Detecting network intrusions

Michael Nørholt Petersen

Kongens Lyngby 2014
M.Sc.

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk M.Sc.-2014

Acknowledgements

I would like to thank Associate Professor Christian W. Probst for his great supervision. Always having time for me, and checking the quality of my work.

Contents

Chapter 1

Introduction

This thesis deals with Intrusion Detection Systems. It is called "Detecting network intrusions", and is done at DTU Compute at the Technical University of Denmark in fulfilment of the requirements for acquiring M.Sc. degree in Computer Science and Engineering. The thesis has been supervised by Christian W. Probst.

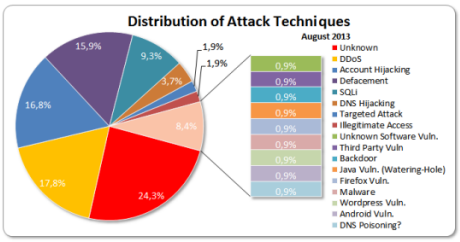
We wish to look at Intrusion Detection Systems from a critical perspective: Can they really help protect against intrusions as promised? Is it possible to make a trustworthy investigation of an Intrusion Detection System, finding its limitations? Is it possible to make an trustworthy best practice for testing Intrusion Detection System?

1.1 Motivation

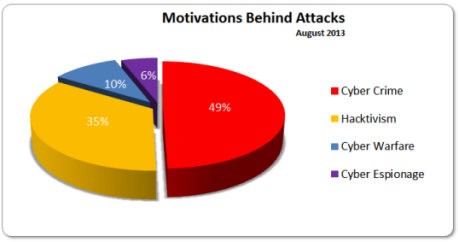
The basic definition of intrusion is that there can be performed a set of actions which compromise the security goals, namely integrity, confidentiality or availability of a computing and networking resource Hachmageddon.com [1] and PWC [2]. Intrusion Detection is the process of identifying and responding to intrusion activities. The internet has evolved rapidly and almost everyone has access. As the use of the Internet rapidly grows, so does the possibility of an

attack. As private users we use the Internet for backup, banking, and other sensitive information handling. Businesses also handle a lot of sensitive information within their internal networks. In most cases a firewall has shown not be sufficient, as absolutely secure systems are unobtainable.
Another issue is threats from insiders abusing their privileges. Also outdated IDS database, and the issue that not all kind of intrusions are known, is a threat. We have included figures to give an impression on how widespread attacks currently are. They are taken from August 2013 and show attacks distributed according to country, targets, motivation and techniques.

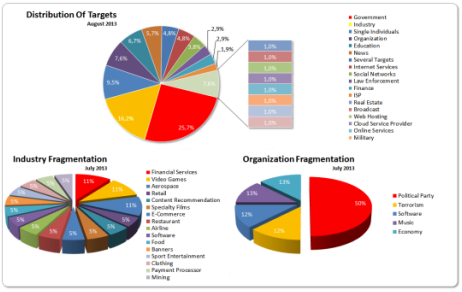
Figure 1.1: Charts from Hachmageddon.com [1]



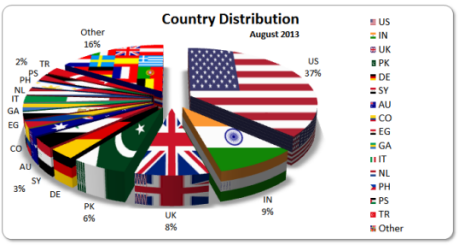
(a) Attack distribution



(b) Motivations



(c) Targets

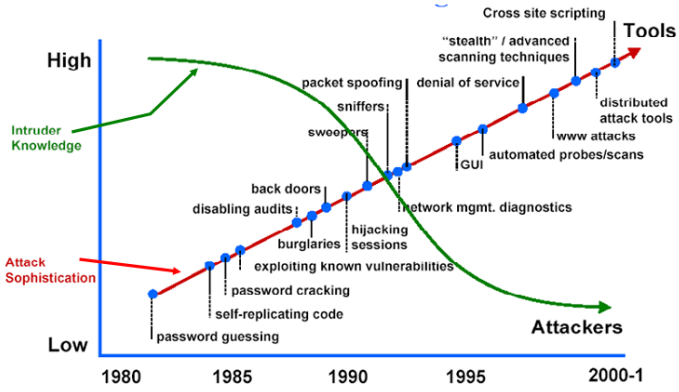


(d) Distribution of countries

Figure 1.1a: We can see that "unknown" attack technique is the one which has the greatest share. Its especially not good for rule based ids that this technique is evolving, because it has big trouble defending against such attacks.
Figure 1.1b: Cyber Crime leads the Motivation Behind Attacks chart with approximately half of the attacks recorded. Hacktivism is stable at 35% while the growth of Cyber Warfare is related (once again) to the cyber skirmishes between India and Pakistan.
Figure 1.1c: Governmental targets lead the Distribution of Target chart with nearly 26%. Industry ranks at number two, while single individuals (victims essentially of account hijackings) rank at number three. It is interestig to notice,

among the organizations victims of Cyber Attacks, the predominance of targets related to Political Parties, a consequence of the social protests exploding all over the world in these troubled days.
Figure 1.1d: US, UK and India confirm their top rank in the Country Distribution chart.

Figure 1.2: Attack sophistication vs. Intruder technical knowledge



Earlier, the intruders needed profound understanding of computers and networks to launch attacks. However, today almost anyone can exploit the vulnerabilities in a computer system due to the wide availability of attack tools (see figure 1.2).

Figure 1.3: Security challenges taken from IBM Corporation [3]

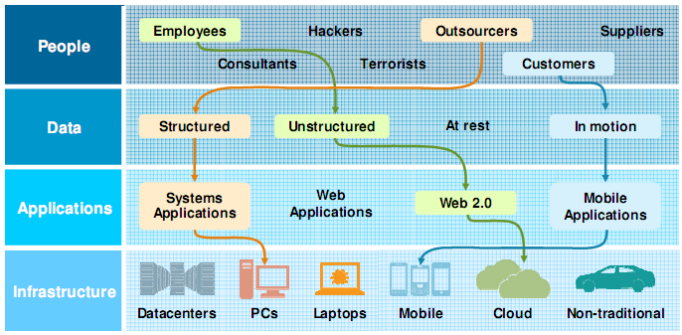


Figure 1.3 shows a four-dimensional puzzle IBM Corporation [3]. It is added to show the different involved actors regarding security challenges.

1.2 Scope

To begin with, the thesis referred to general IDS, mainly open-source programs. By reviewing the newest articles in the field of IDS, we wanted to give a state of the art overview, where we showed the different techniques IDS use, how the patterns are represented and detected. Besides that we wanted to select an unspecified number of IDS programs, and investigate for its limitations, pros and cons, and lastly having a look at how they complemented each other. After some iterations it showed, that is was not possible, so the thesis had to follow a new direction. The new direction would contain; a state of the art overview, where we show the different techniques (IDS) use, how the patterns are represented and detected. Lastly we would give a best practice regarding testing of IDS.
The things which are outside the scope of this thesis, are for instance use of other environments such as neural network, wireless network and cloud computing. Looking at figure 2.21 there exist many combinations which are outside the scope.

1.3 Objective and research question

The main task of this thesis is to give an state of the art IDS overview, including explanation of recent approaches in the field of general IDS. Lastly give a best practice regarding testing of IDS. The research questions that will be answered during this thesis will be:

"Is it possible to make an trustworthy investigation of an Intrusion Detection System which finds its limitations?"

"Is it possible to make an trustworthy best practice for testing Intrusion Detection System?"

The sub questions that can be derived from the research questions are:

1. What is an IDS? - what is the typical architecture?
2. What sort of techniques does the IDS use, how is the patterns represented and detected?
3. What is the common test approach for IDS?
4. Does an IDS cover all potential intrusions?

5. What is the future prospects of IDS?

Reading this thesis will hopefully give an answer to these sub questions.

1.4 Methodology

This thesis is split into different approaches, and they will be explained in the next subsections.

1.4.1 Literature

The existing literature in the field of IDS will be cited as a basis for answering the main research question and the listed sub questions. There exist many different scientific articles describing IDS, and an effort will be made to select the most important, relevant and newest ones. Although the scientific articles is the key distribution for this thesis, there also exist some relevant home pages and slides. The places where the relevant articles will be found are Google search engine and DTU Digital Library.

1.4.2 Procedure

To begin with we wanted to do the following:

At first we will use the KDD Cup 1999 Dataset to find the limitations of an selected IDS. Besides that we will try to cover the following performance objectives for our IDS:

- **Broad Detection Range:** for each intrusion in a broad range of known intrusions, the IDS should be able to distinguish the intrusion from normal behaviour.
- **Economy in Resource Usage:** the IDS should function without using too much system resources such as main memory, CPU time, and disk space.
- **Resilience to Stress:** the IDS should still function correctly under stressful conditions in the system, such as a very high level of computing activity.

The way we will cover them, is to make testing scenarios. To support the testing scenarios which cover **Broad Detection Range** we will use statistical calculations, which base our performance tests on Accuracy, Sensitivity, Specificity and computational time FAR.

This approach were not possible to complete, so we decided to use Snort as an example for an IDS. In addition we chose to make a test of Snort with pytbull (is a python based flexible IDS/IPS testing framework), and based on the used articles in this thesis, we will give a best practice when testing an IDS.

We will use an iterative approach, where it is possible to re-evaluate the procedure, and make new choices.

1.4.3 Evaluation

We will look at the used tools, explain our experience with the installation and the usage. Besides that we will make an evaluation of the testing, and list our findings.

1.5 Abbreviations and terminology

IDS Intrusion Detection System: is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station.

Data Mining Data Mining is the process of extracting patterns from data.

Machine Learning Is a branch of artificial intelligence, is about the construction and study of systems that can learn from data.

Detection Rate The detection rate is defined as the number of intrusion instances detected by the system(True Positive) divided by the total number of intrusion instances present in the test set.

Alert/Alarm A signal suggesting that a system has been or is being attacked.

False Positive is defined as total number of normal instances that were incorrectly classified as intrusions defined by the total number of normal instances.

True Positive A legitimate attack which triggers an IDS to produce an alarm.

False Negative A failure of an IDS to detect an actual attack.

True Negative When no attack has taken place and no alarm is raised.

Firewall The network security door. A firewall is not an IDS but their logs can provide valuable IDS information. A firewall works by blocking unwanted connections based on rules or criteria, such as source address, ports etc.

Honeypot A honeypot is a system that can simulate one or many vulnerable hosts, providing an easy target for the hacker to attack. The honeypot should have no other role to fulfil, therefore all connection attempts are deemed suspicious. Another purpose is delay attackers in their pursuit of legitimate targets, causing the attacker to waste time on the honeypot, whilst the original entry hole is secured, leaving the truly valuable assets alone. Although one of the initial objectives of honeypots is as evidence-gathering mechanisms in the prosecution of malicious hackers, there is much talk of entrapment when deploying honeypots; however, does the vulnerability of the honeypot necessarily give the hacker the right to attack it? In order to reach the honeypot an attacker would have had to circumvent at least one bonafide security device, provided the honeypot is inside your network. In some countries law enforcement agencies cannot prosecute using evidence from a honeypot.

1.6 Thesis Outline

This thesis is divided into 7 chapters plus appendix.

Chapter 1 called Introduction: It gives an basic understanding of what the problem statement is and how we will come up with a solution.

Chapter 2 called Intrusion Detection System overview: It gives an impression of what the recent IDS approaches are and challenges which lies ahead.

Chapter 3 called Typical architecture of Intrusion Detection Systems: Gives a basic understanding of how the different components of the selected IDS works.

Chapter 4 called How to evaluate intrusion detection systems: It gives an understanding of how others have tested and how we will test a given IDS. Additionally, we give an evaluation of the given IDS.

Chapter 5 called Best practice: It contribute to give an advise for others who have the intention of testing an IDS.

Chapter 6 called Conclusion: A final summary of the thesis, where we discuss the role of NIDS, answer the research questions and suggest improvements.

Appendix A called Pytbull and snorby results Images from the programs pytbull and snorby.

Chapter 2

Intrusion Detection System overview

In this chapter we introduce IDS. We explain the different types, and cover the most important approaches used in relation to IDS. We also explain attacks and threats, and focus on the two interesting threats such as multi-step attack and polymorphic worm. Lastly we show a taxonomy of IDS and lastly list some of the challenges regarding installing an IDS.

The chapter contribute to an overall background understanding of general IDS. Besides that we have chosen to divide section 2.3 into theory and use of theory in IDS context. The dividing has been done, so it is easier for the reader to understand the section.

Support Vector Machine has been left out from section 2.3 because it was not possible to find relevant articles. Besides that the Honeypot section only explains advantages/disadvantages with the use of Honeypots.

2.1 Introduction

An intrusion is defined to be a violation of the security policy of the system; intrusion detection thus refers to the mechanisms that are developed to detect violations of system security policy Chebrolu et al. [4].Intrusion detection is based on the assumption that intrusive activities are noticeably different from normal system activities and thus detectable. Intrusion detection is not introduced to replace prevention-based techniques such as authentication and access control; instead, it is intended to complement existing security measures and detect actions that bypass the security monitoring and control component of the system. Intrusion detection is therefore considered as a second line of defence for computer and network systems. Some of the important features an intrusion detection system should posses include:

- Be fault tolerant and run continually with minimal human supervision. The IDS must be able to recover from system crashes, either accidental or caused by malicious activity.
- Posses the ability to resist subversion so that an attacker cannot disable or modify the IDS easily. Furthermore, the IDS must be able to detect any modifications forced on the IDS by an attacker.
- Impose minimal overhead on the system to avoid interfering with the normal operation of the system.
- Be easy to deploy: this can be achieved through portability to different architectures and operating systems, through simple installation mechanisms, and by being easy to use by the operator.
- Be general enough to detect different types of attacks and must not recognize any legitimate activity as an attack (false positives). At the same time, the IDS must not fail to recognize any real attacks (false negatives).

2.2 Types

Network-based IDS (NIDS) is an intrusion detection system which monitors network traffic Deepa et al. [5] and Stallings [6]. It use the technique like packet sniffing, and analyse the collected network data, it tries to discover unauthorized access to a computer network. A typical NIDS facility includes a number of sensors to monitor packet traffic, one or more servers for NIDS management functions, and one or more management consoles for the human interface.

The analysis of traffic patterns to detect intrusions may be done at the sensor, at the management server, or some combination of the two. Sensors can be deployed in one of two modes: inline and passive. An inline sensor is inserted into a network segment so that the traffic that is monitoring must pass through the sensor. One way to achieve an inline sensor is to combine NIDS sensor logic with another network device, such as a firewall or a LAN switch. This approach has the advantage that no additional separate hardware devices are needed; all that is required is NIDS sensor software. An alternative is a stand-alone inline NIDS sensor. The primary motivation for the use of inline sensors is to enable them to block an attack when one is detected. In this case the device is performing both intrusion detection and intrusion prevention functions. More commonly, passive sensors are used. A passive sensor monitors a copy of network traffic; the actual traffic does not pass through the device. From the point of view of traffic flow, the passive sensor is more efficient than the inline sensor, because it does not add an extra handling step that contributes to packet delay. NIDS makes use of signature detection and anomaly detection:

Signature detection The following lists examples of that types of attacks that are suitable for signature detection:

- **Application layer reconnaissance and attacks**: Most NIDS technologies analyze several dozen application protocols. Commonly analyzed ones include Dynamic Host Configuration Protocol (DHCP), DNS, Finger, FTP, HTTP, Internet Message Access Protocol (IMAP), Internet Relay Chat (IRC), Network File System (NFS), Post Office Protocol (POP), rlogin/rsh, Remote Procedure Call (RPC), Session Initiation Protocol (SIP), Server Message Block (SMB), SMTP, SNMP, Telnet, and Trivial File Transfer Protocol (TFTP), as well as database protocols, instant messaging applications, and peer-to-peer file sharing software. The NIDS is looking for attack patterns that have been identified as targeting these protocols. Examples of attack include buffer overflows, password guessing, and malware transmission.
- **Transport layer reconnaissance and attacks**: NIDSs analyze TCP and UDP traffic and perhaps other transport layer protocols. Examples of attacks are unusual packet fragmentation, scans for vulnerable ports, and TCP-specific attacks such as SYN floods.
- **Network layer reconnaissance and attacks**: NIDSs typically analyze IPv4, ICMP, and IGMP at this level. Examples of attacks are spoofed IP addresses and illegal IP header values
- **Unexpected application services**: The NIDS attempts to determine if the activity on a transport connection is consistent with

the expected application protocol. An example is a host running an unauthorized application service.

- **Policy violations**: Examples include use of inappropriate Web sites and use of forbidden application protocols.

Anomaly detection

- **Denial-of-service (DoS) attacks**: Such attacks involve either significantly increased packet traffic or significantly increase connection attempts, in an attempt to overwhelm the target system.
- **Scanning**: A scanning attack occurs when an attacker probes a target network or system by sending different kinds of packets. Using the responses received from the target, the attacker can learn many of the system's characteristics and vulnerabilities. Thus, a scanning attack acts as a target identification tool for an attacker. Scanning can be detected by atypical flow patterns at the application layer (e.g., banner grabbing3), transport layer (e.g., TCP and UDP port scanning), and network layer (e.g., ICMP scanning).
- **Worms**: Worms4 spreading among hosts can be detected in more than one way. Some worms propagate quickly and use large amounts of bandwidth. Worms can also be detected because they can cause hosts to communicate with each other that typically do not, and they can also cause hosts to use ports that they normally do not use. Many worms also perform scanning.

Host-Based IDS is an intrusion detection system that monitors and analyses the internals of a computing system as well as (in some cases) the network packets on its network interfaces (just like a (NIDS) would do).

Stack-Based IDS is an intrusion detection system that examines the packets as they go through the TCP/IP stack.

Protocol-Based IDS (PIDS) is an intrusion detection system which is typically installed on a web server, and is used in the monitoring and analysis of the protocol in use by the computing system. A PIDS will monitor the dynamic behaviour and state of the protocol and will typically consist of a system or agent that would typically sit at the front end of a server, monitoring and analysing the communication between a connected device and the system it is protecting.

Graph-Based IDS is an intrusion detection system which detects intrusions that involve connections between many hosts or nodes. A graph consists of nodes representing the domains and edges representing the network traffic between them.

2.3 Recent approaches

The section cover different technologies which has been used in an IDS. The focus is on the recent approaches, and the technologies has been categorised. Here we briefly sum up the different articles linked with each technology approach:

- **Data mining**: Zhou et al. [8] has a module which match rules. This is the detection engine, which uses K Means algorithm as the clustering analysis algorithm. When an unknown attack gets detected the log module logs it, and their feature extractor gets to work. It makes correlation analysis of the data in the log, and conclude the new association rule, and add it to the rule base. It uses Apriori algorithm correlation analysis.
- **Machine learning**: Natesan et al. [10] base their experiments on the KDDCup 99 data set. They have proposed an Adaboost algorithm with different combination of weak classifiers. The weak classifiers such as Bayes Net, Naive Bayes and Decision tree are used in three different combinations such as BN-NB, BN-DT and NB-DT with Adaboost algorithm to improve the classification accuracy.
- **Hidden Markov Models**: Ariu et al.[11] they address the problems in payload analysis by proposing a novel solution where the HTTP payload is analyzed using Hidden Markov Models. The proposed system is named HMMPayl. Farhadi et al. [12] in order to extract useful information from alerts they use an *alert correlation* algorithm, which is the process of producing a more abstract and high-level view of intrusion occurrences in the network from low-level IDS alerts.
- **Honeypot**: Bhumika [13] they list advantages and disadvantage of the use of Honeypot.
- **Genetic Algorithm**: Dhak et al. [15] propose an IDS based on the use of genetic algorithm. Their architecture is as follows: It starts from initial population generation from pfirewall.log file generated by the firewall system. The packets are the filtered out on the basis of rules. Then the precised data packets go through several steps namely selection, crossover and mutation operation. These processes gets generate best individuals. The generated individuals are the verified by the fitness function to generate the population for next generation.
- **Fuzzy logic**: Shanmugavadivu et al. [17] they propose a system which is a designed fuzzy logic-based system for effectively identifying the intrusion activities within a network. The proposed fuzzy logic-based system can be able to detect an intrusion behaviour of the networks since the rule base

contains a better set of rules. Here, they have used automated strategy for generation of fuzzy rules, which are obtained from the definite rules using frequent items. The experiments and evaluations of the proposed intrusion detection system are performed with the KDD Cup 99 intrusion detection dataset.

2.3.1 Data mining techniques

2.3.1.1 Theory

It is currently used in a wide range of profiling practices, such as marketing, surveillance, fraud detection, and scientific discovery D'silva et al. [7]. A primary reason for using data mining is to assist in the analysis of collections of observations of behaviour. Data Mining is involved in four classes of tasks:

1. **Clustering** it is the task of discovering groups and structures in the data that are in some way or another similar, without using known structures in the data. It is an unsupervised machine learning mechanism for discovering patterns in unlabelled data. It is used to label data and assign it into clusters where each cluster consists of members that are quite similar. Members from different clusters are different from each other. Hence clustering methods can be useful for classifying network data for detecting intrusions. Clustering can be applied on both Anomaly detection and Misuse detection.

   Looking closer at clustering techniques used in IDS, there exist three clustering techniques called K-Means clustering, Y-Means Clustering and Fuzzy C-Means Clustering. All these algorithms reduce the false positive rate and increase the detection rate of the intrusions.

