 ftp-write attacks in the test data set and half of them are detected, i.e. detection rate is 50%. We construct two rules to detect this type of attacks because there are two different characteristic of this type of attacks. But our rules miss half of them.

Guest: Dict and guest attacks have some characteristic so we construct one rule for these attacks. In test data set there are 50 guest attacks and all of them are detected. It means the detection rate is 100%. But false positive number is 15 and the false positive rate is 30%. This rate is a little high but it is so important that there are no missed attacks.

Imap: In test data set there are 8 imap attacks. Our rules detect only 3 of them and miss the remaining 5. But there is a wrong in the data set because the imap attacks are a buffer overflow attacks and packet size must be large as possible as. But in data set some imap attacks labeled packet size is very low. So our detection rate is low.

Ipsweep: In test data set there are 16336 ipsweep attacks and all of them are detected. There are two different version of this attack. So we construct two rules for them. Furthermore, the false positive rate is very low (0.67%).

Land: In test data set there are 35 land attacks and our rule detect all the land rules because the IP addresses of both source and destination are same. So it can be easily detected.

Loadmodule: In test data set there are 9 loadmodule attacks but only 3 of them is detected. Again the test data set wrongly labeled some attacks so detection rate is very low (33.3%).

Neptune: In test data set there are 1526643 neptune attacks but we use only 10% of them in order to get true statistics otherwise almost all the attacks would consist of this type of attacks and overall results would not be meaningful. Our rules detect almost every Neptune attacks (99.98% of them are detected) and there are no wrong detected attacks.

Nmap: In test data set there are 2357 nmap attacks and 2022 of them is detected. The detection rate is 85.78%. Also there are some missed attacks. But there are no wrong detected attacks. Nmap attacks have 3 different version, tcp, udp and icmp types. All udp and icmp types of nmap attacks are detected but there is a miss with tcp type of attacks.

Perlmagic: In test data set, there are 5 perlmagic attacks but our rule set detects 21 attacks as a perlmagic. All the perlmagic attack is detected but the false positive rate is surprised us. It is very huge.

Phf: In test data set, there are 5 phf attacks and all of them are detected. The detection rate is 100%. Furthermore there are no missed and wrong labeled attacks, i.e. the false positive rate is 0%.

Pod: In test data set, there are 10498 pod attacks and all of them are detected. The detection rate is 100%. Besides the false positive rate is 0%.

Portsweep: In test data set, there are 10616 portsweep attacks and 1473 of them is detected. The detection rate is so low (13.87) because portsweep attacks have lots of different types and only some of them is formulated as a rule. In order to increase this detection rate, all other types must be defined exactly in different rules.

Rootkit: In test data set, there are 254 rootkit attacks and 243 of them are detected. So the detection rate is 95.66 and 11 rootkit attacks is missed. Rootkit attacks have also two different versions: udp and icmp. Therefore, two rules are constructed to detect this type of attacks. Icmp versions of these attacks are detected easily but some udp versions of these attacks are not detected.

Satan: In test data set, there are 32625 satan attacks and all of them is detected. However false positive rate is not low enough (13.25%). Because satan have three different versions: icmp, udp and tcp. We construct three rules for them but two rules for tcp and udp increase false positive rate of this type of attacks.

Spy: In test data set, there are only 2 spy attacks and one of them is detected. The other is missed because it is a mislabeled attack.

Syslog: In test data set, there are 4 syslog attacks and all of them are detected and also there are no missed and mislabeled attacks, i.e. detection rate is 100% and false positive rate is 0%. The detection rate is huge because this attacks udp source and destination port number is equal to 514.

Teardrop: In test data set, there are 2173 teardrop attacks and 1027 of them is detected so the detection rate is 47.26%. There are no wrong labeled attacks, i.e. the false positive rate of these attacks is 0%. However some of these attacks are not detected.

Warez: In test data set, there is only one warez attack and it is detected. Besides there is no wrong labeled attacks.

Warezclient: In test data set, there are 1766 warezclient attacks and 805 of them are detected. The detection rate is 45.58% and there is no wrong labeled attack, i.e. the false positive rate is 0%.

Warezmaster: In test data set, there are 19 warezmaster attacks and 18 of them are detected. The detection rate is 94.73% and there is no wrong labeled attack, i.e. the false positive rate is 0%.

Totally in DARPA 1998 data set, there are 234728 attacks and 227915 of them are detected and 5078 of them is false positive (wrong detection) and 11891 of them is false negative (missed). The overall detection performance of our rules is 94.93% and false positive rate is 2.16%. We summarize our test results in Table 30.

Table 30 Our rules performance with DARPA 1998 data set

		Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	**4381**	4962	581	0	100,00%	13.26%
dict	**882**	631	0	251	71.54%	0,00%
eject	**11**	6	0	5	54.54%	0,00%
ffb	**10**	34	24	0	100,00%	240,00%
format	**8**	13	5	0	100,00%	62.50%
Ftp-write	**8**	4	0	4	50,00%	0,00%
guest	**50**	65	15	0	100,00%	30,00%
imap	**8**	3	0	5	37.50%	0,00%
ipsweep	**16336**	16447	111	0	100,00%	0.67%
land	**35**	35	0	0	100,00%	0,00%
loadmodule	**9**	3	0	6	33.33%	0,00%
neptune	**152665**	152643	0	22	99.98%	0,00%
nmap	**2357**	2022	0	335	85.78%	0,00%
perlmagic	**5**	21	16	0	100,00%	320,00%
phf	**5**	5	0	0	100,00%	0,00%
pod	**10498**	10498	0	0	100,00%	0,00%
portsweep	**10616**	1473	0	9143	13.87%	0,00%
rootkit	**254**	243	0	11	95.66%	0,00%
satan	**32625**	36951	4326	0	100,00%	13.25%
spy	**2**	1	0	1	50,00%	0,00%
syslog	**4**	4	0	0	100,00%	0,00%
teardrop	**2173**	1027	0	1146	47.26%	0,00%
warez	**1**	1	0	0	100,00%	0,00%
warezclient	**1766**	805	0	961	45.58%	0,00%
warezmaster	**19**	18	0	1	94.73%	0,00%
Total	**234728**	**227915**	**5078**	**11891**	**94.93%**	**2.16%**

After looking attacks by attacks performance, we group the attacks in Table 31 and give the results in Table 32. Our rules detection rate for DOS and U2R types of attacks more than 90% and also Probe attacks are nearly 85% but in R2L type attack this rate is below to 60%.

The meaning of the result is that DOS, U2R and Probes attacks can be detected with headers information with high detection rate. Besides U2R and Probes attacks gives some false alarms. Also the header detection rate of R2L attacks is low.