   K-Means Clustering is a hard partitioned clustering algorithm, and It uses Euclidean distance as the similarity measure. Hard clustering means that an item in a data set can belong to one and only one cluster at a time. It is a clustering analysis algorithm that groups items based on their feature values into K disjoint clusters such that the items in the same cluster have similar attributes and those in different clusters have different attributes.

   Y-Means Clustering This technique automatically partitions a data set into a reasonable number of clusters so as to classify the data items into normal and abnormal clusters. The main advantage of Y-Means clustering algorithm is that it overcomes the three shortcomings of K-means algorithm namely dependency on the initial centroids, dependency on the

number of clusters and degeneracy. Y-means clustering eliminates the drawback of empty clusters. The main difference between Y-Means and K-Means is that the number of clusters in Y-Means is a self-defined variable instead of a user-defined constant. If the value of K is too small, Y-Means increases the number of clusters by splitting clusters. On the other hand, if value of K is too large, it decreases the number of clusters by merging nearby clusters. Y-Means determines an appropriate value of K by splitting and linking clusters even without any knowledge of item distribution. This makes Y-Means an efficient clustering technique for intrusion detection since the network log data is randomly distributed and the value of K is difficult to obtain manually. Y-means uses Euclidean distance to evaluate the similarity between two items in the data set.

Fuzzy C-Means Clustering (FCM) is an unsupervised clustering algorithm based on fuzzy set theory that allows an element to belong to more than one cluster. The degree of membership of each data item to the cluster is calculated which decides the cluster to which that data item is supposed to belong. For each item, we have a coefficient that specifies the membership degree of being in the kth cluster as follows:

Figure 2.1: Formula

$$u_{ij} = \sum_{k=1}^{n} (d_{ij} / d_{ik})^{(2/m-1)}$$

where, d_{ij} - distance of i^{th} item from j^{th} cluster, d_{ik} - distance of i^{th} item from k^{th} cluster and m - fuzzification factor.

The existence of a data item in more than one cluster depends on the fuzzification value m defined by the user in the range of $[0, 1]$ which determines the degree of fuzziness in the cluster. Thus, the items on the edge of a cluster may be in the cluster to a lesser degree than the items in the center of the cluster. When m reaches the value of 1 the algorithm works like a crisp partitioning algorithm and for larger values of m the overlapping of clusters tends to be more. The main objective of fuzzy clustering algorithm is to partition the data into clusters so that the similarity of data items within each cluster is maximized and the similarity of data items in different clusters is minimized. Moreover, it measures the quality of partitioning that divides a dataset into C clusters.

2. **Classification** it is the task of generalizing known structure to apply to new data. Common algorithms include decision tree learning, nearest neighbour, Naive Bayesian classification, neural networks and support vector machines. It is a supervised learning technique. A classification based

IDS will classify all the network traffic into either normal or malicious. Classification technique is mostly used for anomaly detection.

3. **Regression** Attempts to find a function which models the data with the last error.

4. **Association rule learning** Searches for relationship between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis. Association rule mining determines association rules and/or correlation relationships among large set of data items. The mining process of association rule can be divided into two steps as follows:

   (a) Frequent Item set Generation, Generates all set of items whose support is greater than the specified threshold called as minsupport.

   (b) Association Rule Generation, From the previously generated frequent item sets, it generates the association rules in the form of if then statements that have confidence greater than the specified threshold called as minconfidence.

   The basic steps for incorporating association rule for intrusion detection is as follows:

   (a) The network data is arranged into a database table where each row represents an audit record and each column is a field of the audit records.

   (b) The intrusions and user activities shows frequent correlations among the network data. Consistent behaviours in the network data can be captured in association rules.

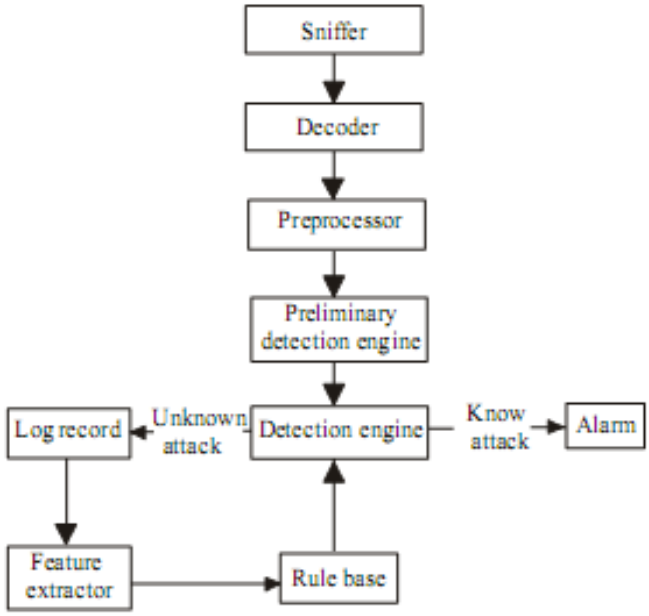
   (c) Rules based on network data can continuously merge the rules from a new run to aggregate rule set of all previous runs.

   (d) Thus with the association rule, we get the capability to capture behaviour for correctly detecting intrusions and hence lowering the false alarm rate.

2.3.1.2 Approach

Zhou et al. [8] they propose a IDS based on data mining technology. In figure 2.2 we can see the structure diagram.

Figure 2.2: Intrusion detection system structure diagram



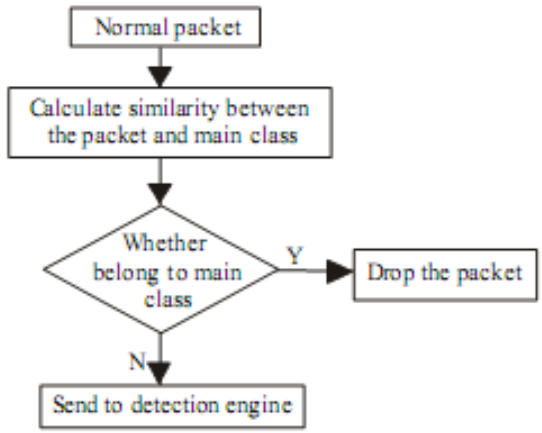
System module function summary:

- **Sniffer**: Mainly acquire data, grab packets from network.
- **Decoder**: Mainly decode and analyze the datagram, store the results.
- **Preprocessor**: Transform the packet to the format for data mining, restructure and process code conversion before matching.
- **Preliminary detection engine**: Mainly filter out normal network packets.
- **Detection engine**: Mainly match rule. It uses K Means algorithm as the clustering analysis algorithm.
- **Log records**: Include packets information which produced by unknown network normal behaviour and unknown intrusion behaviour.
- **Feature extractor**: Make correlation analysis of the data in a log, conclude the new association rule, and add it to the rule base. It uses Apriori algorithm correlation analysis.

- **Alarm**: Transmit an alert when there is an abnormal behaviour.

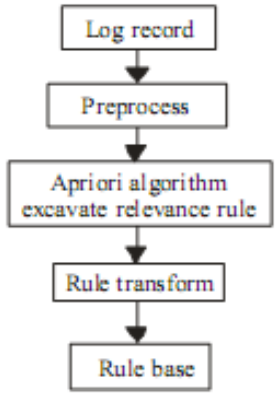
The Workflow: The workflow of the intrusion detection system based on data mining is introduced as follows. Firstly, the sniffer grabs network packets which are analyzed by the decoder. Then preprocessor will process the parsing packets by calling pretreatment function. Secondly, after through the preliminary detection engine, normal packets will be discarded off, and the abnormal packets will be processed by detection engine. Through matching rule, it shows that there are invaded behaviors when successful. At the same time, the system will transmit an alert and prevent intrusion behavior. If it is not successful, the new network normal behavior model will be recorded into log. Finally, the system will make the correlation analysis for the log through the data mining algorithm. If there is a new rule generation, it will be added to the rule base. **Feature extractor**: The workflow of preliminary detection engine using K Means clustering analysis algorithm is shown in figure 2.3.

Figure 2.3: The module Workflow



Feature extractor: The aim of feature extractor is to mine association rules through association rules mining algorithm. First it analyses the abnormal packets, which had been processed by the pretreatment; and then obtains potential or new intrusion behaviour patterns through the Apriori association rules algorithm and produces the corresponding association rule set; Finally it transforms the rule into the intrusion detection rule and adds it to the rule base. The module workflow is shown in figure 2.4.

Figure 2.4: The module Workflow



Results: From the four tables (Table 3 to 6 in figure 2.5), the two important parameters (cluster radius and threshold) have a great influence on the clustering and false detection rate. When threshold is fixed, as the clustering radius increase, the network behaviour pattern classes become fewer. When cluster radius is unchanged, as threshold value becomes lower, the false detection rate becomes higher. Therefore, according to the needs and actual situation of practical applications, they need to adjust cluster radius and threshold to achieve a satisfactory result. Aiming at weakness of self-adaptation ability, low false alarm rate and high misinformation rate of the current most of the intrusion detection system. This study has designed and implemented an intrusion detection system framework based on data mining technology, and has introduced the process of correlation analysis data mining algorithm that how to construct into the intrusion detection model. The test results have shown that the intrusion detection based on data mining system, which overcomes certain limitations of the intrusion detection system, provides self-adaptability, improves the detection efficiency, and reduces the previous deviations caused by domain experts hand writing mode.

Figure 2.5: Results

Table 1: Description of attributes

Attribute name	Description	Category
Ip-len	Length of packet	Continuous type
Ip-ttl	Lifetime	Continuous type
Tcp-win	TCP window size	Continuous type
Ip-options_len	Length of the IP protocol rule option	Continuous type
Tcp-options-len	Length of the protocol rule option	Continuous type
Dsize	Payload size of packet	Continuous type
Udp-len	Length of UDP data packet	Continuous type

Table 2: Data sheet in example.txt

Attribute name	Average value	Mean absolute error
Ip-len	28090	15365
Ip-tl	80	24
Tcp-win	45862	21325
Ip-options-len	0.00142	0.00279
Tcp-options-len	8.16	7.968
Dsize	568.3	596.65
Udp-len	38956	13598

Table 3: Data clustering table

Cluster radius	Network normal behavior pattern class	Network abnormal behavior pattern class
1	0	142
2	1	136
3	3	128
4	6	112
5	9	91

Table 4: Data false detection rate table

Cluster radius	1	2	3	4	5	10
False detection rate (%)	0	0.06	0.15	0.36	0.71	0.86

Table 5: Data clustering table

Threshold	Network normal behavior pattern class	Network abnormal behavior pattern class
100	69	61
200	52	69
300	39	76
500	21	99
800	9	113

Table 6: Data false detection rate table

Threshold	100	200	300	500	800
False detection rate (%)	0.043	0.026	0.015	0.09	0.03

2.3.2 Machine learning techniques

2.3.2.1 Theory

Machine learning is a branch of artificial intelligence, which is about the construction and study of systems that can learn from data Amor et al. [9]. There exist two techniques called decision tree and Bayesian network, and they will now be explained.

Decision Tree A decision tree is composed of three basic elements:

1. **decision node** specifying a test attribute.
2. **edge or branch** corresponding to the one of the possible attribute values which means one of the test attribute outcomes.
3. **leaf** which is also named an answer node, contains the class to which the object belongs.

In decision trees, two major phases should be ensured:

- *Building the tree* Based on a given training set, a decision tree is built. It consists of selecting for each decision node the appropriate test attribute and also to define the class labeling each leaf.
- *Classification* In order to classify a new instance, we start by the root of the decision tree, the we test the attribute specified by this node. The result of this test allows to move down the tree branch relative to the attribute value of the given instance. This process will be repeated until a leaf is encountered. The instance is then being classified in the same class as the one characterizing the reached leaf.

Bayesian Network Bayes networks are one of the most widely used graphical models to represent and handle uncertain information. Bayes networks are specified by two components:

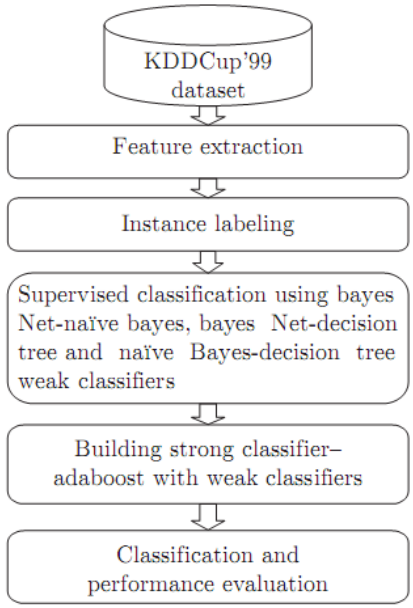
1. *Graphical component* is composed of a directed acyclic graph(DAG) where vertices represent events and edges are relations between events.

2. *Numerical component* consisting in a quantification of different links in the DAC by a conditional probability distribution of each node in the context of its parents.

Naive Bayes are very simple Bayes networks which are composed of DAGs with only one root node (called parent), representing the unobserved node, and several children, corresponding to observed nodes, with the strong assumption of independence among child nodes in the context of their parent.

2.3.2.2 Approach

Figure 2.6: Proposed work



The processes of the proposed system (figure 2.6) is briefly explained in the following Natesan et al. [10]:

- **Process 1**: Preprocessing: For each network connection,the following three major groups of features for detecting intrusions are extracted. They are Basic features, Content features and Traffic features.

- **Process 2**: Instance Labeling: After extracting KDDCup 99 features from each record, the instances are labeled as Normal or any one of the attack category such as Dos, Probe, R2L and U2R.
- **Process 3**: Selection of weak classifiers: The various weak classifiers used in their proposed system are Naive Bayes, Bayes Net and Decision Tree. They have used the single weak classifier along with the boosting algorithm to improve the classification accuracy.
- **Process 4**: Combining weak classifiers: In order to improve the classification accuracy further it has been proposed to combine two weak classifiers along with the boosting algorithm.
- **Process 5**: Building of strong classifier: A strong classifier is constructed by combining two weak classifiers and boosting algorithm. The strong classifier results in higher attack detection rate than single weak classifier.

Results: The overall detection rate and false alarm rate of the three single weak classifiers are shown in figure 2.7. Decision tree was able to give a high detection rate in the case of DoS and Probe attacks and the Naive Bayes algorithm with Adaboost (AdaBoost is a machine learning algorithm, can be used in conjunction with many other learning algorithms to improve their performance. It calls a weak classifier repeatedly in a series of rounds.) detects the R2L and U2R attacks comparatively better than other algorithms.

Figure 2.7: The attack detection rate of different weak classifiers

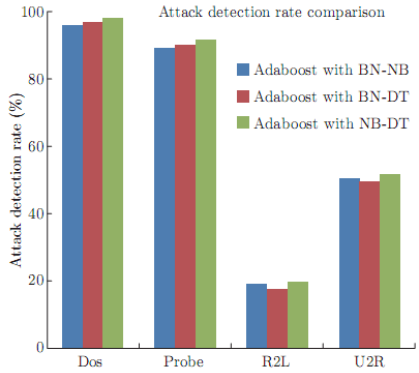
Attack category	% of detection rate		
	Adaboost with Bayes Net	Adaboost with Naïve Bayes	Adaboost with Decision Tree
Dos	95.8	96.7	97.3
Probe	88.5	89.6	91.4
R2L	14.7	19.5	18.4
U2R	49.3	51.2	50.4

Figure 2.8: The false alarm rate of different weak classifiers

S. No	Name of the Weak Classifier	% of false alarm rate	Training Time (sec)	Testing Time (sec)	Time Complexity
1	Bayes Net (BN)	2.78	26.5	1.23	O(mn)
2	Naïve Bayes (NB)	2.61	23.1	0.86	$O(mn^2)$
3	Decision Tree (DT)	2.85	13.2	0.67	O(mn(logn))

Figure 2.9: The attack detection rate of different combinations of weak classifiers

Attack category	% of detection rate		
	Adaboost with BN-NB	Adaboost with BN-DT	Adaboost with NB-DT
Dos	95.7	96.9	97.9
Probe	88.7	90.2	91.6
R2L	19.0	17.7	19.7
U2R	50.4	49.7	51.8



The detection rate of the various attack categories by using the three different combinations of weak classifiers with the Adaboost algorithm shown in figure 2.9. It can be seen that, the performance of NB-DT combination with the Adaboost algorithm is comparatively better than the other two combinations of weak classifiers.

Figure 2.10: False alarm rate comparison

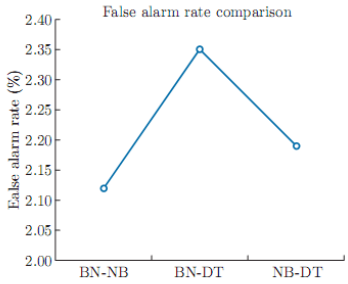


Fig. 5: False alarm rate comparison

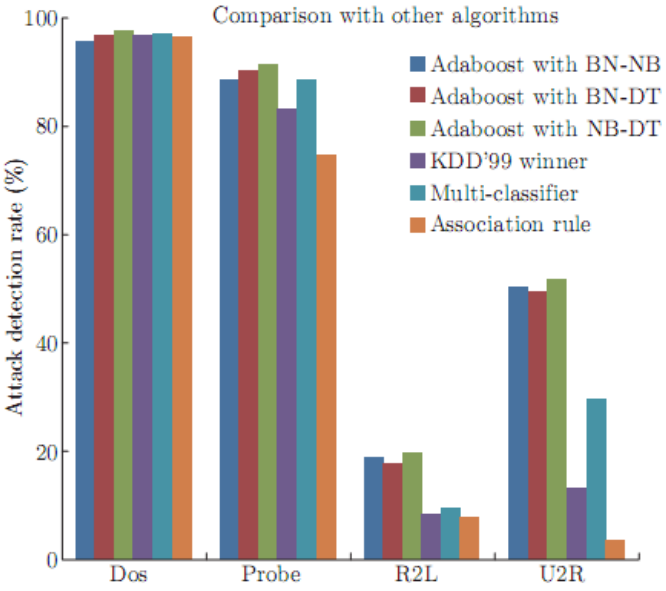
Table 5: The computational time of different weak classifiers

S.No	Combination of weak classifiers with Adaboost	Training Time (sec)	Testing Time (sec)	Time complexity
1	BN-NB	34.3	1.73	$O(mn) + O(mn^2)$
2	BN-DT	29.4	1.13	$O(mn) + O(mn(logn))$
3	NB-DT	21.9	0.92	$O(mn^2) + O(mn(logn))$

The false alarm rate of BN-NB combination of weak classifier with Adaboost decreases to 2.12%, but it shows an increase in the case of BN-DT and NB-DT combinations of weak classifiers as shown in figure 2.10. The training time and the testing time of various combinations of weak classifiers with Adaboost is shown in figure 2.10. The NB-DT combination with the Adaboost took less training time and testing time than other two combinations of weak classifiers.

Summary: They have proposed an Adaboost algorithm with different combination of weak classifiers. The weak classifiers such as Bayes Net, Naive Bayes and Decision tree are used in three different combinations such as BN-NB, BN-DT and NB-DT with Adaboost algorithm to improve the classification accuracy. The various challenges of IDS such as attack detection rate, false alarm rate and computional time for building robust, scalable and efficient system are addressed. It is important to have a low false alarm rate for an IDS with higher detection rate. The experiment result shows that the NB-DT combination with Adaboost algorithm has a very low false-alarm rate with a high detection rate. They have focused mainly to obtain better classification though the time and computational complexities are theoretically high. But practically the time and computational complexities are reduced by processing speed of the computing device.

Figure 2.11: Comparison with other algorithms

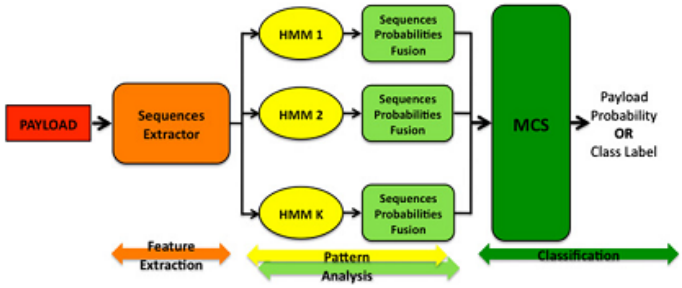


2.3.3 Hidden Markov Models

2.3.3.1 HMMPayl

Ariu et al.[11] they address the problems in payload analysis by proposing a novel solution where the HTTP payload is analyzed using Hidden Markov Models. The proposed system named HMMPayl, performs payload processing in three steps as shown in figure 2.12. First of all, the algorithm they propose for Feature Extraction (step 1) allows the HMM to produce an effective statistical model which is sensitive to the details of the attacks (e.g. the bytes that have a particular value). Since HMM are particularly robust to noise, their use during the Pattern Analysis phase (step 2) guarantees to have a system which is robust to the presence of attacks (i.e., noise) in the training set. In the Classification phase (step 3) they adopted a Multiple Classifier System approach, in order to improve both the accuracy and the difficulty of evading the IDS.

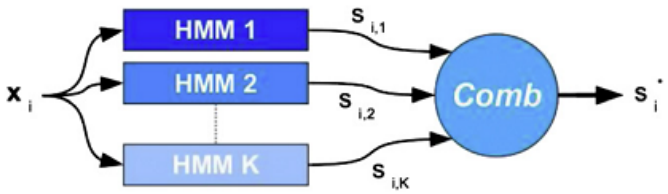
Figure 2.12: A simplified scheme of HMMPayl



Theoretical background - Hidden Markov Models: Hidden Markov Models represent a very useful tool to model data-sequences, and to capture the underlying structure of a set of strings of symbols. HMM is a stateful model, where the states are not observable (hidden). Two probability density functions are associated to each hidden state: one provides the probability of transition to another state, the other provides the probability that a given symbol is emitted from that state.

Theoretical background - Multiple classifier systems: Multiple Classifier Systems (MCS) are widely used in Pattern Recognition applications as they allow to obtain better performance than a single classifier. Ariu et al. [11] they use the MCS paradigm to combine different HMM. A general schema of the proposed HMM ensemble is shown in figure 2.13. A payload x_i is submitted to an ensemble $H = HMM_j$ of K HMM, each HMM_j produces an output s_{ij} and their outputs are combined into a new output s_i^*. Different combination strategies for building a MCS have been proposed in the literature. They can be roughly subdivided into two main approaches, namely the Fusion approach, and the Dynamic approach.

Figure 2.13: A general schema of a MCS based on HMM



Summary: Ariu et al. [11] they proposed an IDS designed to detect attacks

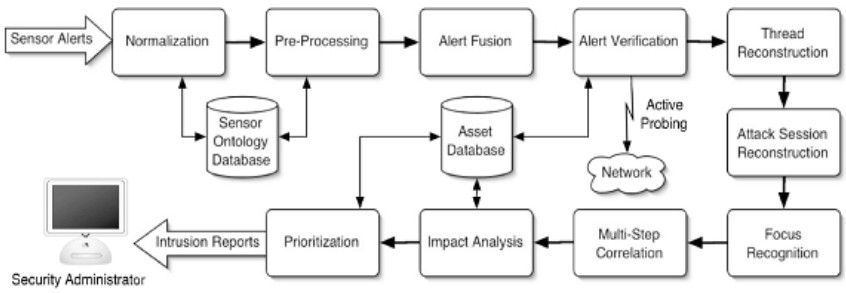
against, Web applications through the analysis of the HTTP payload by HMM. First of all they proposed a new approach for extracting features which exploits the power of HMM in modeling sequences of data. Reported experiments clearly show that this approach provide a statistical model of the payload which is particularly accurate, as it allows detecting attacks effectively, while producing a low rate of false alarms.
HMMPayl has been thoroughly tested on three different datasets of normal traffic, and against four different dataset of attacks. In particular, they have showed that HMMPayl was able to outperform other solutions proposed in the literature. In particular HMMPayl is effective against those attacks such as Cross Site Scripting and SQL-Injection, whose payload statistic is no significantly different from that of normal traffic. These attacks are particularly hard to be detected, as the performance of IDS such as PAYL and McPAD clearly show. In addition, they also showed that the high computational cost of HMMPayl can be significantly reduced by randomly sampling a small percentage of the sequences extracted from the payload, without significantly affecting the overall performance in terms of detection and false alarm rates. Moreover, as HMMPayl relies on the Multiple Classifier System paradigm, they tested the performance attained by the ideal Score Selector as a measure of the maximum gain in performance that could be attained by exploiting the complementarity of the HMM, Experimental results show that the accuracy can be improved with an accurate design of the fusion stage. It is clear that, despite the good results attained in their experiments, the algorithm implemented by HMMPayl could be further improved.
First of all, HMMPayl does not take into account the length of the payload. As different lengths of the payload produce significantly different statistics, clustering the payloads by length, and using a different model for each cluster, would improve the overall accuracy. The second improvement is related to the random sampling strategy, as the whole sequence set could be randomly split among all the classifiers in the ensemble. In such a way all the information inside the payload would be used, where a single HMM is asked to process a smaller number of sequences. Finally, the third improvement is related to the use of trained combination rules instead of a static rule to combine the HMM.

2.3.3.2 Alert correlation and prediction

When we are dealing with large networks with many sensors, we cope with too many alerts fired from IDS sensors each day. Managing and analyzing such amount of information is a difficult task. There may be many redundant or false positive alerts that need to be discarded. Therefore, in order to extract useful information from these alerts they use an *alert correlation* algorithm, which is the process of producing a more abstract and high-level view of intrusion occur-

rences in the network from low-level IDS alerts. Alert correlation is also used to detect sophisticated attacks, a multistep attack is defined as a sequence of simple attacks that are performed successively to reach some goal. During each step of the attack some vulnerability, which is the prerequisite of the next step, is exploited. In other words, the attacker chains a couple of vulnerabilities and their exploits to break through computer systems and escalate his privilege. In such circumstances, IDSs generate alerts for each step of the attack. In fact, they are not able to follow the chain of attacks and extract the whole scenario. Getting advantage of alert correlation, it is possible to detect complex attack scenarios out of alert sequences. In short, alert correlation can help the security administrator to reduce the number of alerts, decrease the false-positive rate, group alerts based on alert similarities, extract attack strategies, and predict the next steps of the attacks.
Farhadi et al. [12] they propose an alert correlation system consisting of two major and two minor components. Minor components are the Normalization and the Preprocessing components that convert heterogeneous alerts to a unified format and then remove redundant alerts. Major components are the ASEA and the Plan Recognition components that extract current attack scenario and predict the next attacker action. In the ASEA component, they used data mining to correlate IDS alerts. The stream of attacks is received as input, and attack scenarios are extracted using stream mining. While reducing the problem of discovering attack strategies to a stream-mining problem has already been studied in the literature, current data mining approaches seem insufficient for this purpose. They still need more efficient algorithms as there are a plethora of alerts and they need real-time responses to intrusions. In the Plan Recognition component, they used HMM to predict the next attack class of the intruder that is also known as plan recognition. The main objective of the attack plan recognition is to arm the management with information supporting timely decision making and incident responding. This helps to block the attack before it happens and provides appropriate timing for organizing suitable actions.

Figure 2.14: Correlation process overview



The reference architecture: Figure 2.14 represents the integrated correlation

process in their solution.

1. **Normalization and Pre-Processing** They converts heterogeneous events from varying sensors into a single standardized format which is accepted by the other components.

2. **Alert Fusion** It combines alerts issued from different sensors, but related to the same activity.

3. **Alert Verification** It takes an alert as an input and determines if the suspicious corresponding attack is successfully performed. Failed attacks are then labelled so that their effectiveness will be decreased in upcoming correlation phases.

4. **Thread Reconstruction** It combines and series the attacks having the same source and target addresses.

5. **Attack Session Reconstruction** Both network-based alerts and host-based alerts that are related to the same attacks are gathered and associated.

6. **Focus Recognition and Multi-step Correlation** They deals with attack that are potentially targeted at wide range of hosts in the enterprise. The "Focus Recognition" component identifies those hosts to which a considerable number of attacks are targeted or originated from. This component hopefully detects port scanning attempts as well as Denial of Service (DoS) attacks. The "Multi-step correlation" component identifies common attack patterns, which are composed of different zones in the network.

7. **Impact Analysis** It calculates the impact factors of current attacks on the target network and assets.

8. **Prioritization** It ends the process with classifying events in different importance groups providing faster ability to find relevant information about a specific host or site.

Summary: Farhadi et al. [12] they presented a system to correlate intrusion alerts and extract attack scenarios as well as to predict the next attacker action. They reduced the problem finding multistage attacks to sequence mining and the problem of finding next attacker action to sequence modelling and prediction. They used DARPA 2000, to evaluate system performance and accuracy. The results show that the system can efficiently extract the attack scenarios and predict the attackers next action. The system has the following advantages:

1. The ASEA is able to operate in real-time environments.
2. The simplicity of ASEA results in low memory consumption and computational overhead.
3. In contrast to previous approaches, the ASEA combines both prior knowledge as well as statistical relationships to detect casual relationship.
4. The prediction component proposes an unsupervised method to predict the next attacker action.
5. The prediction component does not require any knowledge of the network topology, system vulnerabilities, and system configurations. Unlike Bayesian based methods that usually rely on a predefined attack plan library. HMM can perform in the absence of such information.
6. The prediction component performs high-level prediction; hence the model is more robust against over-fitting. In contrast, other plan recognition methods try to predict exactly the attackers next action.

2.3.4 Honeypot

2.3.4.1 Advantages and disadvantages

Bhumika [13] list the advantages of honeypots:

- **Small Data Sets** Hoeneypots only collect data when someone or something is interacting with them. Organizations that may log thousands of alerts a day with traditional technologies will only log a hundred alerts with honeypots. This makes the data honeypots collect much higher value, easier to manage and simpler to analyze.
- **Reduced False Positives** One of the greatest challenges with with most detection technologies is the generation of false positives or false alerts. The larger the probability that a security technology produces a false positive the less likely the technology will be deployed. Hoenypots dramatically reduce false positives. Any activity with honeypots is by definition unauthorized, making it extremely efficient at detecting attacks.
- **Catching False Negatives** Another challenge of traditional technologies is failing to detect unknown attacks. This is a critical difference between honeypots and traditional computer security technologies which rely upon known signatures or upon statistical detection. Signature-based security

technologies by definition imply that "someone is going to get hurt" before the new attack is discovered and a signature is distributed. Statistical detection also suffers from probabilistic failures - there is some non-zero probability that a new kind of attack is going to get undetected. Honeypots on the other hand can easily identify and capture new attacks against them. Any activity with the honeypot is an anomaly, making new or unseen attacks easily stand out.

- **Encryption** It does not matter is an attack or malicious activity is encrypted, the honeypot will capture the activity. As more and more organizations adopt encryption within their environments (such as SSH,IPsec, and SSL) this becomes a major issue. Honeypots can do this because the encrypted probes and attacks interact with the honeypot as an end point, where the activity is decrypted by the honeypot.

- **IPv6** Hoenypots work in any IP environment, regardless of the IP protocol, including IPv6. IPv6 is the new IP standard that many organizations, such as the Department of Defence, and many countries, such as Japan, are actively adopting. Many current technologies, such as firewalls or IDS sensors, cannot handle IPv6.

- **Highly Flexible** Honeypots are extremely adaptable, with the ability to be used in a variety of environments, everything from Social Security Number embedded into a database, to an entire network of computers designed to be broken into.

- **Minimal Resources** Honeypots require minimal resources, even on the largest of networks. A simple, aging Pentium computer can monitor literally millions of IP addresses.

Bhumika [13] also list the disadvantages:

- **Risk** Honeypots are a security resource the bad guys to interact with, there is a risk that an attacker could use a honeypot to attack or harm other non-honeypot systems. This risk varies with the type of honeypot used. For example, simple honeypotsn such as KFSensor have very little risk. Honeynets, a more complex solution, have a great deal of risk. The risk levels are variable for different kinds of honeypot deployments. The usual rule is that the more complicated the deception, the greater the risk. Honeypots that are high-interaction such as Gen I Honeynets are inherently more risky because there is an actual computer involved.

- **Limited Field of View** Honeypots only see or capture that which interacts with them. They are not a passive device that captures activity to

all other systems. Instead, they only have value when directly interacted with. In many ways honeypots are like a microscope. They have a limited field of view, but a field of view that gives them great detail of information.

- **Discovery and Fingerprinting** Though risk of discovery of a honeypot is small for script kiddies and worms, there is always a chance that advanced blackhats would be able to discover the honeypot. A simple mistake in the deception is all a savvy attacker needs to "fingerprint" the honeypot. This could be a misspelled word in one service emulation or even a suspicious looking content in the honeypot. The hacker would be able to flag the honeypot as "dangerous" and in his next attacks, he would most certainly bypass the honeypot. In fact, armed with the knowledge, an advanced blackhat could even spoof attacks to the honeypot thus redirecting attention while he attacks other vulnerable systems in the network.

2.3.5 Genetic algorithm

2.3.5.1 Theory

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions Wikipedia [14]. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.
The evolution usually starts from a population of randomly generated individuals and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.
A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover

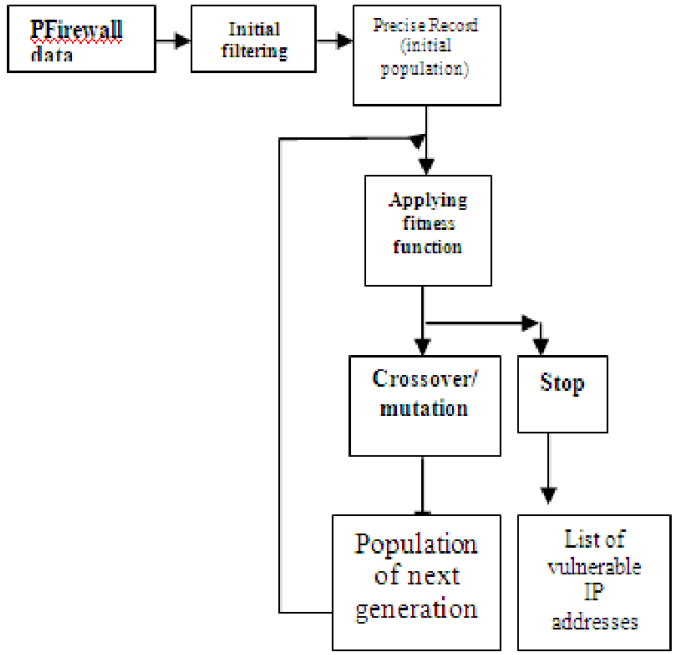
implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

2.3.5.2 Approach

Genetic algorithms are a branch of evolutionary algorithms used in search and optimization techniques. The three dominant functions of a genetic algorithm i.e., selection, crossover and mutation correspond to the biological process: The survival of the fittest. In a genetic algorithm, there is a population of strings (called chromosomes or the genotype of the genome), which encode and indent solutions (called individuals, creatures, or phenotypes). Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and evolves over generations.
In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of individuals are there in a generation, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of individuals, a satisfactory solution may or may not have been reached.

System overview: The detail proposed architecture is shown in figure 2.15. It starts from initial population generation from pfirewall.log file generated by the firewall system. The packets are the filtered out on the basis of rules. Then the precised data packets go through several steps namely selection, crossover and mutation operation. These processes gets generate best individuals. The generated individuals are the verified by the fitness function to generate the population for next generation.

Figure 2.15: Detailed system architecture for GA-RIDS



Results: Dhak et al. [15] they have successfully evolved the rule set and profile of network connection which can detect existing as well as new intrusions. So now the system can be integrated with any of the IDS system to improve the efficiency and the performance of the same. The system can also be able to integrate to the input to the firewall system which can use the rule set defined and generated by the system to block Intrusion. Dhak et al. [15] they have discussed the GA processes and evolution operators also discussed the overall implementation of GA into proposed system. The various operators like selection, crossover and mutation are also discussed. In proposed system they are applying single filtration to the system but in future their plan is to apply multiple filters to enhance the system performance and to reduce time complexity of execution. Again we are planning to apply the proposed system output to the security system like Firewall machine to block the traffic whose IP address entries are made available to the pfirewall.log file and which are detected as vulnerable.

2.3.6 Fuzzy Logic

2.3.6.1 Theory

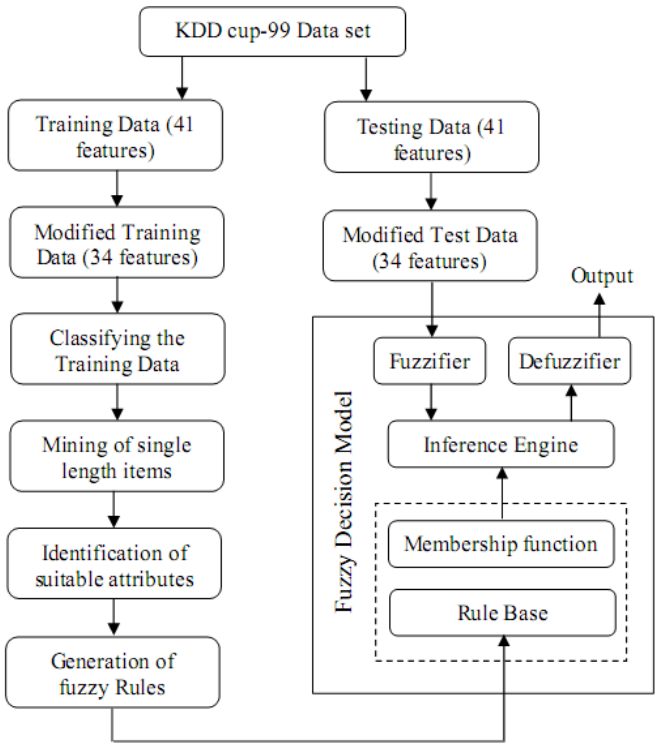
Fuzzy logic is a form of many-valued logic or probabilistic logic; it deals with reasoning that is approximate rather than fixed and exact Wikipedia [16]. Compared to traditional binary sets (where variables may take on true or false values) fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions. Irrationality can be described in terms of what is known as the fuzzjective. Classical logic only permits propositions having a value of truth or falsity. The notion of whether 1+1=2 is absolute, immutable, mathematical truth. However, there exist certain propositions with variable answers, such as asking various people to identify a color. The notion of truth doesn't fall by the wayside, but rather a means of representing and reasoning over partial knowledge is afforded, by aggregating all possible outcomes into a dimensional spectrum. Both degrees of truth and probabilities range between 0 and 1 and hence may seem similar at first.

2.3.6.2 Approach

Shanmugavadivu et al. [17] they propose a system which is a designed fuzzy logic-based system for effectively identifying the intrusion activities within a network. The proposed fuzzy logic-based system can be able to detect an intrusion behaviour of the networks since the rule base contains a better set of rules. Here, they have used automated strategy for generation of fuzzy rules, which are obtained from the definite rules using frequent items. The experiments and evaluations of the proposed intrusion detection system are performed with the KDD Cup 99 intrusion detection dataset. The experimental results clearly show that the proposed system achieved higher precision in identifying whether the records are normal or attack one.

The different steps involved in the proposed system for anomaly-based intrusion detection (shown in figure 2.16) are described as follows:

Figure 2.16: The overall steps of the proposed IDS

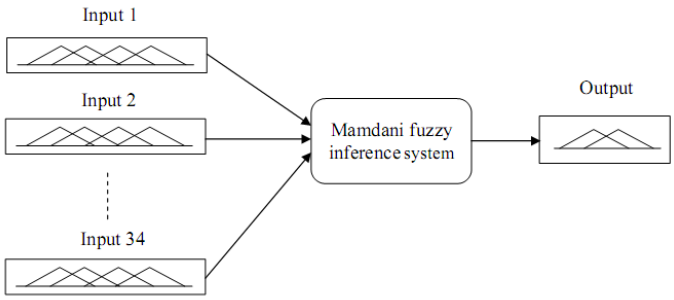


Classification of training data: The first component of the proposed system is of classifying the input data into multiple classes by taking in mind the different attacks involved in the intrusion detection dataset. The dataset they have taken for analysing the intrusion detection behaviour using the proposed system is KDD-Cup 1999 data. Based on the analysis, the KDD-Cup 1999 data contains four types of attacks and normal behaviour data with 41 attributes that have both continuous and symbolic attributes. The proposed system is designed only for the continuous attributes because the major attributes in KDD-Cup 1999 data are continuous in nature. Therefore, they have taken only the continuous attributes for instance, 34 attributes from the input dataset by removing discrete attributes Then, the dataset is divided into five subsets of classes based on the class label. The class label describes several attacks, which comes under four major attacks (Denial of Service, Remote to Local, U2R and Probe) along with normal data. The five subsets of data are then used for generating a better set of fuzzy rules automatically so that the fuzzy system can learn the rules effectively.

Strategy for generation of fuzzy rules: In general, the fuzzy rules given to the fuzzy system is done manually or by experts, who are given the rules by analysing intrusion behaviour. But, in their case, it is very difficult to generate fuzzy rules manually due to the fact that the input data is huge and also having more attributes. But, a few of researches are available in the literature for automatically identifying of fuzzy rules in recent times. Motivated by this fact, they make use of mining methods to identify a better set of rules. Here, definite rules obtained from the single length frequent items are used to provide the proper learning of fuzzy system.

Fuzzy decision module: Zadeh in the late 1960s introduced Fuzzy logic and is known as the rediscovery of multivalued logic designed by Lukasiewicz. The designed fuzzy system shown in figure 2.17 contains 34 inputs and one output, where inputs are related to the 34 attributes and output is related to the class label (attack data or normal data). Here, thirty four-input, single-output of Mamdani fuzzy inference system with centroid of area defuzzification strategy was used for this purpose. Here, each input fuzzy set defined in the fuzzy system includes four membership functions (VL, L, M and H) and an output fuzzy set contains two membership functions (L and H). Each membership function used triangular function for fuzzification strategy.

Figure 2.17: The designed Fuzzy system



Finding an appropriate classification for a test input: For testing phase, a test data from the KDD-cup 99 dataset is given to the designed fuzzy logic system. At first, the test input data containing 34 attributes is applied to fuzzifier, which converts 34 attributes (numerical variable) into linguistic variable using the triangular membership function. The output of the fuzzifier is fed to the inference engine which in turn compares that particular input with the rule base. Rule base is a knowledge base which contains a set of rules obtained from the definite rules. The output of inference engine is one of the linguistic values from the following set Low and High and then, it is converted by the defuzzifier as crisp values. The crisp value obtained from the fuzzy inference engine is

varied in between 0 to 2, where "0" denotes that the data is completely normal and "1" specifies the completely attacked data.