Table 31 Grouping the attacks in DARPA 1998 data set

Attack Types	**Attack Names**
DOS	Back, Land, Neptune, Pod, Syslog, Teardrop
U2R	Eject, Ffbconfig,Format,Loadmodule,Perlmagic,Rootkit
R2L	Dict, Ftp-write, Guest, Imap, Phf, Spy, Warez, Warezmaster, Warezclient
Probes	Ipsweep, Nmap, Portsweep, Satan

Table 32 Our rules performance with DARPA 1998 data set

		Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**169756**	169169	581	1168	99.31%	0.34%
U2R	**297**	320	45	22	92.59%	15.15%
R2L	**2741**	1533	15	1223	55.38%	0.54%
Probes	**61934**	56893	4437	9478	84.69%	7.16%

Then Snort default rules are tested in DARPA data set. Their performance is shown in Table 33. Our rules only look for header part of the packet, and this causes to decrease the detection rate and increase the false detection rate of our rules.

Table 33 Snort default rules performance with DARPA 1998 data set

		Snort Default ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	**4381**	4366	0	15	99.65%	0,00%
dict	**882**	822	72	132	93.19%	8,16%
eject	**11**	0	0	11	0,00%	0,00%
ffb	**10**	1	0	9	10,00%	0,00%
format	**8**	0	0	8	0,00%	0,00%
Ftp-write	**8**	0	0	8	0,00%	0,00%
guest	**50**	20	0	30	40,00%	0,00%
imap	**8**	0	0	8	0,00%	0,00%
ipsweep	**16336**	14359	0	1977	87.89%	0,00%
land	**35**	35	0	0	100,00%	0,00%
loadmodule	**9**	0	0	9	0,00%	0,00%
neptune	**152665**	152278	74	461	99.74%	0.3%
nmap	**2357**	2348	0	9	99.61%	0,00%
perlmagic	**5**	0	0	5	0,00%	0,00%
phf	**5**	0	0	5	0,00%	0,00%
pod	**10498**	10494	0	4	99.96%	0,00%
portsweep	**10616**	10584	0	32	99.69%	0,00%
rootkit	**254**	250	0	4	98.42%	0,00%
satan	**32625**	32580	0	45	99.86%	0,00%
spy	**2**	0	0	2	0,00%	0,00%
syslog	**4**	0	0	4	0,00%	0,00%
teardrop	**2173**	1901	0	272	87.48%	0,00%
warez	**1**	1	0	0	100,00%	0,00%
warezclient	**1766**	1187	37	616	67.21%	2.09%
warezmaster	**19**	18	0	1	94.73%	0,00%
Total	**234728**	**231244**	**183**	**3667**	**98.48%**	**0.07%**

After looking attacks by attacks performance, we show the results in a group in Table 34. Snort default rules detection rate in DOS and Probe types of attacks more than 95% and also U2R attacks are nearly 85% but in R2L type attack this rate is about to 70%.

Table 34 Snort default rules performance with DARPA 1998 data set

		Snort Default ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**169756**	169074	74	756	99.55%	0.04%
U2R	**297**	251	0	46	84.51%	0,00%
R2L	**2741**	2048	109	802	70.74%	3.97%
Probes	**61934**	59871	0	2063	96.66%	0,00%

The DARPA 1998 experiment data has 16.5 GB of tcpdump file. In our rules it takes 708 seconds to test, but in default configuration of snort it takes 1025.1 seconds. The speed is increased by 30.9%.

Also, Snort default rule set detects 16 type attacks of 27 type attacks in DARPA 1998 data set. But our rule set detects25 type attacks of 27 type attacks in DARPA 1998 data set. Attack types not detected from Snort default rule set have very low number in total. So the overall result of Snort default rule set is remaining high.

Portsweep attacks detection performance of our rule set is very low. Because there are lots of different versions of these attacks. Our rules only detect some of them. But Snort default rule set detect these attacks in very huge detection rate. All the other attack types' detection performance of Snort default rule set and our rule set is nearly same.

Some attacks are not detected by the Snort default rule set because they are not widely used in nowadays and the victim operating systems not vulnerable to these attack types any more.

We use DARPA 1998 data set since there is no other publicly available data set to test and improve our intrusion detection method and also all works related these topics use this data set. But DARPA 1998 data set is old and contains some obsolete attacks. Computer attacks are changing very frequently and intrusion detection system rules must be updated very frequently. Hence we artificially generate attacks and construct a test data set different than DARPA to compare Snort rule set performance and our rule set performance.

4.1 Generating Our Data set

Our second test data set is our constructed data set. We simulate the attacks by using Bactrack virtual machine in VMware. Bactrack is a GNU/Linux distribution consists of security-related tools such as scanners and password crackers. It mainly use for digital forensics and penetration testing [34].

For victim machine we use Ubuntu 10.04 virtual machine in VMware. Then, the following attacks are created to the victim machine. In victim machine the wireshark program is working while the attacks happening to save attacked traffic data.

We use hping tool which can produce every packet type (tcp, udp and icmp) and Brutessh script to generate the following attacks and construct data set. Our constructed data set contains attacks which are known to be common and regular. The experiment data has 1.2 MB of pcap file.

We construct the following attacks since they are popular today and threaten to every computer systems. The attack tool Hping is also very popular for hackers to generate attacks. And also it is a good tool for system administrator to test their systems performance.

The commands are used to generate the following attacks as shown in Appendix D.

Dict attack

This is the dictionary attack. A password list is constructed to crack the root password of the victim machine and generate attacks.

Land attack

In this attack source and destination ip address is the same. In these attack TCP flag SYN is set.

Neptune attack

This type of attacks is a SYN flood attack. The attack packet is generated with ttl value equal to 255 and window size equal to 242. Also TCP SYN flag is set.

Pod attack

These types of attack consist of icmp echo ping packet. But ping packets default sizes are 32 bytes and there is no problem. In these attacks the ping packet size is increased.

Teardrop attack

These types of attacks have mis-fragmented UDP packets. This type of attacks is generated with packet IP id number equal to 242 and more fragments bit is set. Also type of the packet is UDP.

Ipsweep attack

These types of attacks are two types. One type of this is scanning port. The other type is generating icmp echo ping packet.

Nmap attack

These types of attacks consist of three types. These are tcp, udp and icmp types. Nmap attack is used to detect victim machine operating system.

Portsweep attack

These types of attacks are port scanning attacks to understand victim machines open ports. The attack is generated with sending TCP SYN packet to every port.

Satan attack

These types of attacks consist of three types generally. These are icmp, tcp and udp types.

After collecting the traffic data, we test Snort default rule set and our rule set by typing the following commands for each saved attack data.

snort –r [attack pcap filename] –c /etc/snort/snort.conf

snort –r [attack pcap filename] –c /etc/snort/snort1.conf (our modified configuration file)

The test results of our rule set are shown in Table 35. All the attacks are detected with a high detection rate and low false alarm rate.