Results: The evaluation metrics are computed for both training and testing dataset in the testing phase and the obtained result for all attacks and normal data are given in figure 2.18, which is the overall classification performance of the proposed system on KDD cup 99 dataset. By analysing the result, the overall performance of the proposed system is improved significantly and it achieves more than 90

Figure 2.18: The classification performance of the proposed IDS

	Metric	Proposed System	
		Training	**Testing**
PROBE	Precision	0.912522	0.912522
	Recall	0.37083	0.37083
	F-measure	0.52735457	0.52735457
	Accuracy	0.906208	0.909323
DOS	Precision	0.993563	0.993828
	Recall	0.90144	0.904154
	F-measure	0.94526236	0.94687236
	Accuracy	0.9478	0.949269
U2R	Precision	0.051948	0.051948
	Recall	0.190476	0.190476
	F-measure	0.08163265	0.08163265
	Accuracy	0.992812	0.993088
R2L	Precision	0.075949	0.075949
	Recall	0.155844	0.155844
	F-measure	0.10212766	0.10212766
	Accuracy	0.991586	0.991909
NORMAL	Precision	0.828439	0.829318
	Recall	0.99416	0.994385
	F-measure	0.90376539	0.90438129
	Accuracy	0.910852	0.903019

2.4 Attacks and threats

Looking at IDS in the perspective of attacks and threats, one can wonder if it really can help you protect against every known/unknown intrusion. The Increase in technology has brought more sophisticated intrusions, with which the network security has become more challenging. Attackers might have different intentions and each attack might have different level.
IDS such as Snort helps in detecting single step intrusions, but not in detecting multistage / multi step attack and attacker behaviour. This attack and

polymorphic worm seems to stand out as potential intrusions an modern IDS cannot detect. We will explain these two topics in more detail in section 2.4.3 and section 2.4.4.
First of all we will turn our focus on the sources of cyber security threats and types of cyber exploits.

2.4.1 Sources of cyber security threats

First of all we list the sources of cyber security threats based on U. S. G. A. Office [18]. We have:
Bot-network operators use a network, or bot-net, of compromised, remotely controlled systems to coordinate attacks and to distribute phishing schemes, spam, and malware attacks.
Criminal groups seek to attack systems for monetary gain. Specifically, organized criminal groups use spam, phishing, and spyware/malware to commit identity theft and online fraud.
Hackers break into networks for the thrill of the challenge, bragging rights in the hacking community, revenge, stalking others, and monetary gain, among other reasons. While gaining unauthorized access once required a fair amount of skill or computer knowledge, hackers can now download attack scripts and protocols from the internet and launch them against victim sites. Thus, while attack tools have become more sophisticated, they have also become easier to use.
Insiders The disgruntled organization insider is a principal source of computer crime. Insiders may not need a great deal of knowledge about computer intrusions because their knowledge of a target system often allows them to gain unrestricted access to cause damage to the system or to steal system data. The insider threat includes contractors hired by the organization, as well as employees who accidentally introduce malware into systems.
Nations use cyber tools as part of their information-gathering and espionage activities. In addition, several nations are aggressively working to develop information warfare doctrine, programs, and capabilities.
Phishers Individuals, or small groups, execute phishing schemes in an attempt to steal identities or information for monetary gain. Phishers may also use spam and spyware/malware to accomplish their objectives.
Spammers Individuals or organizations distribute unsolicited e-mail with hidden or false information in order to sell products, conduct phishing schemes, distribute spyware/malware, or attack organizations (i.e., denial of service).
Spyware/malware authors Individuals or organizations with malicious intent carry out attacks against users by producing and distributing spyware and malware. Several destructive computer viruses and worms have harmed files and hard drives, including the Melissa Macro Virus, the Explore.Zip worm, the

CIH (Chernobyl) Virus, Nimda, Code Red, Slammer, and Blaster.
Terrorists Terrorists seek to destroy, incapacitate, or exploit critical infrastructures in order to threaten national security, cause mass casualties, weaken the U.S. economy, and damage public morale and confidence. Terrorists may use phishing schemes or spyware/malware in order to generate funds or gather sensitive information.

2.4.2 Types of cyber exploits

Next we list the types of cyber exploits based on U. S. G. A. Office [18]. We have:
Denial of service A method of attack from a single source that denies system access to legitimate users by overwhelming the target computer with messages and blocking legitimate traffic. It can prevent a system from being able to exchange data with other systems or use the Internet.
Distributed denial of service A variant of the denial of service attack that uses a coordinated attack from a distributed system of computers rather than from a single source. It often makes use of worms to spread to multiple computers that can then attack the target.
Exploit tools Publicly available and sophisticated tools that intruders of various skill levels can use to determine vulnerabilities and gain entry into targeted systems.
Logic bombs A form of sabotage in which a programmer inserts code that causes the program to perform a destructive action when some triggering event occurs, such as terminating the programmer's employment.
Phishing The creation and use of e-mails and Web sites-designed to look like those of well-known legitimate businesses, financial institutions, and government agencies—in order to deceive Internet users into disclosing their personal data, such as bank and financial account information and passwords. The phishers then use that information for criminal purposes, such as identity theft and fraud.
Sniffer Synonymous with packet sniffer. A program that intercepts routed data and examines each packet in search of specified information, such as passwords transmitted in clear text.
Trojan horse A computer program that conceals harmful code. A Trojan horse usually masquerades as a useful program that a user would wish to execute.
Virus A program that infects computer files, usually executable programs, by inserting a copy of itself into the file. These copies are usually executed when the infected file is loaded into memory, allowing the virus to infect other files. Unlike a computer worm, a virus requires human involvement (usually unwitting) to propagate.
Vishing A method of phishing based on voice-over-Internet-Protocol technology and open- source call center software that have made it inexpensive for

scammers to set up phony call centers and criminals to send e-mail or text messages to potential victims, saying there has been a security problem, and they need to call their bank to reactivate a credit or debit card, or send text messages to cell phones, instructing potential victims to contact fake online banks to renew their accounts.
War driving A method of gaining entry into wireless computer networks using a laptop, antennas, and a wireless network adapter that involves patrolling locations to gain unauthorized access.
Worm An independent computer program that reproduces by copying itself from one system to another across a network. Unlike computer viruses, worms do not require human involvement to propagate.
Zero-day exploit A cyber threat taking advantage of a security vulnerability on the same day that the vulnerability becomes known to the general public and for which there are no available fixes.

2.4.3 Multi step attack

Attacks have changed in form, function, and sophistication from just a few years ago FireEye [19]. Next-generation threats utilize both mass-market malware designed to infect many systems as well as sophisticated, zero-day malware to infect targeted systems. They blend multiple attack vectors cutting across Web, email, and application-based attacks. And today's attacks are aimed at getting valuable data assets sensitive financial information, intellectual property, authentication credentials, insider information and each attack is often a multi-staged effort to infiltrate networks, spread, and ultimately ex-filtrate the valuable data.
From the common Zeus/Zbot infections to the targeted Stuxnet malware, cyber attacks have proven effective at stealing sensitive data, causing financial loss, and damaging corporate reputations. Cybercriminals are transacting billions of dollars in cyber activities. Nation-states are using malware in cyber espionage to spy on opposition activists and disrupt adversary's critical infrastructure. Because of the high stakes, zero-day exploit development and other criminal activities are well funded. This has led to an active underground ecosystem that trades and sells access to systems residing within some of the most sensitive networks in the world.

Next-generation threats are complex, cutting across multiple attack vectors to maximize the chances of breaking through network defences. Multi-vector attacks are typically delivered via the Web or email. They leverage application or operating system vulnerabilities, exploiting the inability of conventional network-protection mechanisms to provide a unified defence. In addition to using multiple vectors, advanced targeted attacks also utilize multiple stages to

penetrate a network and then extract the valued information. This makes it far more likely for attacks to go undetected. The five stages of the attack lifecycle are as follows:

- **Stage 1:** System exploitation. The attack attempts to set up the frst stage, and exploits the system using drive-by attacks in casual browsing. Its often a blended attack delivered across the Web or email threat vectors, with the email containing malicious URLs.
- **Stage 2:** Malware executable payloads are downloaded and long-term control established. A single exploit translates into dozens of infections on the same system. With exploitation successful, more malware executables—key loggers, Trojan backdoors, password crackers,and file grabbers are then downloaded. This means that criminals have now built long-term control mechanisms into the system.
- **Stage 3:** Malware calls back. As soon as the malware installs, attackers have cracked the first step to establishing a control point from within organizational defences. Once in place, the malware calls back to criminal servers for further instructions. The malware can also replicate and disguise itself to avoid scans, turn off anti-virus scanners, reinstall missing components after a cleaning, or lie dormant for days or weeks. By using callbacks from within the trusted network, malware communications are allowed through the firewall and will penetrate all the different layers of the network.
- **Stage 4:** Data ex-filtration. Data acquired from infected servers is ex-filtrated via encrypted files over a commonly allowed protocol, such as FTP or HTTP, to an external compromised server controlled by the criminal.
- **Stage 5:** Malware spreads laterally. The criminal works to move beyond the single system and establish long-term control within the network. The advanced malware looks for mapped drives on infected laptops and desktops, and can then spread laterally and deeper into network file shares. The malware will conduct reconnaissance: it will map out the network infrastructure, determine key assets, and establish a network foothold on target servers.

Firewalls: Firewalls allow generic http Web traffic. Next-generation firewalls (NGFW) add layers of policy rules based on users and applications. NGFW consolidate traditional protections such as anti-virus and IPS but do not add dynamic protection that can detect next- generation threat content or behaviour.

IPS: Signatures, packet inspection, DNS analysis, and heuristics will not detect anything unusual in a zero-day exploit, especially if the code is heavily disguised or delivered in stages.

Anti-virus and Web malware filtering: Since the malware and the vulnerability it exploits are unknown (zero-day), and the website has a clean reputation, traditional anti-virus and Web flters will let it pass. The volume of vulnerabilities in browser plug-ins like Adobe and the exponential combinations of these browsers with operating systems make it hard for anti-virus vendors to keep up.

Email spam filtering: Spoofed phishing sites use dynamic domains and URLs, so blacklisting lags behind criminal activities. It takes more than two days to shut down the average phishing site. Malicious code can also be carried in on laptops, USB devices, or via cloud-based file sharing to infect a machine and spread laterally when it connects into the network. It is common for mobile systems to miss updates to DAT fles and patches, so they are vulnerable to both known and unknown exploits. In general, even up-to-date machines can be infected using zero-day exploits and social engineering techniques, especially when the system is off the corporate network.
Once in place, malware may replicate itself—with subtle changes to make each instance look unique and disguise itself to avoid scans. Some will turn off anti-virus scanners, reinstall after a cleaning, or lie dormant for days or weeks.
Eventually, the code will phone home to the criminal for further instructions, a new payload or to deliver login credentials, financial data, and other valuables. Many compromised hosts provide a privileged base so the criminal can explore further or expand his botnet with new victims.
Most companies dont analyse outbound traffic for these malicious transmissions and destinations. Those organizations that do monitor outbound transmissions use tools that look for known bad actor addresses and regulated data.

2.4.4 Polymorphic worm

A polymorphic worm (PW) is a worm that changes its appearance with every instance (based on Kolesnikov et al. [20]). As a result, byte sequences of different worm instances may look completely different. However, the actual code of the PW typically stays the same.
To change its appearance, a PW can use methods similar to those used by polymorphic viruses. One common method is to take the original code of a worm, encrypt it with a random key, and generate a short decryptor for the key. The polymorphic decryptor (PD) and the key changes with each instance. The code

of the worm does not. This operation is typically performed by the polymorphic engine (PE), included as part of the worms code.
A sophisticated PW can mutate both itself and the exploits it uses. Possible elements of such a PW include:

Attack vectors to penetrate systems. Sophisticated worms use many vectors of attack. The set includes exploits for stack, heap, and other types of overflows, backdoors left by other worms, password sniffing, Man-in-the-Middle attacks, and so forth.

Invariants for attack vectors. A PE uses attack invariants to decide what parts of an attack are volatile so they can be changed without preventing the attack. One example of such invariants are the offsets in an exploit for placing the return addresses and handlers, e.g., Windows Structured Exception Handling-based (SEH) exploitation.

Polymorphic Engine (PE). A PE will generate the mutated versions of the PD and the attacks.

Worm body code. In a simple case, the worm's body might simply contain code that selects an attack vector, generates a set of destinations, mutates the attack and itself using the PE, then sends out the mutated instances.

Traditional signature based IDS, though effective for known attacks but failed to handle the zero-day attack promptly. Recent works on polymorphic worms does not guarantee accurate signature in presence of noise in suspicious flow samples. Paul et al. [21] they propose PolyS, an improved version of Hamsa, a network based automated signature generation scheme to thwart zero- day polymorphic worms. They contribute a novel architecture that reduces the noise in suspicious traffic pool, thus enhancing the accuracy of worm's signature. They also propose a signature generation algorithm for successfully matching polymorphic worm payload with higher speed and memory efficiency.
The typically structure of a polymorphic worm is:

Protocol Framing: is necessary for branch down the code execution path, where software vulnerability exist. The protocol framing string is invariant across all instances of polymorphic worms.

Return Address: Return address or function pointers are the values used to overwrite a jump target to redirect the server execution. Typically a 32 bit integer, of which first 23-bit are normally same across all worm samples. Return address is another invariant part in polymorphic worms.

Exploit Code: These invariant bytes are necessary for abusing vulnerability. It also activates decryption routines and ensures identical malicious activities in all attacks.
Encrypted worm code (Payload): It contains the code to perform malicious activities. In presence of strong encryption routines, the worm payloads take different values in different infection.

Decryption Routine: Its function is to decrypt the encrypted payload by decryption key and passes the control to worms code to start execution. Decryption routines are obfuscated in different instances of polymorphic worms.

Decryption Key: Worm payload is encrypted by polymorphic engines by different keys in different instances. To decrypt the worms payload, corresponding decryption key is required.

Wild Card bytes: These bytes may take any values without affecting the functioning of worms and their spreading capabilities.

In summary, polymorphic worms have two classes of bytes; invariant and variant bytes. Invariant bytes remain same across all instances of the worms while variant bytes change its value in every infection attempt. Typically variant bytes are protocol framing string, exploit code and return address. The other components are in general variant across different instances of a polymorphic worm.

Li et al. [22] presents a survey and comparison of Internet worm detection and containment schemes. They first identify worm characteristics through their behaviour, and then classify worm detection algorithms based on the parameters used in the algorithms. Furthermore, they analyze and compare different detection algorithms with reference to the worm characteristics by identifying the type of worms that can and cannot be detected by these schemes. After detecting the existence of worms, the next step is to contain them. This article explores the current methods used to slow down or stop the spread of worms. The locations to implement detection and containment, as well as the scope of each of these systems/methods, are also explored in depth. Finally, this arti-

cle points out the remaining challenges of worm detection and future research directions.

Figure 2.19: Categorization of worm characteristics

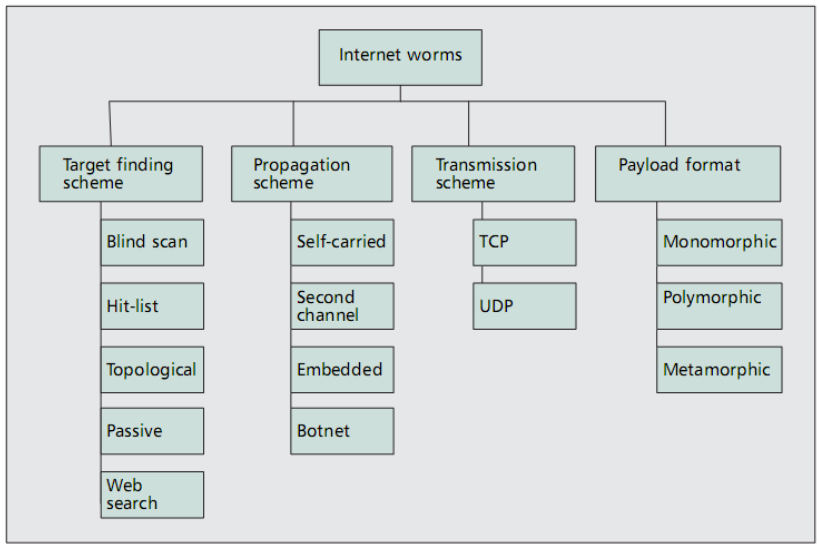
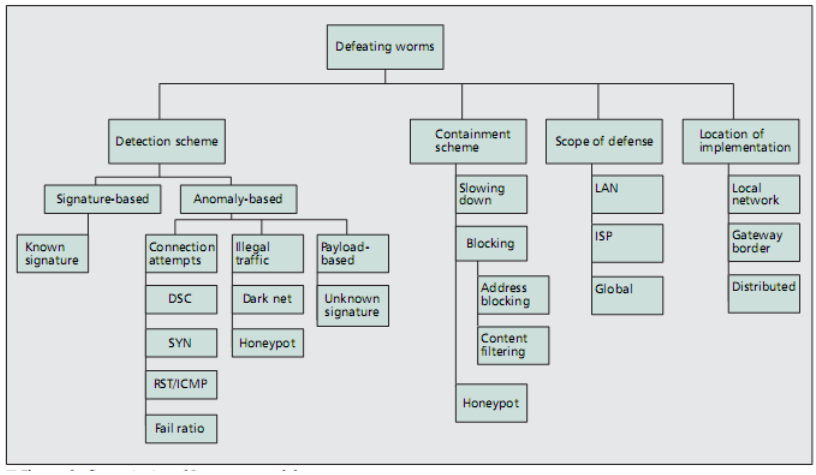


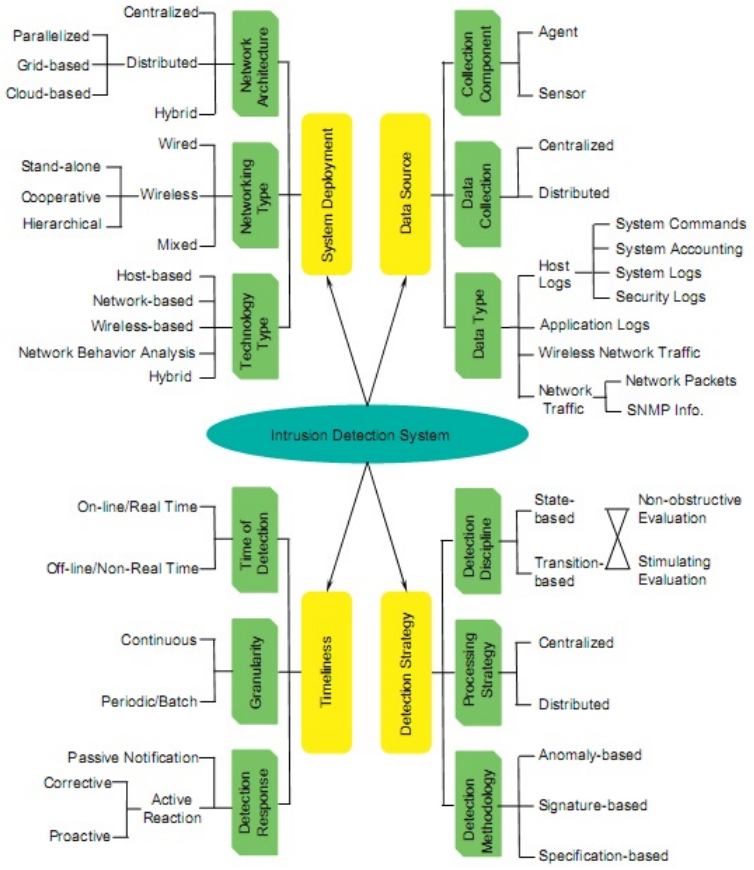
Figure 2.20: Categorization of Internet worm defence



The boxes in the two figures 2.19 and 2.20 are the topics which the article Paul et al. [22] cover. We will not explain the topics, because this would take to much space, and in the end, not accomplish its purpose.

2.5 Taxonomy

Figure 2.21: An overview of IDS taxonomy



2.6 Challenges

Based on Hassan [23] we give an insight of which challenges there exist when a company installs a IDS.

- **Human Intervention** - IDS technology itself is experiencing a lot of enhancements. It is therefore very important for organizations to clearly

define their prospect from the IDS implementation. The IDS has not reached a level where it does not require human interference. There exist of course some automation features like reporting the administrator in case of detection of a malicious activity, avoiding the malicious connection for a configurable period of time, dynamically changing a routers access control list in order to prevent a malicious connection etc. Therefore the security administrator must investigate the attack once it is detected and reported, determine how it occurred, correct the problem and take necessary action to prevent the occurrence of the same attack in the future.

- **Historical analysis** - It is very important factor to monitor IDS logs regularly on a daily basis. Today's IDS has not yet achieved the level where it can provide historical analysis of the intrusive activities detected over a span of time. This is still a manual activity. Hence it is vital for an organization to have a distinct incident handling and response plan if an intrusion is detected and reported by the IDS. Also, the organization should have expert security personnel to handle this kind of situation.

- **Deployment** - The success of an IDS implementation depends to a large degree on how it has been deployed. In most cases, it is required to apply a fusion solution of network based and host based IDS to gain from both cases. In fact one technology complements the other. This decision can differ from one organization to another. A network based IDS is an instant choice for many organizations because of its capability to monitor multiple systems and also the truth that it does not need a software to be loaded on a production system different from host based IDS. Some organizations implement a hybrid solution. Organizations installing host based IDS solution needs remember that the host based IDS software is processor and memory challenging. So it is very important to have sufficient available resources on a system before establishing a host based sensor on it.

- **Sensors** - It is important to maintain sensor to manager ratio. There is no strict rule for calculating this ratio. It depends upon how many different types of traffic is monitored by each sensor and in which background. Most of the organizations deploy a ratio of 10:1, while some organizations maintain 20:1 and some others go for 15:1. It is very important to plan the baseline strategy before starting the IDS implementation and avoid false positives. A poorly configured IDS sensor may post a lot of false positive ratios to the console and even a ratio of 10:1 or even enough better sensors to the console ratio can be missing.

- **False positive and negative alarms rate** - It is not possible for a IDS to be ideal because the network traffic is so complicated. The erroneous results in IDS are divided into two types: false positives and false

negatives. False positives take place when the IDS erroneously identify a problem with benign traffic. False negatives occur when redundant traffic is overlooked by the IDS. Both create problems for security administrators or practitioners and demands that the malicious threats must be detected powerfully. A greater number of false positives are generally more acceptable but can burden a security administrator with bulky amounts of data to filter through. However, because it is unnoticed, false negatives do not provide a security administrator a chance to check the data. Therefore IDS to be implemented should minimize both false positive and negative alarms.

- **Signature database** - A common policy for IDS in detecting intrusions is to remember signatures of known attacks. The signature patterns must be acknowledged first. New threats are often unrecognisably by eminent and popular IDS. Signatures can be masked as well. There exist a challenge by the ongoing event between new attacks and detection systems. It is therefore important to update the signature database every time a different kind of attack is detected and repair for the same is available.

- **Monitor traffic in large networks** - It is important that the NIDS is placed tactically correct in a network, or else attacks can avoid NIDS sensors by passing through alternative ways in the network. Moreover, though many IDS products available in the market are efficient to distinguish different types of attacks, they may fail to recognize attacks that use many attack sources. Many IDS cannot cleverly correlate data from numerous sources. Newer IDS technologies must influence integrated systems to increase an overview of distributed intrusive activity. Therefore IDS must be able to successfully monitor traffic in a large network.

Chapter 3

Typical architecture of Intrusion Detection Systems

In this chapter we introduce the Common Intrusion Detection Framework (CIDF), next we explain general packet inspection and list some pattern matching algorithms. We select Snort as our main IDS, and therefore explain its different components. Lastly we go into more details about Snort, re-examining the performance bottleneck and Snort improvements attempts.

The chapter contribute to give a basic understanding of general IDS architecture, used pattern matching algorithms, and focus on Snorts architecture, internals and improvements attempts.

3.1 The Common Intrusion Detection Framework (CIDF)

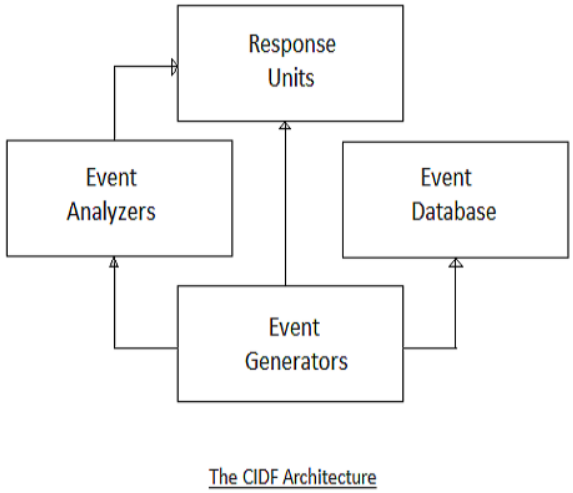
The Common Intrusion Detection Framework (CIDF) is a recent standardization effort, which began in early 1997 among all the DARPA-funded intrusion detection projects (Pieprzyk et al. [24]). The idea of a common framework arose when DARPA decided to make all the intrusion detection systems that

was funding interoperate, and therefore make the benefits arising from these projects more useful and accessible to the wider community.
Although initially confined within these DARPA projects, in April 1998 the work of the CIDF community was put forward to the IETF (LA meeting) with the aim of creating an IETF working group on CIDF. As mentioned in the CIDF specification document, the goal of the CIDF specification is twofold:
1. The specification should allow different IDSs to interoperate and share information as fully as possible.
2. The specification should allow components of IDSs to be easily reused in a contexts different from those they were designed for.

All CIDF components deal in GIDOs (Generalized Intrusion Detection Objects) that are represented via a standard common format. GIDOs are data that is moved around in the intrusion detection system. GIDOs can represent events that occurred in the system, analysis of those events, prescriptions to be carried out, or queries about events.
The CIDF specification covers a number of issues related to the creation of a framework:

- A set of architectural conventions for how different parts of IDSs can be modeled as CIDF framework.
- A way to represent GIDOs, where they can: describe events that occurred in the system. instruct an IDS to carry out some action. Query an IDS as to what has occurred, or describe an IDS component.
- A way to encode GIDOs into streams of bytes suitable for transmission over a network or storage in a file.
- Protocols for CIDF components to find each other over a network and exchange GIDOs.
- Application Programming Interfaces to reuse CIDF components.

Figure 3.1: CIDF architecture



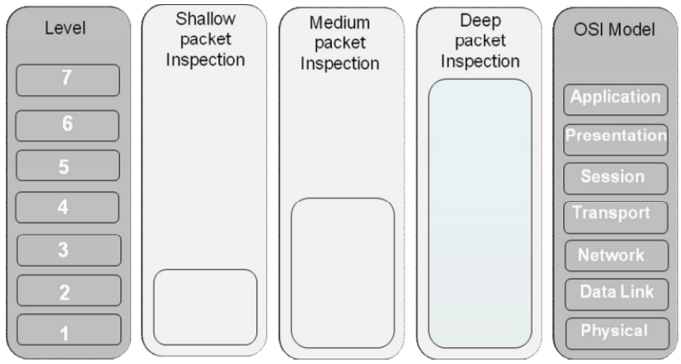
The CIDF architecture consists of four components: Event generators, Event analyzers, Event databases and Response units (refCIDFArch). Figure 3.1 shows the interconnection among these components. Event generator components collect, filter and convert event data. Analyzer components analyze any kind of event data transmitted to them by any CIDF component. Database components are the repositories for any kind of data when the storage is necessary. Response components issue commands in response to attacks and carry out actions such as killing processes, resetting connections, altering file permissions, etc. CIDF is designed to be an open architectural standard. It is independent of implementation languages, operating systems, and network protocols.

3.2 Packet inspection

Deep Packet Inspection (DPI) is a technology that enables the network owner to analyse internet traffic, through the network, in real-time and to differentiate them according to their payload [25]. Since, this has to be done on real time basis at the high speeds it cannot be implemented by software running on normal processors or switches. It has only become possible in the last few years through advances in computer engineering and in pattern matching algorithms. Originally the Internet protocols required the network routers to scan only the header of an Internet Protocol (IP) packet. The packet header contains the ori-

gin and destination address and other information relevant to moving the packet across the network. The "payload" or content of the packet, which contains (all or part of) the text, images, files or applications transmitted by the user, was not considered to be a concern of the network operator. DPI allows network operators to scan the payload of IP packets as well as the header. DPI systems use expressions to define patterns of interest in network data streams. The equipment is programmed to make decisions about how to handle the packet or a stream of packets based on the recognition of a regular expression or pattern in the payload. This allows networks to classify and control traffic based on the content, applications, and subscribers. Many of the functions provided by DPI technology have been available before to limited extent depending on the level of packet analysis. Packet inspection technologies that have been in use in networking environments can be classified in three classes. These three classes are "shallow", "medium", and "deep" packet inspection. Figure 3.2 provides a visual representation of the depth of inspection each of these technologies allows for.

Figure 3.2: OSI model



3.2.1 Shallow Packet Inspection

Shallow packet inspection (SPI) examines the headers of the packets (which is the information placed at the beginning of a block of data, such as the sender and recipient's IP addresses), as opposed to the body or "payload" of the packet [25]. This kind of packet inspection allows the communications to remain 'virtually anonymous' since the content of the packets is not observed, and the information in the header is used only to route the packet SPI technologies drive the (relatively) simplistic firewalls found in the recent generations of operating systems, such as Windows XP, Windows Vista, and OS X. These firewalls

stand between a particular client computer and the network that it is attached to. They limit user-specified content from either leaving, or being received by, the client computer. When a server sends a packet to a client computer, SPI technologies examine the packets header information and evaluate it against a blacklist. These firewalls, specifically, focus on the source and destination IP address that the packet is trying to access. If the packets header information is on the blacklist, the packet is not delivered. When SPI technology refuses to deliver a packet, the technology simply refuses to pass it along without notifying the source that the packet has been rejected. SPI cannot read beyond the information contained in a header and focuses on the second and third layers in the OSI model. SPI examines the senders and receivers IP address, the number of packets that a message is broken into, the number of hops a packet can make before routers stop forwarding it, and the synchronization data that allows for reassembling the packets into a format that the receiving application can understand. SPI cannot read the session, presentation, or applications layers of a packet; it is unable to peer inside a packets payload to survey the packets contents.

3.2.2 Medium Packet Inspection

Medium Packet Inspection (MPI) is typically used to refer to "application proxies", or devices that stand between end-users' computers and ISP/Internet gateways [25]. These proxies can examine packet header information against their loaded parse-list.When a packet enters the proxy, it is analyzed against a parse-list that system administrators can easily update. A parse-list allows specific packet-types to be allowed or disallowed based on their data format types and associated location on the Internet, rather than on their IP address alone. MPI devices can read the presentation layer of the packets payload and identify facets of the application layer. Using MPI devices, administrators could prevent client computers from receiving flash files from YouTube, or image files from social networking sites. MPI technologies can prioritize some packets over others by examining the application commands that are located within the application layerand the file formats in the presentation layer. MPI devices suffer from poor scalability which limits their usefulness for ISPs, where tens of thousands of applications can be transmitting packets at any given moment.

3.2.3 Deep Packet Inspection

Deep Packet Inspection (DPI) technologies are intended to allow network operators precisely to identify the origin and content of each packet of data that

passes through the networking hubs [25].Whereas MPI devices have very limited application awareness, DPI devices have the potential to look inside all traffic from a specific IP address, pick out the HTTP traffic, then drill even further down to capture traffic headed to and from a specific mail server, and can then reassemble e-mails as they are typed out by the user. DPI devices are designed to determine what programs generate packets, in realtime, for hundreds of thousands of transactions each second.

3.2.4 Challenges

When applying DPI on the network there exist some challenges AbuHmed et al. [26]. In the following we list some of them:

- **The search algorithm complexity**: the complexity of the algorithm and the operations of comparison against the signatures of intruder decrease the throughput of the system. Thus, search algorithms are the main focus point in DPI researches, whereas matching process is resource consuming. For example, the string matching routines in Snort account for up to 70% of total execution time and 80% of instructions executed on real traces.

- **Increasing number of intruder signatures**: attacks increases every day and therefore there is need for new intruder signatures. Therefore, the large number of signatures makes the task of IDS harder whereas the matching process must inspect traffic against all attacks fingerprints.

- **The overlapping of signatures**: the signatures of attacks usually are not general so the signatures can be categorized into groups according to common properties like protocol type. For example http packet in Snort has 1096 signatures. Therefore, there is a need for process the packets before matching process.

- **The location of signature unknown**: due to verity types of attacks on different types of applications, the pattern of intruders is not localized in specific place in the packet which means that the IDS must inspect all the payload of the packet against the attacker signatures.

- **Encrypted data**: the data which is encrypted cannot be inspected by DPI. However,there are some solutions to overcome this problem by plugging the DPI component behind the decryption device.

3.3 Pattern matching algorithms

We give in this section an brief explanation of single-keyword and multiple-keyword pattern matching algorithms. These basic pattern matching algorithms is used in IDS, when searching and detecting intrusions.

3.3.1 Single-Keyword pattern matching algorithms

Single keyword matching means locating all occurrences of a given pattern in the input text string Aoe [27]. Next we will briefly explain the well known Single-keyword pattern algorithms there exist.

Brute Force Algorithm This approach scans the text from left to right and checks the characters of the pattern character by character against the substring of the text string beneath it. Let m and n be the lengths of the pattern and the text. In the BF approach, the longest (worstcase) time required for determining that the pattern does not occur in the text is O(mn).

Knuth-Morris-Pratt Algorithm The KMP algorithm scans the text from left to right, using knowledge of the previous characters compared to determine the next position of the pattern to use. The algorithm first reads the pattern and in O(m) time constructs a table, called the next function, that determines the number of characters to slide the pattern to the right in case of a mismatch during the pattern matching process. The expected theoretical behaviour of the KMP algorithm is O(n+m), and the next function takes O(m) space.

Boyer-Moore Algorithm The BM algorithm approach is the fastest pattern matching algorithm for a single keyword in both theory and practice. The BM compares characters in the pattern form right to left. If a mismatch occurs, the algorithm computes a shift, that is, the amount by which the pattern is moved to the right before a new matching is attempted. It also reprocesses the pattern in order to produce the shift tables. The expected theoretical behaviour of the BM is equal to that of the KMP, but many experimental results show that the BM is faster than the KMP.

Karp-Rabin Algorithm The KR uses extra memory to advantage by treating each possible m-character section (where m is the pattern length) of the text string as a keyword in a standard hash table, computing the hash function of it, and checking whether it equals the hash function of the pattern. Although the KR algorithm is linear in the number of references to the text string per characters passed, the substantially higher running time of this algorithm makes

it unfeasible for pattern matching in strings.

3.3.2 Multiple-Keyword pattern matching algorithms

Aho-Corasick Algorithm This algorithm is multipattern matching algorithm which locates all the occurrence of a set of patterns in a text of string Hasib et al. [28]. It first creates deterministic finite automata for all the predefined patterns and then by using automaton, it processes a text in a single pass. It consists of constructing a finite state pattern matching automata from the patterns and then using the pattern matching automata to process the text string in a single pass.

Commentz Walter Algorithm It combines the Boyer-Moore technique with the Aho-Corasick algorithm. In preprocessing stage, differing from Aho-Corasick algorithm, Commentz Walter algorithm constructs a converse state machine from the patterns to be matched. Each patten to be matched adds states to the machine, starting from right side and going to the first character of the pattern, and combining the same node. In searching stage, Commentz Walter algorithm uses the idea of Boyer-Moore algorithm. The length of matching window is the minimum pattern length. In matching window is the minimum pattern length. In matching window, Commentz Walter scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch it uses a precomputed shift table to shift the window to the right.

Rabin-Karp Algorithm It calculates a hash value for the pattern, and for each M-character subsequence of text to be compared Khan et al. [29]. If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence. If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

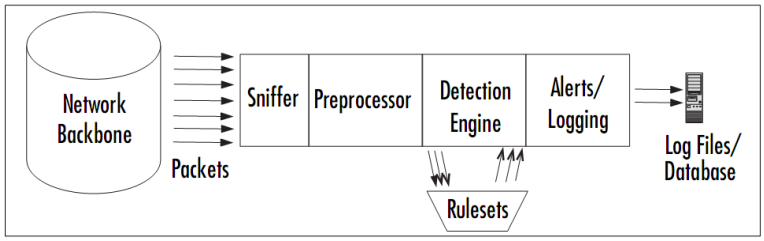
3.4 Snort as an example

Snort is an open source NIDS using among other signatures to detect intrusions, and has become the de facto standard for IPS. It is selected as our example, which we want to investigate. We look at its components, into its internals, re-examings the performance bottle neck, and lastly give examples of Snort improvements.

3.4.1 Components

Snort's architecture consists of four basic components: The sniffer, preprocessor, the detection engine and the output Syngress [30].

Figure 3.3: Figure of Snort architecture



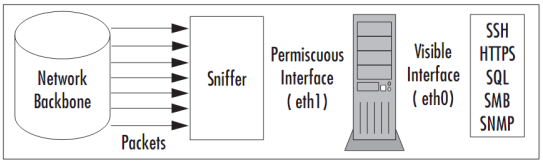
3.4.1.1 Packet sniffer

A packet sniffer is a device used to tap into networks. It works in a similar fashion to a telephone wiretap, but its used for data networks instead of voice networks. In the case of the Internet, this usually consists of IP traffic, but in local LANs and legacy networks, it can be other protocols suites, such as IPX and AppleTalk traffic. Because IP traffic consists of many different higher-level protocols, many sniffers analyze the various network protocols to interpret the packets into something human-readable. Packet sniffers have vaious uses:

- Network analysis and troubleshooting
- Performance analysis and benchmarking
- Eavesdropping for clear-text passwords and other interesting tidbits of data

Encrypting your network traffic can prevent people from being able to sniff your packets into something readable. The sniffer needs to be set up to obtain as many packets as possible. As a sniffer, Snort can save the packets to be processed and viewed later as a packet logger. See figure 3.4.

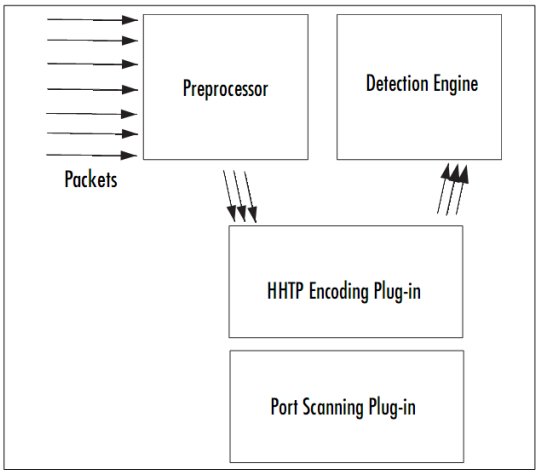
Figure 3.4: Figure of Snorts sniffer component



3.4.1.2 Preprocessor

A preprocessor takes the raw packets and checks them against certain plug-ins. These plug-ins check for a certain type of behaviour from the packet. Once the packet is determined to have a particular type of behaviour, it is then sent to the detection engine. From figure 3.5, you can see how the preprocessor uses its plug-ins to check a packet. Snort supports many kind of preprocessors and their attendant plug-ins, covering many commonly used protocols as well as larger-view protocol issues such as IP fragmentation handling, port scanning and flow control, and deep inspection of richly featured protocols. This is an incredibly useful feature for an IDS because plug-ins can be enabled and disabled as they are needed at the preprocessor level, allocating computational resources and generating alerts at the level optimal for your network.

Figure 3.5: Figure of Snorts preprocessor component



3.4.1.3 Detection engine

Once packets have been handled by all enabled preprocessors, they are handed off to the detection engine. The detection engine is the meat of the signature-based IDS in Snort. The detection engine takes the data that comes from the preprocessor and its plug-ins, and that data is checked through a set of rules. If the rules match the data in the packet, they are sent to the alert processor. The rules themselves consist of two parts:

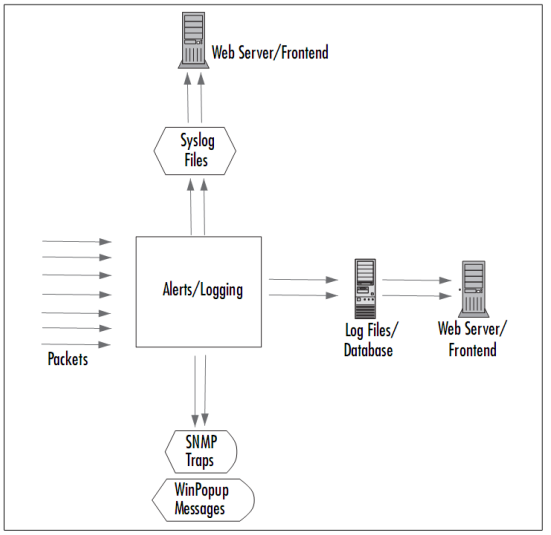
- **The rule header** The rule header is basically the action to take, type of network packet (TCP,UDP,ICMP, and so forth), source and destination IP addresses, and ports.
- **The rule option** The option is the content in the packet that should make the packet match the rule.

The detection engine and its rules are the largest portion of new information to learn and understand with Snort. Snort has a particular syntax that it uses with its rules. Rule syntax can involve the type of protocol, the content, the length, the header, and other various elements, including garbage characters for defining buffer overflow rules.

3.4.1.4 Alerting component

After the Snort data goes through the detection engine, it needs to go out somewhere. If the data matches a rule in the detection engine, an alert is triggered. Alerts can be sent to a log file, through a network connection, through UNIX sockets or Windows Popup (SMB), or SNMP traps. The alerts can also be stored in an SQL database such as MySQL and Postgres. See figure 3.6.

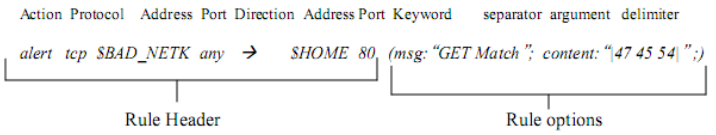
Figure 3.6: Figure of Snorts alerting component



3.4.2 Snort internals

As mentioned in section 3.4.1.3 the snort rules consist of two parts, namely rule header and rule option (based on Sen [31]). (1) Rule header: This has a static definition and i composed of the 5 tuple as shown. It must be present in every rule. (2) Rule options: This has variable definitions. It is not always present, and it has more than 50 options available to account for different requirements in the description of possible rules.