Table 35 Our rules performance with our data set

	# of attacks	Our ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	-	-	-	-	-	-
dict	7	7	0	0	100,00%	0,00%
eject	-	-	-	-	-	-
ffb	-	-	-	-	-	-
format	-	-	-	-	-	-
Ftp-write	-	-	-	-	-	-
guest	-	-	-	-	-	-
imap	-	-	-	-	-	-
ipsweep	452	454	2	0	100,00%	0.44%
land	84	84	0	0	100,00%	0,00%
loadmodule	-	-	-	-	-	-
neptune	160	160	0	0	100,00%	0,00%
nmap	199	199	0	0	100,00%	0,00%
perlmagic	-	-	-	-	-	-
phf	-	-	-	-	-	-
pod	272	272	0	0	100,00%	0,00%
portsweep	96	97	1	0	100,00%	1.03%
rootkit	-	-	-	-	-	-
satan	264	264	1	1	99.62%	0,37%
spy	-	-	-	-	-	-
syslog	-	-	-	-	-	-
teardrop	153	153	0	0	100,00%	0,00%
warez	-	-	-	-	-	-
warezclient	-	-	-	-	-	-
warezmaster	-	-	-	-	-	-
Total	**1687**	**1690**	**4**	**1**	**99.94%**	**0.23%**

We group the attacks in Table 36 and give the results in Table 37. Our rules detection rate performance is 100% in DOS and R2L types and also Probe attacks are nearly 100%.

Table 36 Grouping the our constructed attacks

Attack Types	Attack Names
DOS	Land, Neptune, Pod, Teardrop
U2R	-
R2L	Dict
Probes	Ipsweep, Nmap, Portsweep, Satan

Table 37 Our rules performance with our data set

	# of attacks	Our ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**669**	669	0	0	100,00%	0,00%
U2R	-	-	-	-	-	-
R2L	**7**	7	0	0	100,00%	0,00%
Probes	**1011**	1014	4	1	99.90%	0.39%

Then we test Snort default rule set performance with our constructed data set. The test results are shown in Table 38. Snort default rule set also detects all the attacks with high detection rate and only gave 2 false positives.

Table 38 Snort default rule performance with our data set

	# of attacks	Snort Default ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	-	-	-	-	-	-
dict	**7**	7	0	0	100,00%	0,00%
eject	-	-	-	-	-	-
ffb	-	-	-	-	-	-
format	-	-	-	-	-	-
Ftp-write	-	-	-	-	-	-
guest	-	-	-	-	-	-
imap	-	-	-	-	-	-
ipsweep	**452**	454	2	0	100,00%	0.44%
land	**84**	84	0	0	100,00%	0,00%
loadmodule	-	-	-	-	-	-
neptune	**160**	160	0	0	100,00%	0,00%
nmap	**199**	199	0	0	100,00%	0,00%
perlmagic	-	-	-	-	-	-
phf	-	-	-	-	-	-
pod	**272**	272	0	0	100,00%	0,00%
portsweep	**96**	96	0	0	100,00%	**0,00%**
rootkit	-	-	-	-	-	-
satan	**264**	264	0	0	100,00%	0,00%
spy	-	-	-	-	-	-
syslog	-	-	-	-	-	-
teardrop	**153**	153	0	0	100,00%	0,00%
warez	-	-	-	-	-	-
warezclient	-	-	-	-	-	-
warezmaster	-	-	-	-	-	-
Total	**1687**	**1689**	**2**	**0**	**100,00%**	**0.11%**

We show the results in a group in table 39.

Table 39 Snort default rule performance with our data set

	# of attacks	Snort Default ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**669**	669	0	0	100,00%	0,00%
U2R	**-**	-	-	-	-	-
R2L	**7**	7	0	0	100,00%	0,00%
Probes	**1011**	1013	2	0	100,00%	0.19%

If we compare our rule set performance with Snort default rule set performance for our constructed data set, we can say that the results are nearly same. Snort default rules detection rate is a little higher and false alarm rate is a little lower than our rules. Since Snort default rule set inspects full packet payloads to find signatures. But our rule set inspects only header part of packet.

In our rules it takes 2.26 seconds to test but in default configuration of snort it takes 5.03 seconds. The speed is increased by 55.06%.

4.2 Enhancing our Payload Independent Rule Set with Modified Default Snort Rules

Snort default rule set is a well-accepted rule set. We next consider modifying these rules to enhance our new manually designed rules.

First, 53 different rule types of Snort default rule set are combined a file named snort_all. Second, payload detection rule options are excluded from this file.

In Snort default rule set, there exist 3382 rules. After extracting header rules 66 rules remain in rule set. To extract header rules we use some UNIX commands presented in Appendix E.

After extracting header rules, first we test the rule set with DARPA 1998 data set. The results are shown in Table 40. The results show that Snort

header rules detection performance is decreased from 98.48% to 82.23% compared with Snort default rule set performance. However the time detection performance is increased. While the Snort default rules takes 1025.1 seconds, the Snort header takes 728 seconds. Because of rule reduction some attacks are missed.

Table 40 Snort header rules performance with DARPA 1998 data set

		Snort header ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate	false alarm rate (%)
back	**4381**	41	0	4340	0,93%	0,00%
dict	**882**	7	0	875	0,79%	0,00%
eject	**11**	0	0	11	0,00%	0,00%
ffb	**10**	0	0	10	0,00%	0,00%
format	**8**	0	0	8	0,00%	0,00%
Ftp-write	**8**	0	0	8	0,00%	0,00%
guest	**50**	2	0	48	4,00%	0,00%
imap	**8**	0	0	8	0,00%	0,00%
ipsweep	**16336**	14326	0	2010	87,69%	0,00%
land	**35**	35	0	0	100,00%	0,00%
loadmodule	**9**	0	0	9	0,00%	0,00%
neptune	**152665**	132460	0	20205	86,76%	0,00%
nmap	**2357**	2037	0	320	86,42%	0,00%
perlmagic	**5**	0	0	5	0,00%	0,00%
phf	**5**	0	0	5	0,00%	0,00%
pod	**10498**	5408	0	5090	51,51%	0,00%
portsweep	**10616**	8405	0	2211	79,17%	0,00%
rootkit	**254**	6	0	248	2,36%	0,00%
satan	**32625**	28315	0	4310	86,78%	0,00%
spy	**2**	0	0	2	0,00%	0,00%
syslog	**4**	0	0	4	0,00%	0,00%
teardrop	**2173**	893	0	1280	41,09%	0,00%
warez	**1**	1	0	0	100,00%	0,00%
warezclient	**1766**	1077	0	689	60,98%	0,00%
warezmaster	**19**	18	0	1	94,73%	0,00%
Total	**234728**	**193031**	**0**	**41697**	**82.23%**	**0,00%**

After looking attacks by attacks performance, we show the results in a group in Table 41.