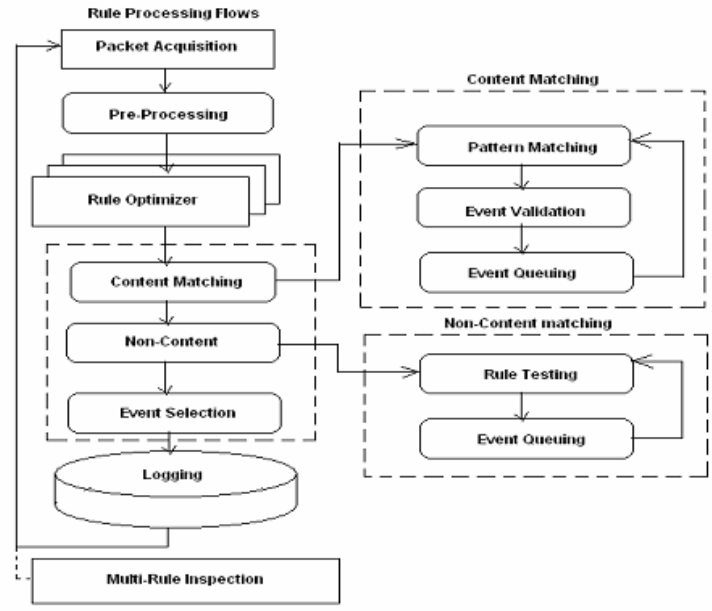
Figure 3.7: Rule syntax



Snort 2.0 uses a High Performance Multi-Rule Inspection engine for detecting patterns. Packets are first analyzed by the Rule Optimizer to select the appropriate set of rules for inspection. The Multi-Rule Inspection engine searches for rule matches, builds a queue of detected rule matches, and selects best rule match for logging. The process of inspecting Network traffic for matches may be described in these three steps:

(1) Rule Optimization to produce efficient rule sets for inspection.
(2) Set based inspection algorithms that perform high-speed multi-pattern content searches.
(3) Parameterized inspection techniques which allow for complicated pattern inspection.

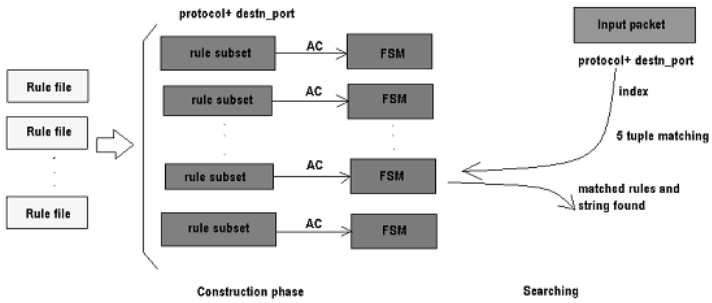
Figure 3.8: Rule syntax



Rule Sets and Search Types used in general by snort are the four categories of rules for testing: (1) Protocol Field Rules (2)Generic Content Rules (3) Packet anomaly Rules (4) IP Rules.

The whole process cycle that Snort takes, starts from taking in the rule files, constructing the state machine, and the walking through it and searching for pattern matches with the input strings. It may be represented as shown in figure 3.9.

Figure 3.9: Diagram of the pattern matching module of snort



At the heart of Snort lies the pattern matching algorithm. It has been estimated that almost 70-80 percent of the time spent by snort in pattern detection. Therefore, it is important to improve the efficiency of pattern matching algorithm.

3.4.3 Re-examining the performance bottle neck in Snort and Bro

Designing a high-speed network intrusion detection system (NIDS) has attracted much attention in recent years due to ever-increasing amount of network traffic and ever-complicated attacks. Numerous studies have been focused on accelerating pattern matching for a high-speed design because some early studies observed that pattern matching is a performance bottleneck. However, the effectiveness of such acceleration has been challenged recently. Po-ChingLin et al. [32] therefore re-examines the performance bottleneck by profiling two popular NIDSs, Snort and Bro with various types of network traffic in detail. In the profiling, they find pattern matching can dominant the Snort execution if the entire packet payloads in the connections are scanned, while executing the policy scripts is an obvious bottleneck in the Bro execution. The work suggest three promising directions towards a high-speed NIDS design for future research: a method to precisely specify the possible locations of the signatures in long connections, a compiler to transform the policy scripts to efficient binary codes for execution, and an efficient design of connection tracking and packet reassembly.

In addition to profiling the decoding and preprocessing stages, they have divided the rule matching of Snort into the following stages to study the significance of pattern matching:

- *mpse:* Snort searches a buffer of application payload for a group of patterns associated with the application protocols. This stages name, mpse, denotes multiple pattern matching searching engine because the search executes a multiple string matching algorithm.
- *Rule tree matching:* Snort verifies the other rule options in the detection rules whose selected patterns are found in the mpse stage. If the verification result is positive, a rule match will be asserted.
- *pcre:* This stage performs regular expression matching when Snort verifies the pcre rule option because matching regular expressions is commonly considered to be a heavy part in deep packet inspection. Although this stage is part of rule tree matching, they measured its execution time separately from that of rule tree matching due to its importance.
- *others:* This stage involves the other processing in rule matching, such as raising an alert or reading the initial configurations.

3.4.3.1 Baseline profiling with normal traffic

- The proportion of the mpse stage in the total execution can vary significantly, depending on the input packet traces and the system configuration that specifies how deep to look into in a connection. In other words, speeding up pattern matching can have only limited contributions to the overall performance in certain conditions. Precisely specifying the possible locations of the patterns in the long connections will be helpful to the overall performance.
- The mpse stage is effective for filtering out the rules that cannot be matched, so the execution time in the rule tree matching, including the pcre stage, is mostly rather short. Specifically, accelerating regular expression matching therefore contributes little to speed up Snort for normal network traffic. While the observation is plausibly true for normal traffic, they point out that regular expression matching can be still very important because it is not always possible to find a substring in a regular expression to write it in the content option for fast filtering, and an attacker may deliberately forge the packet payload with a signature in the content option to force the execution of regular expression matching.
- The preprocessing stage consumes a noticeable proportion of time. Figure 3.10 summarizes the top three time-consuming preprocessors for each application protocol, and the stream5 preprocessor ranks the first in each row, suggesting tracking the TCP/UDP flows and packet reassembly be a target for acceleration, in addition to pattern matching. Besides stream5,

they also find the preprocessors for decoding the commands or fields in the application protocols, e.g. http_inspect and SMTP_detect, may also take a noticeable proportion of time, meaning protocol parsing is the second target to be accelerated in the preprocessing. Snort tracks the error packets returned from the target hosts when detecting port scanning with low sensitivity in the default configuration. They noticed that the P2P traffic involves many ICMP "port unreachable" packets due to the UDP packets destined for incorrect port numbers. The packets can raise the tracking of error packets in the port_scan_detect preprocessor and should account for the higher proportion of execution time in the preprocessing.

Figure 3.10: The top three time-consuming preprocessors

Traffic type	1st	2nd	3rd
HTTP	stream5 (53%)	http_inspect (15%)	port_scan_detect (9%)
SSH	stream5 (54%)	ftp_telnet_checks (10%)	http_inspect (6%)
FTP	stream5 (90%)	port_scan_detect (5%)	frag3 (2%)
SMTP	stream5 (46%)	SMTP_detect (43%)	port_scan_detect (4%)
On-line game	stream5 (45%)	ftp_telnet_checks (14%)	http_inspect (10%)
P2P	stream5 (35%)	port_scan_detect (28%)	ftp_telnet_checks (7%)

3.4.3.2 Profiling with abnormal network traffic

The network traffic in section 3.4.3.1 is all normal, but it is common that abnormal or even malicious traffic is mixed in real network traffic. Therefore, they have also measured the Snort performance to see the impact when Snort process abnormal network traffic.
Splitting packets into small IP fragments and TCP segments is a well-known method to evade NIDS detection. Modern NIDSs have been equipped with the capability of reassembling the split packets to counter the evasion. Snort comes with two preprocessors, frag3 and stream5, to reassemble IP fragments and TCP segments. In this profiling, they have deliberately split packets from the traffic during a period of web browsing into IP fragments and TCP segments using fragroute to test the performance of both preprocessors in Snort.
Figure 3.11 summarizes the execution time and the proportions for the processing stages. For IP fragments, Snort collects and reassembles them before scanning the packet payloads. The preprocessing stage occupies a relatively large proportion, meaning that the stage can heavy when reassembling a number of IP fragments. Similarly, handling a number of small TCP segments also

makes the preprocessing stage a heavy component in the Snort Execution.

Figure 3.11: The execution time in each Snort stage for split packets (in CPU cycles)

Traffic type	Decoding	Preprocessing	Mpse	Rule tree	Pcre	Others	Total
IP fragments (total)	444.34K	1.88M	640.55K	563.10K	127.81K	594.72K	4.25M
IP fragments (per packet)	617.14	2.61K	889.65	782.07	177.52	825.99	5.90K
IP fragments (proportion) (%)	10.46	44.22	15.07	13.25	3.00	14.00	100
TCP segments (total)	50.80M	121.66M	10.94M	4.96M	53.21K	48.37M	236.80M
TCP segments (per packet)	874.51	2.09K	188.38	85.43	0.91	832.63	4.08K
TCP segments (proportion) (%)	21.46	51.38	4.62	2.10	0	20.44	100

They also experimented with the ISCX dataset which involves malicious packet traces for evaluating intrusion detection. They used the malicious traces of HTTP denial of service, distributed denial of service using an IRC botnet, and brute force SSH in the profiling [1]. The profiling results are presented in figure 3.12. On average, the proportion of the execution time in the mpse stage amounts to nearly one-third, and the preprocessing stage also takes nearly one-third of the total execution time. The results show that speeding up pattern matching alone is insufficient for a high-speed design even for malicious packet traces. When looking into the preprocessing stage, they found stream5 is the most time-consuming preprocessor for the three malicious traces, and takes 53.31% of the preprocessing time on average.
To sum up, even for the malicious packet traces that are publicly available and picked for their profiling, pattern matching in the mpse stage takes at most around two thirds of the total execution time.

Figure 3.12: The execution time in each Snort stage for the ISCX dataset (in CPU cycles)

Traffic type	Decoding	Preprocessing	Mpse	Rule tree	Pcre	Others	Total
HTTP DoS (total)	11.96G	43.98G	41.52G	14.29G	5.14G	18.47G	135.36G
HTTP DoS (proportion) (%)	8.83	32.49	30.67	10.56	3.80	13.65	100
IRC DDoS (total)	44.08G	187.87G	326.00G	109.30G	3.43G	125.34G	796.02G
IRC DDoS (proportion) (%)	5.33	22.72	39.42	13.22	4.15	15.16	100
SSH brute force (total)	22.38G	87.26G	59.99G	37.38G	15.53G	28.76G	251.30G
SSH brute force (proportion) (%)	8.91	34.72	23.87	14.87	6.18	11.45	100

3.4.3.3 Profiling with different system configurations

Among the preceding experiments,they are most interested in how the profiling result will be different if the payloads in an entire connection are scanned.In this regard,for example, Snort provides the server_flowdepth configuration option

[1] `www.iscx.ca/dataset`

for the http_inspect preprocessor to set the depth to look into.If the option value is 0, Snort will scan the entire traffic from the server to the client,which amounts to the majority of HTTP traffic. Snort also provides the data_chan option for the ftp_telnet preprocessor.If this option is turned off, Snort will inspect all of the packets for data transfer. Figure 3.13 compares the profiling results from Snort running with the HTTP and FTP traffic, based on the default configuration and that inspecting the entire connection. The comparison implies that slightly modifying the system configuration will result in a significantly different result.The mpse stage becomes dominant in the total execution time if Snort scans the entire packet payloads in the HTTP and FTP connections. The throughput for the HTTP traffic decreases from 83.3 MB/s to 15.38 MB/s, and that for the FTP traffic decreases from 63.49 MB/s to 32.08 MB/s.

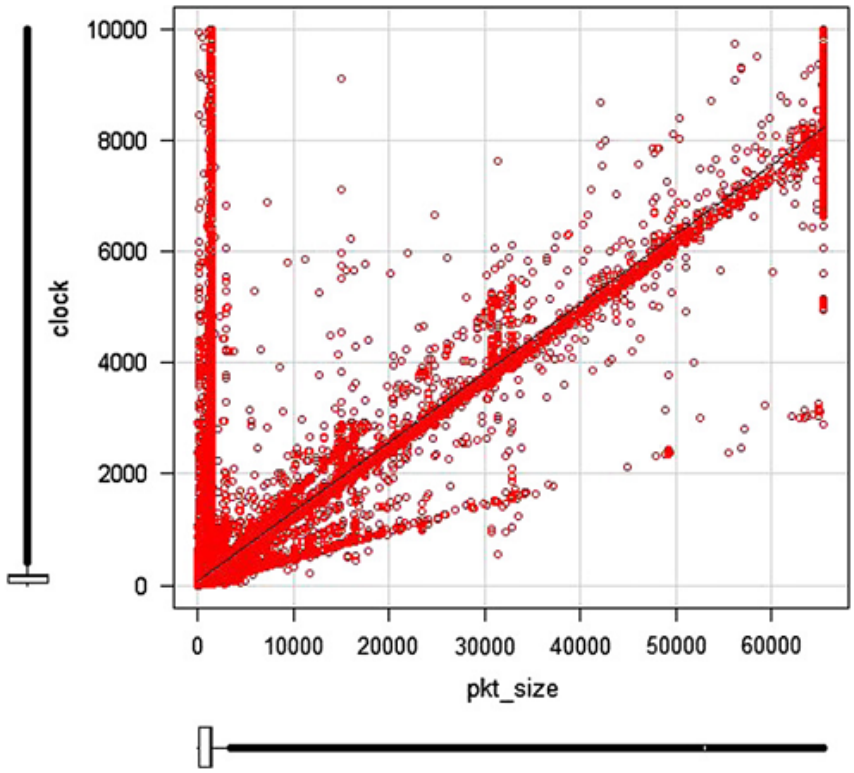
Figure 3.13: Comparison of Snort execution with the default and new configuration (execution time in CPU cycles)

Traffic type	Decoding	Preprocessing	Mpse	Rule tree	Pcre	Others	Total
HTTP default (total)	2.58G	8.67G	5.65G	4.75G	0.92G	6.98G	29.55G
HTTP default (proportion) (%)	8.47	29.15	18.98	12.88	3.05	23.38	100
HTTP new (total)	3.31G	15.28G	202.50G	54.97G	5.17G	34.19G	315.42G
HTTP new (proportion) (%)	1.05	4.84	64.20	17.43	1.64	10.84	100
FTP default (total)	18.30M	38.01M	0.19M	0.34M	27.10K	9.16M	66.03M
FTP default (proportion) (%)	27.72	57.57	0.29	0.51	0.04	13.87	100
FTP new (total)	17.03M	56.62M	935.53M	10.77M	0	38.39M	1,058.34M
FTP new (proportion) (%)	1.61	5.35	88.40	1.02	0	3.62	100

3.4.3.4 Analysing the factors in pattern matching

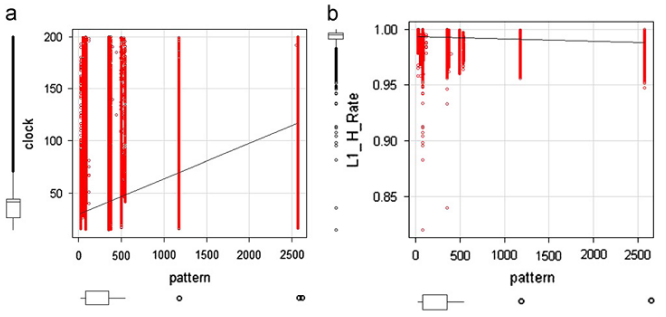
According to the preceding profiling, pattern matching can dominate the execution time if every packet payload is scanned. Besides the algorithm design, two factors can determine the scanning time of pattern matching: the *payload length* and *the number of patterns* in a group.
For the profiling, they used the packet traces of 20 GB from the NCTU Beta-Site, a bulk repository of daily network traffic in a campus. Figure 3.14 presents the relationship between the scanning time and the payload up to 64 kB. As expected, the scanning time is generally proportional to the payload length, except hundreds of outliers, which are just a relatively small proportion of the total number of payloads. This observation is consistent with that Snort runs pattern matching by tracking a finite state machine, and the time complexity is linear to the payload length.

Figure 3.14: The relationship between the scanning time and the payload length)



When looking into the relationship between the average scanning time per character in the payload and the number of patterns simultaneously scanned in a group, as presented in figure 3.15(a), they found that the per-character scanning time can vary wildly, even though the average time slightly increases with the number of patterns in a group. They investigated the hit rates of the L1 cache, as presented in figure 3.15(b), and found that the variation of the hit rates for a given pattern group has a larger effect on the scanning time than the sizes of pattern groups, even though the average hit rates slightly decrease with the increasing number of patterns in a group.

Figure 3.15: The relationship between the per-character scanning time and the sizes of pattern groups. (a) The scanning time vs. the sizes of pattern groups and (b) the L1 cache hit rates vs. the sizes of pattern groups)



3.4.3.5 Profiling with the bulk network traffic

They also demonstrate that the aggregate of the execution time derived from individual sample packet traces can be used to predict the results when profiling with the bulk network traffic in a real environment. They divided the packet traces of 20 GB from the NCTU BetaSite into 10 subsets according to the hashing values from the source/destination IP addresses and the source/destination ports, and then measured the execution time in the Snort stages for each subset. Figure 3.16 compares the aggregate execution time for the subsets and the time for the original bulk traffic.The aggregate time in every stage except the preprocessing is close to the execution time in the same stage for the bulk traffic. Note that the mpse stage dominates the execution time for the bulk traffic. To verify the heavy-tailed nature in the bulk traffic, a connection is defined to be long if the total payloads in either direction are longer than k bytes. If $k = 100.000$ the number of long connections is only 0.55% of the total number of connections, but the payloads in the long connections account for 98.70% of the total payloads in bytes. In other words, the large volume of the bulk traffic is almost contributed by the long connections.
Figure 3.17 summarizes the composition of application protocols in the long connections. The protocols marked OTHER are those not recognized by the port numbers, such as those from P2P applications and on-line games, and they contribute 76.58% of the payloads in bytes. Since the entire payloads are scanned in the long connections , as specified in some of the rules, executing the mpse stage is expensive. They believe that is why past studies have claimed pattern matching is a bottleneck in the NIDS processing. If the rules can be refined to avoid scanning the entire payloads in the long connections by precisely locating

the possible occurrences of malicious content in the payloads, the performance of Snort can be improved significantly, and the false positives can be also reduced.

Figure 3.16: Comparison of the aggregate execution time for the subsets of packet traces and the time for the original bulk traffic (in CPU cycles)

Traffic type	Decoding	Preprocessing	Mpse	Rule tree	Pcre	Others	Total
Aggregate time	19.75G	59.65G	388.57G	8.68G	795.39M	109.57G	586.22G
Time for the bulk traffic	20.20G	125.54G	445.60G	7.94G	631.50M	85.66G	685.94G

Figure 3.17: The composition of application protocols in the long connections

Protocol	Number of connections	% of the payload volume
OTHER (TCP)	1554	68.30
HTTP	1027	13.11
OTHER (UDP)	1452	8.28
SSH	328	7.55
HTTPS	22	2.73
telnet	22	< 0.05
IRC	1	< 0.05
DNS	3	< 0.05
SIMAP	2	< 0.05
ICMP-ECHO	1	< 0.05
X11	1	< 0.05

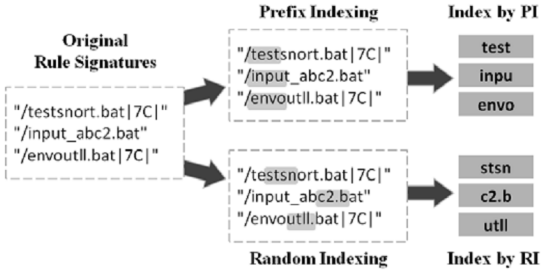
3.4.4 Snort improvement attempts

3.4.4.1 Snort rule indexing

As explained in section 3.4.1.3 the Snort program uses rules to detect potential malicious packets. Because the portion of malicious packets is usually small, it is not efficient to examine incoming packets with all Snort rules. Kang [33] they apply two indexing methods to Snort rules, Prefix Indexing and Random Indexing, to reduce the number of rules to be examined. They focus on the idea that they can apply indexing method to grouped Snort rules. Indexing method is a method which commonly is used to find a certain item from a large set of data. An index is a substring which is part of a Snort rule signature. Each index points to a subset of Snort rules, which have signatures that include the index; therefore the Snort rules are arranged into several groups according to indices. Before the packet inspection stage, the rules are pre-processed to create

indices and group the rules together. One of the main issues in indexing is how to choose substrings from signatures of Snort rules when building indices. The indices for PI(Prefix Indexing) are determined by extracting first N bytes of strings from each Snort rule signature. To use the first N bytes as an index is efficient in cases that the beginning of rule signatures within a ruleset shares a number of common strings. While PI is the simplest way to group Snort rules with indices, grouping with other substrings that appear in the middle of Snort rule signatures could be more appropriate in some cases. Based on this intuition, they derived a naive approach which selects indices randomly from signatures of the Snort rules. They called this naive approach as Random Indexing (RI). RI extracts an N-byte substring from arbitrary point of Snort rule signatures. See figure 3.18 it shows the difference between PI and RI, when applied to the same Snort ruleset.