Table 41 Snort header rules performance with DARPA 1998 data set

		Snort header ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate	false alarm rate (%)
DOS	**169756**	138837	0	30919	81,78%	0,00%
U2R	**297**	6	0	291	2,02%	0,00%
R2L	**2741**	1105	0	1636	40,31%	0,00%
Probes	**61934**	53083	0	8851	85,70%	0,00%

The results show that DOS and Probes attacks can be detected with high rate by looking header part of the packet. However R2L attacks are needed payload inspection. The U2R attacks detection rate is very low because lots of these types of attacks are obsolete and not detected.

After that snort header rules and our rules are combined and tested with DARPA 1998 data set again. The results show in Table 42. The detection rate is increased compared with Snort header rules. Because some missed attacks are detected with our rules.

Table 42 Snort header rules combined with our rules performance with DARPA 1998 data set

		Snort header ruleset performance + Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	**4381**	4962	581	0	100,00%	13.26%
dict	**882**	631	0	251	71,54%	0,00%
eject	**11**	6	0	5	54,54%	0,00%
ffb	**10**	34	24	0	100,00%	240,00%
format	**8**	13	5	0	100,00%	62.50%
Ftp-write	**8**	4	0	4	50,00%	0,00%
guest	**50**	65	15	0	100,00%	30,00%
imap	**8**	3	0	5	37,50%	0,00%
ipsweep	**16336**	16447	111	0	100,00%	0.67%
land	**35**	35	0	0	100,00%	0,00%
loadmodule	**9**	3	0	6	33,33%	0,00%
neptune	**152665**	152643	0	22	99,98%	0,00%
nmap	**2357**	2345	0	12	99,49%	0,00%
perlmagic	**5**	21	16	0	100,00%	320,00%
phf	**5**	5	0	0	100,00%	0,00%
pod	**10498**	10498	0	0	100,00%	0,00%
portsweep	**10616**	9311	0	1305	87,70%	0,00%
rootkit	**254**	243	0	11	95,66%	0,00%
satan	**32625**	36951	4326	0	100,00%	13.25%
spy	**2**	1	0	1	50,00%	0,00%
syslog	**4**	4	0	0	100,00%	0,00%
teardrop	**2173**	1027	0	1146	47,26%	0,00%
warez	**1**	1	0	0	100,00%	0,00%
warezclient	**1766**	1253	0	513	70,95%	0,00%
warezmaster	**19**	18	0	1	94,73%	0,00%
Total	**234728**	**236524**	**5078**	**3282**	**98.60%**	**2.16%**

After looking attacks by attacks performance, we show the results in a group in Table 43.

Table 43 Snort header rules combined with our rules performance with DARPA 1998 data set

		Snort header ruleset performance + Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**169756**	169169	581	1168	99,31%	0,34%
U2R	**297**	320	45	22	92,59%	15,15%
R2L	**2741**	1981	15	775	71,72%	0,54%
Probes	**61934**	65054	4437	1317	97,87%	7,16%

The results show that DOS, R2L and Probes attacks detection rates are increased a little. But in U2R attacks, there exists many increases. Because our rules detect much this types of attacks. Besides the false alarm rate of U2R is increased. The test takes 736 second. The time detection performance is nearly same with Snort headers but it is better than the Snort default rules 1025.1 seconds. The speed is increased by 28.20%.

Then we test Snort header rules performance with our constructed data set. The test results are shown in Table 44. Snort header rule set detection overall performance is decreased from 100% to 73.62% compared with Snort default rule set.

Table 44 Snort header rules performance with our data set

	# of attacks	Snort header ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	-	-	-	-	-	-
dict	7	0	0	0	0,00%	0,00%
eject	-	-	-	-	-	-
ffb	-	-	-	-	-	-
format	-	-	-	-	-	-
Ftp-write	-	-	-	-	-	-
guest	-	-	-	-	-	-
imap	-	-	-	-	-	-
ipsweep	452	213	0	0	47,12%	0,00%
land	84	84	0	0	100,00%	0,00%
loadmodule	-	-	-	-	-	-
neptune	160	160	0	0	100,00%	0,00%
nmap	199	199	0	0	100,00%	0,00%
perlmagic	-	-	-	-	-	-
phf	-	-	-	-	-	-
pod	272	272	0	0	100,00%	0,00%
portsweep	96	55	0	0	57,29%	0,00%
rootkit	-	-	-	-	-	-
satan	264	106	0	0	40,15%	0,00%
spy	-	-	-	-	-	-
syslog	-	-	-	-	-	-
teardrop	153	153	0	0	100,00%	0,00%
warez	-	-	-	-	-	-
warezclient	-	-	-	-	-	-
warezmaster	-	-	-	-	-	-
Total	**1687**	**1242**	**0**	**0**	**73,62%**	**0,00%**

After looking attacks by attacks performance, we show the results in a group in Table 45.

Table 45 Snort header rules performance with our data set

	# of attacks	Snort header ruleset performance				
		detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	669	669	0	0	100,00%	0,00%
U2R	-	-	-	-	-	-
R2L	7	0	0	0	0,00%	0,00%
Probes	1011	573	0	0	56,67%	0,00%

The results show that DOS detection rate remain 100%. But R2L and Probes attacks detection rates are decreased compared with Snort

default rule set performance. However the time detection performance is increased. While the Snort default rules takes 5.03 seconds, the Snort header takes 2.88 seconds. Because of rule reduction some attacks are missed.

After that Snort header rules and our rules are combined and tested with our constructed data set again. The results show in Table 46. The detection rate is increased compared with Snort header rules. Because some missed attacks are detected with our rules.

Table 46 Snort header rules combined with our rules performance with our data set

		Snort header ruleset performance + Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
back	-	-	-	-	-	-
dict	**7**	7	0	0	100,00%	0,00%
eject	-	-	-	-	-	-
ffb	-	-	-	-	-	-
format	-	-	-	-	-	-
Ftp-write	-	-	-	-	-	-
guest	-	-	-	-	-	-
imap	-	-	-	-	-	-
ipsweep	**452**	454	2	0	100,00%	0.44%
land	**84**	84	0	0	100,00%	0,00%
loadmodule	-	-	-	-	-	-
neptune	**160**	160	0	0	100,00%	0,00%
nmap	**199**	199	0	0	100,00%	0,00%
perlmagic	-	-	-	-	-	-
phf	-	-	-	-	-	-
pod	**272**	272	0	0	100,00%	0,00%
portsweep	**96**	97	1	0	100,00%	1.03%
rootkit	-	-	-	-	-	-
satan	**264**	265	1	0	99.62%	0,37%
spy	-	-	-	-	-	-
syslog	-	-	-	-	-	-
teardrop	**153**	153	0	0	100,00%	0,00%
warez	-	-	-	-	-	-
warezclient	-	-	-	-	-	-
warezmaster	-	-	-	-	-	-
Total	**1687**	**1691**	**4**	**0**	**100,00%**	**0.23%**

After looking attacks by attacks performance, we show the results in a group in Table 45.

Table 47 Snort header rules combined with our rules performance with our data set

		Snort header ruleset performance + Our ruleset performance				
	# of attacks	detected	false positive	false negative	detection rate (%)	false alarm rate (%)
DOS	**669**	669	0	0	100,00%	0,00%
U2R	-	-	-	-	-	-
R2L	**7**	7	0	0	100,00%	0,00%
Probes	**1011**	1015	4	0	100,00%	0.39%

The results show that all types of attacks are detected. But combined ruluset gives some false alarms because of our rules performance only. However the time detection performance is increased. The test takes 2.95 second. The time detection performance is nearly same with Snort headers but it is better than the Snort default rules 5.03 seconds. The speed is increased by 41.35%.

CHAPTER 5

CONCLUSION

Our results in this thesis show that faster intrusion detection systems can be designed with signatures that do not depend on packet payloads. We observe that a significant amount of intrusions can be detected by only inspecting the packet headers. However, we do not propose our approach as the only intrusion detection facility for a given system. Rather we see it as a fast detection pre-scanning tool to eliminate the intrusions that can be detected without payload inspection.

Our future work includes developing an automatic signature extraction method rather than manually processing the traffic traces. In addition, time and state based approaches can be incorporated. It is also possible to apply this method to detection tools other than Snort.

REFERENCES

[1] T. Chen, Z. Fu, L. He and T. Strayer "Recent Development in Network Intrusion Detection" IEEE Network January/February, 2009

[2] A. Lazarevic, V. Kumar and J. Srivastava "Intrusion Detection: A Survey" Massive Computing, Volume 5, Part I, 19-78, 2005

[3] http://www.cert.org/present/internet-security-trends, 2011

[4] M. Sabhnani and G. Serpen "KDD Feature Set Complaint Heuristic Rules for R2L Attack Detection" Proceedings of the International Conference on Security and Management, SAM '03, Las Vegas, Nevada, USA, Volume 1 2003 pages 310-316, June 23 – 26, 2003

[5] F. Godinez, D. Hutter and R. Monroy "On the Use of Word Networks to Mimicry Attack Detection" Emerging Trends in Information and Communication Security, International Conference, ETRICS 2006, Freiburg, Germany, Proceedings 2006 pages 423-435, June 6-9, 2006

[6] S. Chebrolu, A. Abraham, J. P. Thomas "Feature deduction and ensemble design of intrusion detection systems" Computer & Security, 2005

[7] W. H. Chen, S. H. Hsu, H. P. Shen "Application of SVM and ANN for intrusion detection" Computers & Operations Research, 2005

[8] S. T. Sarasamma, Q. A. Zhu, J. Huff "Hierarchical Kohonen Net for Anomaly Detection in Network Security" IEEE Transactions On Systems, Man, And Cybernetics, 2005

[9] F. Yu, Z. Chen, Y. Diao "Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection" ANCS '06 Proceedings of the 2006 ACM/IEEE Symposium On Architecture For Networking And Communications Systems, 2006

[10] O. Depren, M. Topallar, E. Anarim, M. K. Ciliz "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks" Expert Systems with Applications, 2005

[11] T. Shon, J. Moon "A hybrid machine learning approach to network anomaly detection" Information Sciences, 2007

[12] S. Peddabachigaria, A. Abrahamb, C. Grosanc, J. Thomas "Modeling intrusion detection system using hybrid intelligent systems" Journal of Network and Computer Applications, 2007

[13] S. J. Horng, M. Y. Su, Y.H. Chen, T. W. Kao, R. J. Chen, J. L. Lai, C. D. Perkasa "A novel intrusion detection system based on hierarchical clustering and support vector machines" Expert Systems with Applications, 2011

[14] L. Khan, M. Awad, B. Thuraisingham "A new intrusion detection system using support vector machines and hierarchical clustering" The VLDB Journal, 2007

[15] Z. Yu, J. Tsai, T. Weigert "An Automatically Tuning Intrusion Detection System" IEEE Transactions On Systems, Man, And Cybernetıcs—Part B: Cybernetics, 2007

[16] K. Shafi, H. A. Abbass "An adaptive genetic-based signature learning system for intrusion detection" Expert Systems with Applications, 2009

[17] G. P. Spathoulas, S. K. Katsikas "Reducing false positives in intrusion detection systems" Computers & Security, 2010

[18] S. Kim and H. Lee "Reducing Payload Scans for Attack Signature Matching Using Rule Classification" ACISP 2008, LNCS 5107, pages 350-360, 2008

[19] http://en.wikipedia.org/wiki/Snort_%28software%29#cite_note-1, 2011

[20] http://www.esecurityplanet.com/trends/article.php/3681296/Snort-Open-Source-Network-Intrusion-Prevention.htm, 2011

[21] http://sourcefire.mktoweb.com/GartnerMaqicQuadrant.html, 2011

[22] www.snort.org, 2011

[23] www.snort.org/snort, 2011

[24] W. Wang, R. Battiti "Identifying Intrusions in Computer Networks with Principal Component Analysis" Availability, Reliability and Security, 2006

[25] http://base.secureideas.net, 2011

[26] http://www.snort.org/start/requirements, 2011

[27] B. Caswell, J. Beale "Snort Intrusion Detection and Prevention Toolkit", Syngress Publishing, 750 pages, 2007

[28] http://en.wikipedia.org/wiki/Wireshark, 2011

[29]http://www.infoworld.com/d/open-source/greatest-open-source-software-all-time-776?page=0,2&source=fssr, 2011

[30] Kristopher Kendall "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems" DISCEX, 1999