Figure 3.18: Figure of Random Prefixing and Prefix Indexing



Experiments To obtain their experimental results they used Snort 2.9.0.0. Snort has a number of rule files which are divided into several categories by the types of attacks. They conducted the experiments with Web-cgi ruleset, by applying the two indexing methods. Web-cgi ruleset includes 371 rules to detect the attacks on CGI programs. The total length of signatures for the web-cgi ruleset is 4.686 bytes, with the average of 12.63 bytes per rule. They introduce three statistical values for the performance estimation: the number of indices(NI), the average number of rules per index(ANPI) and the maximum number of rules per index(MNPI). NI refers to the size of the entire indices, which is related to the amount of strings to be examined with an input packet; therefore if NI decreases, the packet inspection process can have better performance. ANPI is another important factor of performance as it has a close relationship with the amount of strings to be inspected; the fewer the strings, the less it takes for the deep packet inspection. They found that the appropriate value for ANPI is between 1.5 and 1.8, experimentally. MNPI has effects on the deep packet inspection time, especially for the worst case. If the size of MNPI, M, is significantly large, the worst case time required for the deep packet inspection increases as the the packet inspection involves at least M times of

comparison to the input packet. To summarize, an indexing result with smaller MNPI, especially which has a close value to ANPI, shows the best performance in the packet inspection. Figure 3.19 shows the experimental results of the two indexing methods on web-cgi ruleset, where the length of each index is set to 4 bytes. The two methods, PI and RI show similar results in most cases. The best case of RI has the smallest MNPI, whose value is also close to ANPI; therefore the indexing with RI, in its best case, provides a fairly equally distributed result. Other results with bigger MNPI values mean that their rule groupings were biased to a certain index. If the biased index is matched, a large number of rules should by fully examined, causing longer deep packet inspection time.

Figure 3.19: Figure of statistical values of web-cgi ruleset

Methods / Values	PI	RI	
		Best	Worst
Length of Index	4	4	4
Number of Indices (NI)	209	175	133
Average Number of Rules per Index (ANPI)	1.77	2.12	2.78
Maximum Number of Rules per Index (MNPI)	13	13	53

To evaluate the performance of packet examination, they estimated the total amount of strings to be examined for an input packet, since the total number of bytes of strings is a critical factor of the performance in the attack detection process, regardless of the string matching approach being used. The smaller the total number of bytes of strings to be examined becomes, the shorter the packet examination time would be taken. They estimate the performance of the index selection algorithm, by calculating the total number of bytes of strings to be examined. The total number of bytes of strings (TBS) is the sum of the number of bytes of all indices (NBI) and the number of bytes of signatures in the Snort rules matched indices (NBR). NBI is the number of indices times the length of the index, as the whole indices should be examined during the inspection. NBR is the sum of the length of the rule signatures that are pointed to by matched indices. To calculate NBR, they need to know which index was matched, because only the rules that are linked to the index will be examined; however, every time the inspection proceeds, different indices will be matched according to incoming packets. There is no way to foresee the indices will be matched unless the packet inspection is actually conducted; therefore they formularized the following equation: NBR = The number of Matched Indices x ANPI x The Average Length of Signatures. They also calculate the average case by measuring the average of the best and worst cases. The equation is: TBS (x) = NI x The Length of the Index + The Number of Rules for $x\%$ of Indices x The Average Length of Signatures per Rule.

Figure 3.20: Figure of the estimation of the amount of strings to be examined for web-cgi ruleset

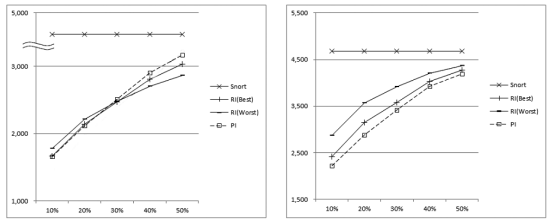
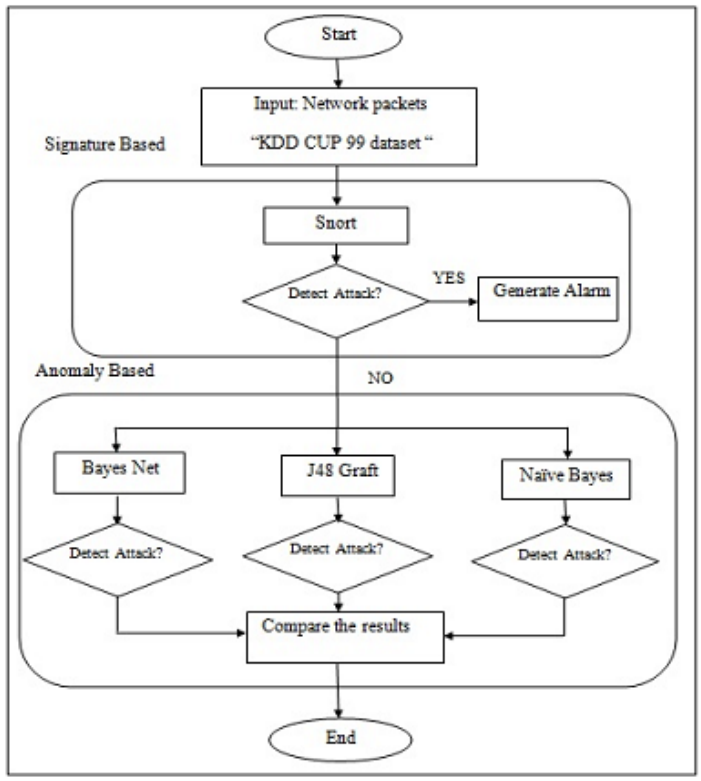


Figure 3.20 shows the results of the simulation for the web-cgi ruleset: Figure 3.20 (a) is for the average case and figure 3.20 (b) for the worst case. The Y axis represents TBS and the X axis represents the percentage of matched indices for an input packet. As can be seen in the graphs, the two index selection algorithms show better performance than the original Snort. RI shows unpredictable performance, showing two different aspects: in the best case, the performance of RI is almost equal to PI but, in the worst case, its performance is far worse than PI. This improvement is a result of having smaller MNPI. It can be said that the rules are well distributed into groups, which also implies that the amount of strings to be examined is normalized. **Summary** In the paper they applied two indexing methods, PI and RI that extracts indices from Snort rule signatures, to reduce the number of rules to be examined. They conducted a number of experiments, which showed significant improvement to the packet inspection, reducing the number of strings to be matched to around 26% of the original Snort.

3.4.4.2 Hybrid IDS using Snort with Naive Bayes

This project starts with analysis network packets by using Snort as 1st phase to detect the entire KDD cup 99 dataset based on Signature based IDS. The 2nd phase use the anomaly detection engine that will be based on Naive Bayes algorithm, J48graft Decision Tree algorithm and Bayes Net algorithm. All the three anomaly detection engines will detect the attacks based on their algorithms. After all the three analysis are done, the researcher makes comparison between all the results in order to evaluate performance of the Hybrid IDS using Snort and Naive Bayes algorithm. In figure 3.21 the general design is showed, which illustrates the whole procedure of this research.

Figure 3.21: General design of the research



Accuracy Accuracy is the proportion of correct classification classes namely True Positive(TP) and True Negative(TN) over the total number classifications.

Figure 3.22: Rate of Accuracy between All the Three Algorithms

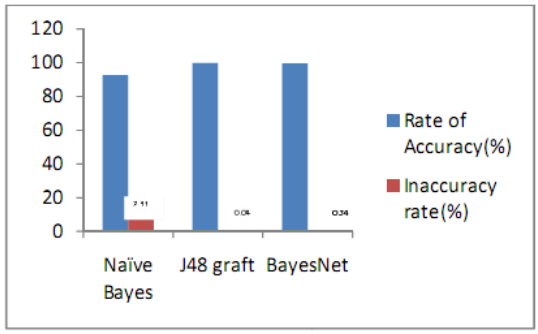
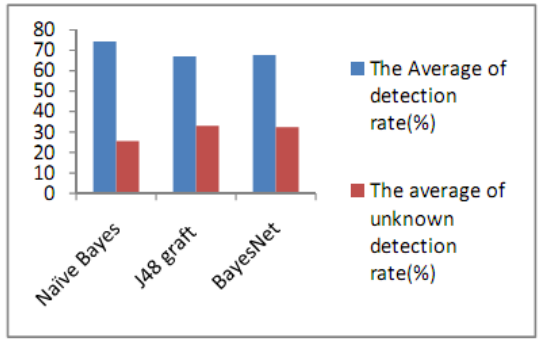


Figure 3.22 compares the rate of accurate and inaccurate classified instances between all the three algorithms. It can be seen clearly that the rate of accuracy with runs J48graft shows better result than the other two algorithms which is 99.96%. This is because of the ability of J48graft to construct tree and classify instance correctly among all the classes type.

Detection Rate The total detected detected attacks amongst all the scanned data is called detection rate.

Figure 3.23: The average of detection rate and unknown detection rate for All the Three Algorithms

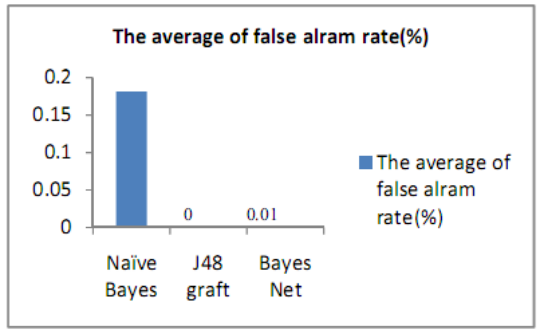


As shown in figure 3.23 the average detection rate and unknown detection rate between the three algorithms. According to the result the average of the detection rate by Naive Bayes is above that of the average rate of detection by Bayes Net and J48graft algorithm respectively. In contrast the unknown detection rate by uses J48graft is much higher than the unknown detection rate by uses Bayes Net and Naive Bayes respectively because the process of learning and classifica-

tion is feeble with uses J48graft to the classes that had been represented by a small number of connections.

False alarm rate False alarm rate is the proportion of normal data which is falsely detected and labeled as an attack.

Figure 3.24: The average of False Alarm Rate for All the Three Algorithms



It can be seen clearly from the graph that the average is zero with runs J48graft while the average is slightly different by runs bayes net and by runs naive bayes algorithm. As stated by the average value, J48graft is better than Bayes Net and Naive Bayes algorithm respectively in term of false alarm rate. This slight difference of the average of false alarm rate is because of the ability of the J48graft to constructed tree and classifies connections better than bayes net and naive bayes respectively.

Chapter 4

How to evaluate intrusion detection systems

This chapter presents the parts in which you can evaluate IDS's. The parts consist of tools and data, hereby an explanation of the available testing tools, and available datasets. In addition, four relevant articles will briefly be explained focusing on Snort test setup. The last part consist of, how we will test and an evaluation based on the IDS/IPS testing framework called pytbull.

We want to evaluate IDS, so it is possible to clarify the limitations.

The chapter contribute to give an answer to, how to evaluate intrusion detection systems.

4.1 Tools and data

The section looks at the different possible testing tools such as pytbull etc. ,and the available datasets such as KDD cup 1999 dataset etc.

4.1.1 Testing tools

If you use Google and search for "intrusion detection system testing tool/framework" you will get many hits. When you try to find a suitable tool to use, only 1 is suitable, and it is called pytbull.

Pytbull Pytbull is a python based flexible IDS/IPS testing framework shipped with more than 300 tests, grouped in 9 modules, covering a large scope of attacks (clientSideAttacks, testRules, badTraffic, fragmentedPackets, multipleFailedLogins, evasionTechniques, shellCodes, denialOfService, pcapReplay). Besides that it is automatic, and free. [1]

Other testing tools exist, and they are:

1. **BackTrack**: is a distribution based on the Ubuntu Linux distribution aimed at digital forensics and penetration testing use.
   BackTrack arranges tools into 12 categories: Information gathering, Vulnerability assessment, Exploitation tools, Privilege escalation, Maintaining access, Reverse engineering, RFID tools, Stress testing Forensics, Reporting tools, Services, Miscellaneous. [2]

2. **Kali Linux**: is a Debian-derived Linux distribution designed for digital forensics and penetration testing. It is maintained and funded by Offensive Security Ltd. developed by rewriting BackTrack, their previous forensics Linux distribution [3].

3. **Metasploit**: is a computer security project that provides information about security vulnerabilities and aids in penetration testing and IDS signature development [4].

4.1.2 Available datasets

In this section we will briefly explain some of the datasets for testing IDS's.

[1] `http://pytbull.sourceforge.net/`
[2] `http://www.backtrack-linux.org/`
[3] `http://www.kali.org/`
[4] `http://www.metasploit.com/`

4.1.2.1 KDD cup 1999 dataset

The 1998 DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs (Nadiammai et al. [34]). The objective was to survey and evaluate research in intrusion detection. A standard set of data includes a wide variety of intrusions simulated in a military network environment. The DARPA 1998 dataset includes training data with seven weeks of network traffic and two weeks of testing data providing two million connection records. A connection is a sequence of TCP packets starting and ending at some well defined times, between source IP address to a target IP address with some well defined protocol. Each connection is categorized as normal, or as an attack, with one specific attack type. The training dataset is classified into five subsets namely Denial of service attack, Remote to Local attack, User to Root attack, Probe attacks and normal data. Each record is categorized as normal or attack, with exactly one particular attack type. They are classified as follows:

- **DOS** (Denial of service attack) Denial of service (DOS) is class of attack where an attacker makes a computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate user access to a machine.
- **R2L** (Remote to local (user) attack) A remote to local (R2L) attack is a class of attacks where an attacker sends packets to a machine over network, then exploits the machine's vulnerability to illegally gain local access to a machine.
- **U2R** (User to root attack) User to root (U2R) attacks is a class of attacks where an attacker starts with access to a normal user account on the system and is able to exploit vulnerability to gain root access to the system.
- **Probing** (Surveillance and other probing) Probing is class of attacks where an attacker scans a network to gather information or find known vulnerabilities. An attacker with map of machine and services that are available on a network can use the information to notice for exploit.

4.1.2.2 NSL-KDD dataset

NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 dataset [5]. Furthermore, the number of records in the NSL-KDD train and test sets are reasonable. This advantage makes it affordable to run

[5] `http://nsl.cs.unb.ca/NSL-KDD/`

the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research work will be consistent and comparable.

The NSL-KDD dataset has the following advantages over the original KDD data set:

- It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.
- There are no duplicate records in the proposed test sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on frequent records.
- The number of selected records from each of level difficulty group is inversely proportional to the percentage of records in the original KDD data set. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques.

4.1.2.3 UNB ISCX Intrusion Detection evaluation dataset

In network intrusion detection system(IDS), anomaly-based approaches in particular suffer from accurate evaluation, comparison, and deployment which originates from the scarcity of adequate datasets. Many such datasets are internal and cannot be shared due to privacy issues, others are heavily anonymized and do not reflect current trends, or they lack certain statistical characteristics. These deficiencies are primarily the reasons why a perfect dataset is yet to exist. Thus, researchers must resort to datasets which they can obtain that are often suboptimal. As network behaviors and patterns change and intrusions evolve, it has very much become necessary to move away from static and one-time datasets toward more dynamically generated datasets which not only reflect the current traffic compositions and intrusions, but are also modifiable, extensible, and reproducible.
At ISCX [6], a systematic approach to generate the required datasets is introduced to address this need. The underlying notion is based on the concept of profiles which contain detailed descriptions of intrusions and abstract distribution models for applications, protocols, or lower level network entities. Real traces are analyzed to create profiles for agents that generate real traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP. In this regard, a set of guidelines is established to outline valid datasets, which set the basis for generating

[6] Information Security Centre of Excellence: `http://www.iscx.ca/`

profiles. These guidelines are vital for the effectiveness of the dataset in terms of realism, evaluation capabilities, total capture, completeness, and malicious activity. The profiles are then employed in an experiment to generate the desirable dataset in a testbed environment. Various multi-stage attacks scenarios were subsequently carried out to supply the anomalous portion of the dataset. The intend for this dataset is to assist various researchers in acquiring datasets of this kind for testing, evaluation, and comparison purposes, through sharing the generated datasets and profiles.
To simulate user behavior, the behaviors of their Center's users were abstracted into profiles. Agents were then programmed to execute them, effectively mimicking user activity. Attack scenarios were then designed and executed to express real-world cases of malicious behavior. They were applied in real-time from physical devices via human assistance; therefore, avoiding any unintended characteristics of post-merging network attacks with real-time background traffic. The resulting arrangement has the obvious benefit of allowing the network traces to be labeled. This is believed to simplify the evaluation of intrusion detection systems and provide more realistic and comprehensive benchmarks [7].

4.2 How have other tested?

In this section we will cover relevant articles, which focus on testing IDS. We will not list the article's results and conclusions, because the main goal is to explain the techniques and tools that the authors has used for testing the relevant IDS.

4.2.1 Performance evaluation of Snort and Suricata

Alhomoud et al. [35] they have tested and analysed the performance of Snort and Suricata. Both programs were implemented in three different platforms (ESXi virtual server, Linux 2.6 and FreeBSD) to simulate a real environment.

[7] `http://www.iscx.ca/datasets` and `http://ali.shiravi.com/84`

Figure 4.1: Network design setup

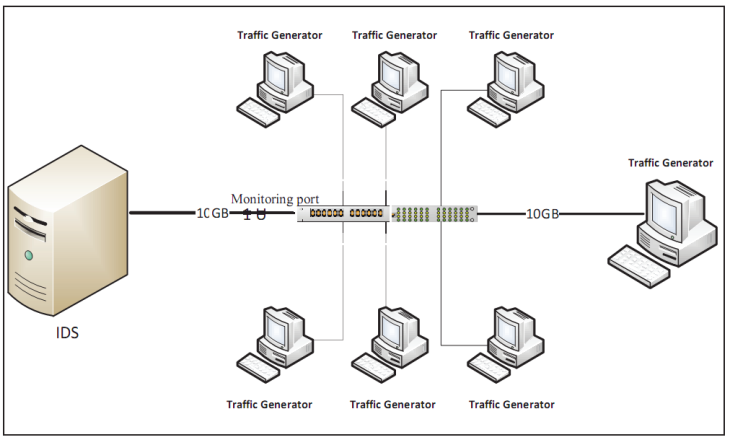


Figure 4.2: Network component specifications

Machine Type	Hardware Description	Tools used
Windows SP2	Dell Precsion, T3400,Intel Quad-core , Q6600,2GB Ram , 1Gbps network card	LAN Traffic Generator
FreeBSD Linux 2.6	Dell Precsion, T3400,Intel Quad-core , Q6600,2GB Ram , 10Gbps network card	Suricata, Snort Bandwidth monitor
ESXi SERVER	Dell Precsion, T3400,Intel Quad-core , Q6600,4GB Ram , 1Gb network card (for monitoring server), 10Gb for IDS	VMware ESXi Hypervisor Linux 2.6 Suricata, Snort Bandwidth monitor
Attacker	Dell Precsion, T3400,Intel Quad-core , Q6600,2GB Ram , 1Gbps network card	Backtrack Linux Metasploite 3 Framework
Network Switch	ProCurve series 2900	

Test scenarios were designed to test the performance of Suricata and Snort on different operating systems. Both IDS were subject to the same tests and under the exact same conditions. In order to get more accurate results, all scenarios were tested with packet sizes (1470, 1024, 512) for both TCP and UDP. The test was performed for the speed ranging from 250Mbps to 2.0Gbps. In all the scenarios Suricata and Snort were configured to load and run similar number of rules to monitor.

4.2.2 A performance analysis of Snort and Suricata

Recently, there has been shift to multi-core processors and consequently multi-threaded application design. Suricata is a multithreaded open source NIDPS, being developed via the Open Information Security Forum (OISF). Day et al. [36] describes an experiment, comprising of a series of innovative tests to establish whether Suricata shows an increase in accuracy and system performance over the de facto standard, single threaded NIDPS Snort.
Figure 4.3 [8], illustrates some of the metrics that constitute capacity.

Figure 4.3: Metrics of Capacity

Test Metrics	Resources Used
Packets per Second	CPU Cycles, network interface bandwidth, memory bus bandwidth.
Bytes per second (average packet size)	CPU Cycles, network interface bandwidth, memory bus bandwidth.
Protocol Mix	CPU cycles and memory bus bandwidth.
Number of unique hosts	Memory size, CPU cycles, memory bus bandwidth.
Number of new connections per second	CPU cycles and memory bus bandwidth.
Number of concurrent connections	Memory size, CPU cycles, memory bus bandwidth.
Alarms per second	Memory size, CPU cycles, memory bus bandwidth.

The test-bed was setup in a virtual environment, facilitating experiment portability and security. It also allowed for faster experiment initialisation. This was necessary for frequent repetition and re-configuration of the experiment tests. Vmware workstation 6.5 was used as the virtualisation platform, largely due to superior IO and disk performance over competitors Virtual Box and Virtual PC. Snort and Suricata were configured to run using identical rule-sets.
It was decided to capture background traffic from a busy universities web and application server. This was then merged with exploit traffic, created using the Metasploit Framework. The Metasploit Framework contains a total of 587 exploit modules, allowing attack data to be easily generated in quantity.
The capacity of a NIDPS is closely connected to the CPU capacity of the system. Thus, Snort and Suricata should be subjected to CPU impairment, to evaluate their efficiency under stressful conditions. VMware was used to allow the number of logical and physical cores to be reduced. The cores themselves were stressed by generating threads, causing an adjustable and measureable

[8] informed by Hall and Wiley "Capacity Verification for High Speed Network Intrusion Detection Systems"

workload. This was performed using the application cpulimit, which generates configurable workloads across the processor, allowing for the total amount of stress applied by each thread, to be limited by a percentage of the CPU capacity.
The following resources were monitored: CPU utilisation, memory utilisation, persistent storage bandwidth and network interface bandwidth. This was performed using the Linux command line utility dstat.

4.2.3 Evaluating intrusion detection systems in high speed networks

Alserhani et al. [37] they have focused on signature-based IDS with an emphasis on evaluating their performance in high-speed traffic conditions. They have selected Snort as a test platform because of its popularity and status as a de facto IDS standard.