[31] http://en.wikipedia.org/wiki/Ping_of_death, 2011

[32]http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attacks.html, 2011

[33] http://en.wikipedia.org/wiki/Rootkit, 2011

[34] http://www.backtrack-linux.org, 2011

Appendix A: Rule set

```
# This file created by Mehmet Cem TARIM

alert tcp any any -> any 80 ( msg:"back"; sid:2000001; dsize:>1400;
flags:PA,12; ttl:64; tos:0; fragbits:D+; )

alert tcp any 20,23,110,143,513 -> any 1000: ( msg:"dict and guest";
sid:2000002; dsize:17<>20; flags:PA,12; detection_filter: track by_dst,
count 5, seconds 60; )

alert tcp any any -> any 23 (msg:"eject"; sid:2000003; dsize:58;
flags:PA,12; ttl:64; tos:16; window:32120; fragbits:D+; )

alert tcp any any -> any any ( msg:"ffb and format"; sid:2000004;
dsize:71; flags:PA,12; ttl:254; tos:0; window:8760; fragbits:D+; )

alert tcp any 20:21 -> any 1000: ( msg:"ftp-write-1"; sid:2000005;
flags:S,12; ttl:254; tos:0; window:24820; fragbits:D+; )

alert tcp any 1000:1100 -> any 20,21,513 ( msg:"ftp-write-2";
sid:2000006; flags:S,12; ttl:63; tos:0; window:512; fragbits:D+; )

alert tcp any any -> any 23 ( msg:"guest"; sid:2000007; dsize:26<>39;
flags:PA,12; )

alert tcp any any -> any 143 ( msg:"imap"; sid:2000008; dsize:>1000;
flags:PA,12; window:32120; fragbits:D+; )

alert tcp any any -> any !80,!25,:100 ( msg:"ip-sweep-1"; sid:2000009;
flags:S,12; ttl:64; window:512; detection_filter: track by_src, count 3,
seconds 60; )

alert icmp any any -> any any ( msg:"ipsweep-2"; sid:2000010;
dsize:<30; itype:8; detection_filter: track by_src, count 3, seconds 60;)
```

alert ip any any -> any any (msg: "land"; sid:2000011; sameip;)

alert tcp any any -> any 23 (msg:"loadmodule"; sid:2000012; dsize:40<>50; flags:PA,12; ttl:64; window:32696; fragbits:D+;)

alert tcp any any -> any any (msg:"neptune"; sid:2000013; dsize:<40; flags:S,12; ttl:255; window:242; detection_filter: track by_src, count 3, seconds 1;)

alert tcp any any -> any any (msg:"nmap-1"; sid:2000014; dsize:<40; ttl:254; flags:F,12; window:2048;)

alert udp any 137,138 -> any 137,138 (msg:"nmap-2"; sid:2000015; dsize:<250; ttl:63;)

alert icmp any any -> any any (msg:"nmap-3"; sid:2000016; ttl:63; itype:8;)

alert tcp any any -> any 23 (msg:"perlmagic"; sid:2000017; dsize:19; flags:PA,12; ttl:64; window:32120; fragbits:D+;)

alert tcp any any -> any 80 (msg:"phf"; sid:2000018; dsize:50<>60; flags:PA,12; ttl:64; window:32120; fragbits:D+;)

alert ip any any -> any any (msg:"pod"; sid:2000019; dsize:1480;)

alert tcp any 1000: -> any !80 (msg:"port-sweep"; sid:2000020; dsize:<40; flags:S,12; detection_filter: track by_src, count 10, seconds 1;)

alert icmp any any -> any any (msg:"rootkit-1"; sid:2000021; dsize:50<>100; ttl:63; itype:8;)

alert udp any 1000: -> any 1000: (msg:"rootkit-2"; sid:2000022; dsize:4<>40;)

alert icmp any any -> any any (msg:"satan-1"; sid:2000023; dsize:<20;)

alert tcp any 1000: -> any :10000,!80 (msg:"satan-2"; sid:2000024; dsize:<40; flags:S,12; ttl:64; detection_filter: track by_src, count 5, seconds 1;)

alert udp any !53 -> any !53 (msg:"satan-3"; sid:2000025; dsize:<25; ttl:64; tos:0;)

alert tcp any 10000: -> any 23 (msg:"spy"; sid:2000026; dsize:45<>50; flags:PA,12; ttl:64; tos:16; window:32120;)

alert udp any 514 -> any 514 (msg:"syslog"; sid:2000027; dsize:<10; ttl:64;)

alert udp any any -> any any (msg:"teardrop"; sid:2000028; dsize:28; id:242; fragbits:M;)

alert tcp any any -> any 21 (msg:"warez"; sid:2000029; dsize:>30; ttl:64; flags:PA,12; tos:0; window:32120;)

alert tcp any 20 -> any 1000: (msg:"warezclient"; sid:2000030; dsize:>300; flags:PA,12; ttl:254; window:24820; detection_filter: track by_dst, count 5, seconds 1;)

alert tcp any 1000: -> any 21 (msg:"warezclient"; sid:2000031; dsize:<20; flags:S,12; ttl:64; window:512;)

alert tcp any 20 -> any 1000: (msg:"warezmaster"; sid:2000032; dsize:<40; ttl:254; flags:S,12; window:24820;)

Appendix B: Modified Snort config file

```
#--------------------------------------------------
#   http://www.snort.org     Snort 2.8.5.2 Rule set
#   Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
var HOME_NET $wlan0_ADDRESS
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var FTP_SERVERS $HOME_NET
var SNMP_SERVERS $HOME_NET
portvar HTTP_PORTS 80
portvar SHELLCODE_PORTS !80
portvar ORACLE_PORTS 1521
portvar FTP_PORTS 21
var AIM_SERVERS
[64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,64.12.163.0/24,64.12.20
0.0/24,205.188.3.0/24,205.188.5.0/24,205.188.7.0/24,205.188.9.0/24,
205.188.153.0/24,205.188.179.0/24,205.188.248.0/24]
```

```
var RULE_PATH /etc/snort/rules

var PREPROC_RULE_PATH /etc/snort/preproc_rules

preprocessor stream5_global: max_tcp 8192, track_tcp yes, \

                        track_udp no

output unified2: filename snort.log, limit 128

include $RULE_PATH/local.rules
```

Appendix C: Information about Snort

Snort modes of work

Snort can be configured in three main working modes: [22]

• Sniffer mode, which reads network packets and displays them on the console (screen).

• Packet Logger mode, which logs the packets to the disk.

• Network Intrusion Detection System (NIDS) mode, Snort monitors network traffic and analyzes it for matches against a user defined rule set and performs several actions.

Understanding standard alert output

When Snort generates an alert message, it will usually look like the following [22]:

[**] [116:56:1] (snort_decoder): T/TCP Detected [**]

The first number is the Generator ID, this tells the user what component of Snort generated this alert. In this example, we know that this event came from the "decode" (116) component of Snort [22].

The second number is the Snort ID (sometimes referred to as Signature ID). Rule-based SIDs is written directly into the rules with the sid option. In this case, 56 represent a T/TCP event [22].

The third number is the revision ID. This number is primarily used when writing signatures, as each rendition of the rule should increment this number with the rev option [22].

High performance configuration

To increase snort performance unified logging and a unified log reader such as barnyard can be used. This allows Snort to log alerts in a binary form as fast as possible while another program performs the slow actions, such as writing to a database.

Reading Pcaps

Instead of having Snort listen on an interface, we can give it a packet capture to read. Snort will read and analyze the packets as if they came off the wire. This can be useful for testing and debugging Snort [22].

snort –r outside.tcpdump

Running Snort as a Daemon

To run Snort as a daemon, we can the add -D switch to any combination described in the previous sections. For example:

snort –r outside.tcpdump -c /etc/snort.conf -D

Configuring snort

Includes: The include keyword allows other snort config files to be included within the snort.conf indicated on the Snort command line. It works much like an #include from the C programming language, reading the contents of the named file and adding the contents in the place where they include statement appears in the file.

Format:

include <include file path/name>

Preprocessors: Preprocessors allow the functionality of Snort to be extended by allowing users and programmers to drop modular plugins into Snort fairly easily. Preprocessor code is run before the detection engine is called, but after the packet has been decoded. The packet can be modified or analyzed in an out-of-band manner using this mechanism.

Preprocessors are loaded and configured using the preprocessor keyword. The format of the preprocessor directive in the Snort config file is:

preprocessor <name>: <options>

Snort preprocessros are frag3, sfportscan, rpc decode, performance monitor, http inspect, smtp preprocessor, ftp/telnet preprocessor, ssh, dns, ssl/tls, arp spoof preprocessor, dce/rpc 2 preprocessor, sensitive data preprocessor, normalizer.

Output modules

Output modules allow Snort to more flexible in the formatting and presentation of output to its users. The output modules are run when the alert or logging subsystems of Snort are called, after the preprocessors and detection engine. The format of the directives in the config file is very similar to that of the preprocessors [22].

Multiple output plugins may be specified in the Snort configuration file. When multiple plugins of the same type (log, alert) are specified, they are stacked and called in sequence when an event occurs. As with the standard logging and alerting systems, output plugins send their data to /var/log/snort by default or to a user directed directory (using the -l command line switch). Output modules are loaded at runtime by specifying the output keyword in the config file:

output <name>: <options>

output alert_syslog: log_auth log_alert

Writing Snort rules

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule’s action, protocol, source and destination IP addresses and net masks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

Rules headers: The rule header contains the information that defines the who, where, and what of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the rule action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 available default actions in Snort, alert, log, pass, activate, and dynamic.

- **Alert:** Generate an alert using the selected alert method, and then log the packet
- **Log:** Log the packet
- **Pass:** Ignore the packet
- **Activate:** Alert and then turn on another dynamic rule
- **Dynamic:** Remain idle until activated by an activate rule , then act as a log rule

Protocols: The next field in a rule is the protocol. There are four protocols that Snort currently analyzes for suspicious behavior TCP, UDP, ICMP, and IP.

IP addresses: The next portion of the rule header deals with the IP addresses and port information for a given rule. The keyword any may be used to define any address. Snort does not have a mechanism to provide host name lookup for the IP address fields in the config file. The addresses are formed by a straight numeric IP address and a CIDR block. The CIDR block indicates the net mask that should be applied to the rule's address and any incoming packets that are tested against the rule. A CIDR block mask of /24 indicates a Class C network, /16 a Class B network, and /32 indicates a specific machine address. For example, the address/CIDR combination 192.168.1.0/24 would signify the block of addresses from 192.168.1.1 to 192.168.1.255. Any rule that used this designation for, say, the destination address would match on any address in that range [22].

Port numbers: Port numbers may be specified in a number of ways, including any ports, static port definitions, ranges, and by negation. Any

ports are a wildcard value, meaning literally any port. Static ports are indicated by a single port number, such as 111 for port mapper, 23 for telnet, or 80 for http, etc. Port ranges are indicated with the range operator: The range operator may be applied in a number of ways to take on different meanings [22].

log udp any any -> 192.168.1.0/24 1:1024 log udp

log udp traffic coming from any port and destination ports ranging from 1 to 1024

log tcp any any -> 192.168.1.0/24 :6000

log tcp traffic from any port going to ports less than or equal to 6000

log tcp any :1024 -> 192.168.1.0/24 500:

log tcp traffic from privileged ports less than or equal to 1024 going to ports greater than or equal to 500

Port negation is indicated by using the negation operator !. The negation operator may be applied against any of the other rule types (except any, which would translate to none). The example of the port negation is shown below [22].

log tcp any any -> 192.168.1.0/24 !6000:6010

The direction operator: The direction operator -> indicates the orientation, or direction, of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source host, and the address and port information on the right side of the operator is the destination host. There is also a bidirectional operator, which is indicated with a <> symbol. This tells Snort to consider the address/port pairs in either the source or destination orientation. This is handy for recording/analyzing both sides of a conversation, such as telnet or POP3 sessions. An example of the bidirectional operator being used to record both sides of a telnet session is shown below [22].

log tcp !192.168.1.0/24 any <> 192.168.1.0/24 23

Rule Options

Rule options form the heart of Snort's intrusion detection engine, combining ease of use with power and flexibility. All Snort rule options are separated from each other using the semicolon (;) character. Rule option keywords are separated from their arguments with a colon (:) character.

activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA; \

content: "|E8C0FFFFFF|/bin"; activates: 1; \

msg: "IMAP buffer overflow!";)

dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by: 1; count: 50;)

There are four major categories of rule options.

- **General:** These options provide information about the rule but do not have any affect during detection
- **Payload:** These options all look for data inside the packet payload and can be inter-related
- **Non-payload:** These options look for non-payload data
- **Post-detection:** These options are rule specific triggers that happen after a rule has "fired."

Table 48 General rule options

Keyword	Description
Msg	The msg keyword tells the logging and alerting engine the message to print with the packet dump or alert.
reference	The reference keyword allows rules to include references to external attack identification systems.

Gid	The gid keyword (generator id) is used to identify what part of Snort generates the event when a particular rule fires.
Sid	The sid keyword is used to uniquely identify Snort rules.
Rev	The rev keyword is used to uniquely identify revisions of Snort rules.
classtype	The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class.
Priority	The priority keyword assigns a severity level to rules.
metadata	The metadata keyword allows a rule writer to embed additional information about the rule, typically in a key-value format.

Table 49 Payload detection rule options

Keyword	Description
Content	The content keyword allows the user to set rules that search for specific content in the packet payload and trigger response based on that data.
Rawbytes	The rawbytes keyword allows rules to look at the raw packet data, ignoring any decoding that was done by preprocessors
Depth	The depth keyword allows the rule writer to specify how far into a packet Snort should search for the specified pattern.
Offset	The offset keyword allows the rule writer to specify where

	to start searching for a pattern within a packet.
Distance	The distance keyword allows the rule writer to specify how far into a packet Snort should ignore before starting to search for the specified pattern relative to the end of the previous pattern match.
Within	The within keyword is a content modifier that makes sure that at most N bytes are between pattern matches using the content keyword.
uricontent	The uricontent keyword in the Snort rule language searches the normalized request URI field.
Isdataat	The isdataat keyword verifies that the payload has data at a specified location.
Pcre	The pcre keyword allows rules to be written using perl compatible regular expressions.
byte test	The byte test keyword tests a byte field against a specific value (with operator).
byte jump	The byte jump keyword allows rules to read the length of a portion of data, then skip that far forward in the packet.
ftpbounce	The ftpbounce keyword detects FTP bounce attacks.
asn1	The asn1 detection plugin decodes a packet or a portion of a packet, and looks for various malicious encodings.
Cvs	The cvs keyword detects invalid entry strings.

Table 50 Non-Payload detection rule options

Keyword	Description
fragoffset	The fragoffset keyword allows one to compare the IP fragment offset field against a decimal value.
Ttl	The ttl keyword is used to check the IP time-to-live value.
Tos	The tos keyword is used to check the IP TOS field for a specific value.
Id	The id keyword is used to check the IP ID field for a specific value.
Ipopts	The ipopts keyword is used to check if a specific IP option is present.
Fragbits	The fragbits keyword is used to check if fragmentation and reserved bits are set in the IP header.
Dsize	The dsize keyword is used to test the packet payload size.
Flags	The flags keyword is used to check if specific TCP flag bits are present.
Flow	The flow keyword allows rules to only apply to certain directions of the traffic flow.
Flowbits	The flowbits keyword allows rules to track states during a transport protocol session.
Seq	The seq keyword is used to check for a specific TCP sequence number.
Ack	The ack keyword is used to check for a specific TCP acknowledge number.

Window	The window keyword is used to check for a specific TCP window size.
Itype	The itype keyword is used to check for a specific ICMP type value.
Icode	The icode keyword is used to check for a specific ICMP code value.
icmp_id	The icmp id keyword is used to check for a specific ICMP ID value.
icmp_seq	The icmp seq keyword is used to check for a specific ICMP sequence value.
Rpc	The rpc keyword is used to check for a RPC application, version, and procedure numbers in SUNRPC CALL requests.
ip_proto	The ip proto keyword allows checks against the IP protocol header.
Sameip	The sameip keyword allows rules to check if the source ip is the same as the destination IP.

Table 51 Post-Detection rule options

Keyword	Description
Logto	The logto keyword tells Snort to log all packets that trigger this rule to a special output log file.
Session	The logto keyword tells Snort to log all packets that trigger this rule to a special output log file.
Resp	The logto keyword tells Snort to log all packets that

	trigger this rule to a special output log file.
React	This keyword implements an ability for users to react to traffic that matches a Snort rule by closing connection and sending a notice.
Tag	The tag keyword allow rules to log more than just the single packet that triggered the rule.
Activates	This keyword allows the rule writer to specify a rule to add when a specific network event occurs.
activated_by	This keyword allows the rule writer to dynamically enable a rule when a specific activate rule is triggered.
Count	This keyword must be used in combination with the activated by keyword. It allows the rule writer to specify how many packets to leave the rule enabled for after it is activated.
Replace	Replace the prior matching content with the given string of the same length. Available in inline mode only.
detection_filter	Track by source or destination IP address and if the rule otherwise matches more than configured rate it will fire.

Appendix D: Used command to generate Our data set

dict attack

./brutessh.py -h 192.168.1.143 -u root -d list.txt

land attack

hping3 -S -a 192.168.1.143 -p 21 192.168.1.143

neptune attack

hping3 -S 192.168.1.143 -t 255 -w 242

pod attack

hping3 192.168.1.143 -1 -d 1480

teardrop attack

hping3 192.168.1.143 -x -d 28 -N 242 -2

ipsweep attack

hping3 192.168.1.143 -w 512 -S -p ++0

hping3 192.168.1.143 -1 -C 8

nmap attack

hping3 192.168.1.143 -1 -t 63 -C 8

hping3 192.168.1.143 -d 200 -2 -p 138 -s 138 -t 63

hping3 192.168.1.143 -d 200 -2 -p 137 -s 137 -t 63

hping3 192.168.1.143 -8 1-100 -F

hping3 192.168.1.143 -d 10 -t 254 -F -w 2048

portsweep attack

hping3 192.168.1.143 -d 5 -s 10000 -p ++0 –S

satan attack

hping3 192.168.1.143 -d 5 -s 2000 -p ++0 -S

hping3 192.168.1.143 -d 3 -2

hping3 192.168.1.143 -1

Appendix E: Used command to extract header rules of Snort rule set

```
cat bad-traffic.rules > snort_all

cat exploit.rules >> snort_all

cat community-exploit.rules >> snort_all

cat scan.rules >> snort_all

cat finger.rules >> snort_all

cat ftp.rules >> snort_all

cat telnet.rules >> snort_all

cat rpc.rules >> snort_all

cat rservices.rules >> snort_all

cat dos.rules >> snort_all

cat community-dos.rules >> snort_all

cat ddos.rules >> snort_all

cat dns.rules >> snort_all

cat tftp.rules >> snort_all

cat web-cgi.rules >> snort_all

cat web-coldfusion.rules >> snort_all

cat web-iis.rules >> snort_all

cat web-frontpage.rules >> snort_all

cat web-misc.rules >> snort_all
```

```
cat web-client.rules >> snort_all

cat web-php.rules >> snort_all

cat community-sql-injection.rules >> snort_all

cat community-web-client.rules >> snort_all

cat community-web-dos.rules >> snort_all

cat community-web-iis.rules >> snort_all

cat community-web-misc.rules >> snort_all

cat community-web-php.rules >> snort_all

cat sql.rules >> snort_all

cat x11.rules >> snort_all

cat icmp.rules >> snort_all

cat netbios.rules >> snort_all

cat misc.rules >> snort_all

cat attack-responses.rules >> snort_all

cat oracle.rules >> snort_all

cat community-oracle.rules >> snort_all

cat mysql.rules >> snort_all

cat snmp.rules >> snort_all

cat community-ftp.rules >> snort_all

cat smtp.rules >> snort_all

cat community-smtp.rules >> snort_all

cat imap.rules >> snort_all

cat community-imap.rules >> snort_all
```

```
cat pop2.rules >> snort_all

cat pop3.rules >> snort_all

cat nntp.rules >> snort_all

cat community-nntp.rules >> snort_all

cat community-sip.rules >> snort_all

cat other-ids.rules >> snort_all

cat web-attacks.rules >> snort_all

cat backdoor.rules >> snort_all

cat community-bot.rules >> snort_all

cat community-virus.rules >> snort_all

cat experimental.rules >> snort_all

cat snort_all | grep -v '#' > snort_all_nocomment

cat snort_all_nocomment | grep -v content: > snort_header_nocontent

cat snort_header_nocontent | grep -v isdataat > snort_header_noisdataat

cat snort_header_noisdataat | grep -v pcre > snort_header_nopcre

cat snort_header_nopcre | grep -v byte* > snort_header_nobyte

cp snort_header_nobyte snort-header.rules
```