The test bench setup is as follows: The network is composed of six machines using ProCurve Series 2900 switch as shown in 4.4. The test bench comprises a number of high performance PCs running open source tools to generate background traffic, run attack signatures and monitor network performance. The hardware description of the network is shown in 4.5. Snort was also tested for its accuracy on the different operating systems (OS) platforms (Windows and Linux). The platforms were tested by injecting a mixture of heavy network traffic and scripted attacks through the Snort host. Snort.conf in its default configuration was selected for evaluation. The performance of Snort was also evaluated under the following variant conditions:

- Generating attacks from different operating system hosts.
- Varying traffic payload, protocol and attack traffic in different scenarios.
- Subjecting it to hardware constraints of virtual machine configurations.

Figure 4.4: Test Bench

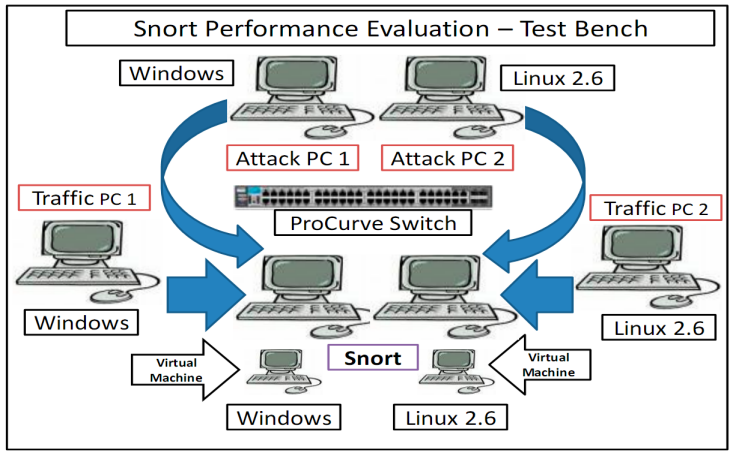


Figure 4.5: Network component specifications

Machine Type	Hardware Description	Tools Used
Network traffic/ A back ground traffic generator- PC 1 (Win SP 2)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, IGb/s RJ45, Network Card (Broadcom NetXtremo gigabit Ethernet).	NetCPS [10], Tfgen [11], Http generator [12], LAN Traffic Version 2 [13]& D-ITG V 2.6 [14].
Network traffic/ back ground traffic generator - PC 2 (Linux 2.6)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, IGb/s RJ45, Network Card (Broadcom NetXtremo gigabit Ethernet).	D-ITG V 2.6 [14] & hping V 2 [15].
Attack Machines • PC 1 – Win SP 2 • PC 2 – Linux 2.6	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, IGb/s RJ45, Network Card (Broadcom NetXtremo gigabit Ethernet).	Metaplsoit framework [16] on both platforms.
IDS Machines • Snort at Win SP 2 Platform • Snort at Linux 2.6 Platform (Snort Version 2.8.3)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, IGb/s RJ45, Network Card (Broadcom NetXtremo gigabit Ethernet).	• Snort [3] & Bandwidth Monitor [17] on Win SP 2. • Snort [3] and nload [18] (BW Monitor) on Lin 2.6.
Network Switch	ProCurve Series 2900, 10Gb/s switch with 24x1 Gb/s ports and 2x10 Gb/s 3CR17762-91-UK ports.	

4.2.4 An analysis of packet fragmentation attacks vs Snort

Fu et al. [38], Snort IDS was tested. VMware virtual machines were used as both the host and victim. Other tools were also implemented in order to generate attacks against the IDS. The experiment results show the performance of Snort IDS when it was being attacked, and the ability of Snort to detect attacks

in different ways.
This research started with the creation of a virtual network using the virtualization software VMware workstation 6.0. In order to carry on the packets fragmentation attacks experiments, three virtual machines were included in the network. One victim, one attacker, and one test machine were created in VMware workstation. The attacker generated the attacks and sent them to the victim, in order to test the Snort IDS installed on the victim. Test machine was used to record the packets sent in the network, in order to analyze and replay the packets.
A variety of tools were installed and configured in three virtual machines. The victim was equipped with sniffing tool Wireshark and intrusion detection tool Snort IDS for recording the network traffic and testing the intrusion detection capability. Testing tool Metasploit framework and scanning tool Nmap were running on the attacker for exploiting the vulnerabilities of the victim. Attack packets were generated by Scapy, which was also installed in the attacker, Tcpdump was installed in the test machine, it was used to capture and save the packets sent in the network. Tcpreplay was also carried by the test machine, for replaying the collected network traffic.

4.3 How will we test?

Searching the Internet wont give us a standard testing methodology for IDS. We base our approach on some existing suggestions Puketza et al. [39] and Nadiammai et al. [34].

We will try to cover the following performance objectives for our IDS:

- **Broad Detection Range:** for each intrusion in a broad range of known intrusions, the IDS should be able to distinguish the intrusion from normal behaviour.
- **Economy in Resource Usage:** the IDS should function without using too much system resources such as main memory, CPU time, and disk space.
- **Resilience to Stress:** the IDS should still function correctly under stressful conditions in the system, such as a very high level of computing activity.

We will design test scenarios to cover these objectives. To support the testing scenarios which cover **Broad Detection Range** we will use statistical calcu-

lations, which base our performance tests on Accuracy, Sensitivity, Specificity and computational time FAR, see section 4.3.2.

We will use these software tools:

- **Tcpreplay 4.0** Tcpreplay is a suite of BSD GPLv3 licensed tools written by Aaron Turner for UNIX (and Win32 under Cygwin) operating systems which gives you the ability to use previously captured traffic in libpcap format to test a variety of network devices. It allows you to classify traffic as client or server, rewrite Layer 2, 3 and 4 headers and finally replay the traffic back onto the network and through other devices such as switches, routers, firewalls, NIDS and IPS's We will use it to simulate an attacker. [9]

- **Ostinato** Ostinato is an open-source, cross-platform network packet crafter/traffic generator and analyzer with a friendly GUI. Craft and send packets of several streams with different protocols at different rates. Ostinato aims to be "Wireshark in Reverse" and become complementary to Wireshark. We will use it to simulate background traffic. [10]

- **Snorby 2.6.2** Snorby is an open-source ruby on rails application for networking security monitoring. It integrates with intrusion detection systems like Snort, Suricata and Sagan. [11]

Besides these tests we will have a look at Pytbull mentioned in section 4.1.1.

4.3.1 Dataset problems

Complex and new cases of intrusions, new bugs, security issues and vulnerabilities are evolving everyday for a number of reasons. Consequently, researchers in the domains of Intrusion Detection Systems and Intrusion Prevention Systems constantly design new methods to lessen the aforementioned security issues. However, getting suitable datasets for evaluating various research designs in these domains is a major challenge for the research community, vendors and data donors over the years. As a result, most intrusion detection and prevention methodologies are evaluated using wrong categories of datasets because the limitations of each category of evaluative datasets are unknown.
Nehinbe [40] list some issues regarding the use of datasets. They are:

[9] `http://tcpreplay.synfin.net/`
[10] `https://code.google.com/p/ostinato/`
[11] `https://snorby.org/`

Data privacy issues:
Data privacy that subsumes security policies, sensitivity of realistic data, risks in disclosing digital information and lack of trust are factors that do not allow realistic data to be shared among users, industries and research community. Consequently, most corporate organizations rarely disclose the lessons they learned from previous computer attacks to the research community. Thus, most research designs are not often tested with realistic problems.

Getting approval from data owner:
Getting access to some real datasets may require researchers to apply for approvals from the custodian of the datasets. Some data donors such as Cooperative Association for Internet Data Analysis (CAIDA) often require intending users to sign undertaken or Acceptable Use Policies (AUP) that contain restrictions to the time of usage and information that can be published regarding the datasets (CAIDA, 2011). In CAIDA (2011), Acceptable Use Policies is granted to registered academic, non-profit researchers, government and CAIDA members. Some data donors restrict users to different segments of the datasets. Moreover, experience shows that some approvals from the data donors can take bureaucratic processes to secure which may not happen during the time frame of the research. In other words, approvals to authorize the usage of some datasets are frequently delayed

Scope of evaluative datasets:
Intrusive datasets often vary from one network segment to another. Apart from the fact that there is variability in the patterns of computer attacks across the globe, the issue of activities that should be classified as normal and abnormal traffics are subjective in some cases. For these reasons, most publicly available datasets rapidly become obsolete, and unsuitable for making strong scientific claims.

Different research objectives:
The aims, objectives and methods of the studies are factors that also influence the choice of datasets that will be suitable for evaluating models that are designed to investigate intrusion detection and prevention problems. The NSL-KDD dataset is not suitable for investigating redundant alerts that are common problems in real networks because of the limited size of the dataset. Different researchers frequently use novel methods to investigate the same aims and objectives. As a result of this, researchers often tweak network traces in most cases to suit the objectives of their studies. By doing so, some researchers use series of data mining procedures such as data pre-processing and data cleaning to lessen the challenges in matching data with the objectives of the studies. Apart from resource utilization in terms of time and efforts, the researchers may not possess adequate knowledge necessary to enable that the new datasets become perfect

replica of the original datasets. Hence, original quality of the datasets is often lost. This is the major reason why most of the research findings in the domains of the IDSs and IPSs are very difficult to be repeated by other researchers in order to validate scientific claims.

Problem of documentations:
Most of the off-line datasets that are available for the researchers in the domains of the IDSs and the IPSs lack proper documentations. There is insufficient information about the network environment at which most of the datasets are simulated. The kinds of intrusions that are simulated, the mission of the intruders, operating systems of the attacking and destination machines, the size of the packets and other vital information that may assist analysts are not often disclosed by the data donors. Additionally, the limitations and main usage of each off-line dataset are not frequently published by the donors. Hence, many researchers tend to adapt network datasets for purposes that contravene the scope of the datasets. Another problem is that the IDS models that use the KDD 1999 and the KDD 1998 datasets that were properly labelled by the donor, recorded low performance evaluations due to the inherent flaws in the datasets. Hence, accurate interpretations of the results of evaluations conducted with the publicly available datasets are major challenges for the users.

Understanding the datasets:
Most data donors do not publish the level of success of the intruders in the datasets. Thus, high level of expertise is often required to isolate failed attacks and attacks that need countermeasures from each other whenever these categories of attacks are present in the same dataset. Hence, the efficacies of the existing intrusion aggregations are debatable because they have the tendencies to erroneously cluster failed attacks and true positives that can achieve the objectives of the attackers together

Data labellings:
Some available datasets are manually labelled datasets while some are packet traces without identities. Some trace files are background effects of some attacks collected in synthetic networks. Hence, donors such as the Shmoo group often warn users strongly about the validity of the datasets downloaded from their repository.

Availability of evaluative datasets:
Another emerging threat to the usage of Internet traces is that most trace files are not readily available for evaluating IDSs and IPSs designs without being pre-processed (Nehinbe, 2011).This is because most of the available Internet traces are tcpdump files that were logged and compressed in Packet Capture (PCAP) formats.Most IDSs such as Snort in IDS mode and Bro, and IPS such as Snortin Inlinemode are unable to decode zip files until each of the files is

correctly pre-processed into a readable format that the device can sniff.

Discrepancies in evaluative datasets:
Experiences working with some of the datasets show that they have some discrepancies due to missing attributes and values. These are usual problems when ever intrusive datasets are collected from different operating systems, different networks and different locations.Consequently, selecting a suitable method for eliminating discrepancies in intrusive datasets is a central problem in the usage of the IDSs and the IPSsfor safeguarding computer infrastructure.

4.3.1.1 Available dataset conclusion

Based on the above, we now comment on how useful the three datasets mentioned in section 4.1.2 are:

KDD Cup 1999 Dataset:
Looking at the official site [12] we can see that the last update of the dataset was in October 28, 1999. Besides this the available files are difficult to use. The data is available in simple text files, with no src/dest ip-addresses, and ports. It is not possible to replay this file to test a given IDS. If it had been a pcap file then we could have used a tool like Tcpreplay.

UNB ISCX Intrusion Detection Evaluation Dataset:
Looking at the datasets homepage [13], and reading the article regarding the creation of the dataset, it was not possible to get access to this dataset. We requested an Academic License Agreement but did not create a reaction from the vendors.

4.3.2 Statistical calculations

Nadiammai et al. [34], we base our performance tests on Accuracy, Sensitivity, Specificity and computational time FAR.

Accuracy is the total number of detected attacks among all the other attack

[12] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
[13] `http://www.iscx.ca/datasets`

data:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Sensitivity is the possibility that it is possible to predict positive instances.

$$\frac{TP}{TP + FN} \tag{4.2}$$

Specificity is the possibility that it is possible to predict negative instances.

$$\frac{TN}{TN + FP} \tag{4.3}$$

False Alarm Rate specifies the total of normal data that are mistakenly taken as attack.

$$\frac{FP}{FP + TN} \times 100 \tag{4.4}$$

Figure 4.6: Table of metrics for intrusion evaluation

Confusion Matrix (Standard Metrics)		Predicted Connection Label	
		Normal	Intrusion
Actual Connecti on Label	Normal	True Negative (TN)	False Alarm (FP)
	Intrusion	False Negative (FN)	Correctly Detected(TP)

The confusion matrix (see figure 4.6) is a visualized tool typically used in classification method and it is said to be a matching matrix in clustering method.

4.4 Evaluation

In this section we evaluate. We focus on the installation/usage experience with Snort, pytbull/snorby results, and installation/usage experience with pytbull and snorby.

4.4.1 Snort installation

The victim machine had Ubuntu 10.04 installed, and the first version of Snort which was installed were 2.8.5.2-2build1 which was from 2010-04-06. It was installed with Synaptic Package Manager. There exist 3 different categories of accessing the Snort ruleset [14]:

1. The subscriber release (payble, and has the current version of the rulesets "2955").

2. The registered user release (free of charge for 30 days, does not have the current version).

3. The Community release (is free and is updated on a daily basis, but does only contain a subset of the official ruleset)

To use the (2) we had to remove the Ubuntu version of Snort, and install the current version 2.9.5.5. When running pytbull (see section 4.4.5.1) in the beginning the detection was not good by Snort. So in order to improve the detection we searched for rulesets. We found ETOpen Ruleset [15] which is an excellent antimalware IDS/IPS ruleset. This and the community rules (3) were added to Snort, and then the detection got better (the results from pytbull showed more green and orange, see section 4.4.5.1). We had added following rulesets: default, community, snort ruleset version 2953 and ETOpen ruleset. We also tried "pulledpork" [16] to get the newest rulesets, but this action was not different from getting the rulesets by your self.

[14] `http://www.snort.org/snort-rules/?`
[15] `http://www.emergingthreats.net/open-source/etopen-ruleset/`
[16] `https://code.google.com/p/pulledpork/`

4.4.2 Snort usage

When Snort was installed you had to adjust the config file and the rulesets to fit to your environment. Searching the internet there were many suggestions for installing Snort etc. Therefore there should be Snort setup files to test if the rules was added correctly so Snort detected intrusion attempts. These tests should determine if Snort had reached its highest security level.

4.4.3 Evaluating pytbull/snorby results

It is hard to evaluate the results, because:

1. The attack signature names in Snorby does not match the test names in pytbull (no explanation of attacks, what the actual test case tests for).
2. At some test results pytbull does not show any "color" (detection result).
3. There is no explanation of why the test result has been assigned the actual detection color.
4. We don't know how many alert events pytbull should generate at snort.

By asking the head developer of pytbull we have clarified some issues.

Assignment of protection color: Each test can have a "pattern" defined in the payloads, as follows:

```
self.payloads.append([
    '{TEST_NAME}',
    'command',
    {COMMAND_ARGS_ARRAY},
    '{PATTERN}'
])
```

When pytbull receives the log from snort, it extracts the part that correspond to the test that was just performed and sees whether it matches the "pattern". In this case, the test is considered as successful and the color is green. In the case something else has been received, the color will be orange. And when nothing is received at all, the color is red.

No test color: In the case pytbull receives an alert from the IDS that does not match the expected pattern or if not pattern was specified for the test, the color is orange.

Pytbull extracting alerts from IDS: It gathers the IDS log file after each test and performs a diff of this file between 2 tests to only extract the new alerts generated by the IDS. The new alerts are considered as generated by the test.

Now we look at figures from chapter A. Looking at figure A.1 we can see that there are no red color (no detection). That is good, but some of the tests are not included because they have not been assigned a color. In figure A.2, A.3 and A.4 we have listed the detailed section of the tests which has been executed. Looking at figure A.5 we can see a list of the discovered/matched intrusions by Snort, which snorby has picked up from its database (containing logs from Snort). Snort has logged many alerts called "frag3: Fragments smaller than configured min fragment length". The signature called "Snort Alert [...]" is because the event couldn't be found in the naming files. At figure A.6 we can see how many alerts Snort has generated (High/Medium/Low severity), and a distribution of the different alert signatures. In figure A.7 we have added the High severity alerts (only one occurrence in all of the events), so it can give an impression of which alerts Snort labels as High severity. In figure A.8 we have listed the frag3 which is labelled as Medium severity. At first not all of the test modules could be executed, because some of them stopped because of error. After we fixed these errors we got the test results shown in figure A.9, A.10, A.11 and A.12 We could not execute two test modules IP Reputation and Client Side Attacks. IP Reputation had to connect to the Internet at the same time the actual computer was connected to the local network. The Client Side Attacks module did not work, even though a server instance was executed correctly on the victim computer.

4.4.4 Summing up

Based on the experiments and findings above, we briefly summarize pros and cons for Snort :
Pros: Snort is quickly installed and running on the network. Snort rules are fairly easy to write, where the syntax is straightforward. Snort has good support available [17]. Snort supports multiple formats, and last and not least, its free.

Cons: Snort rules must be developed carefully. This is necessary to reduce the number of false alarms of information generated and to reduce the amount of

[17] Snort homepage: `http://www.snort.org/`

information logged.

4.4.5 Other programs

In this section we evaluate pytbull and snorby, focusing on installation and usage.

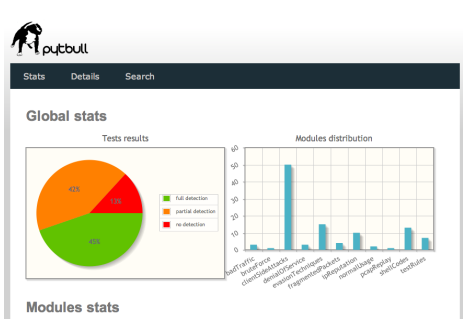
4.4.5.1 Pytbull installation

The installation of the program was straight forward, based on instructions you could easily install the client and server. The instructions can be seen on pytbulls homepage [18]. The program "ncrack" was needed to be installed on the client, so the bruteForce test module could run. The installation of the "ncrack" requires additional packages.

4.4.5.2 Pytbull usage

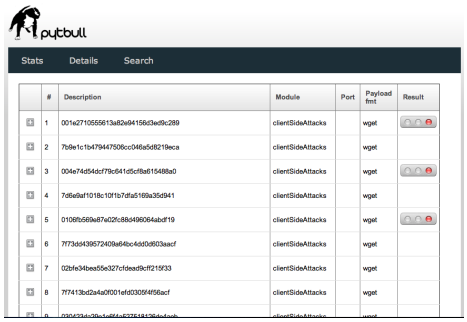
The program's usage instructions is insufficient, it contains only one example of starting the program from a terminal. So in order to get to know the available terminal arguments we looked in the pytbull code. It was easy to misunderstand the program, because it starts with a ipv6 warning, and then the menu. You could interpret this as a problem, and that the tests have not been executed. It turns out that the tests wasn't executed, it was only the initialization. Another issue was about the ftp on the server (victim machine), there was no documentation about setting users and password on the pytbull homepage. One bad thing about the pytbull program is that you could not save or export your results. Every time you run pytbull it will erase your previous results.

[18] `http://pytbull.sourceforge.net/index.php?page=documentation`

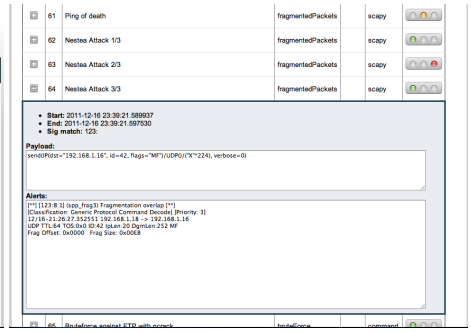


(a) Overview

(b) Diagrams



(c) Results



(d) Alert

Figure 4.7: Pictures of pytbull

4.4.5.3 Snorby installation and usage

The keyword for this installation was "dependencies" between different packages. It has a nice look, and has easy understandable menus. Some times when you use the program your login fails. This must be a bug in the program. When you delete your discovered sensors you need to reboot the computer and this is a bit annoying. Snorby has some predefined categories of signatures but it fails to classify the alerts signatures which have been found by Snort.

Chapter 5

Best practice

In this chapter we first cover measurable IDS characteristics and challenges of IDS testing. Next we list the appropriate tools to use, and suggest test procedures.

This chapter contribute, to give an advise for others who have the intention of testing an IDS.

5.1 Quantitatively measurable IDS characteristics

In this section we list a partial set of measurements that can be made on IDSs Mell et al. [41]. The focus is specifically upon those measurements that are quantitative and that relate to detection accuracy:

- **Coverage**: This measurement determines which attacks an IDS can detect under ideal conditions. For signature-based systems, this would simply consist of counting the number of signatures and mapping them to a standard naming scheme. For non signature based systems, one would

need to determine which attacks out of the set of all known attacks could be detected by a particular methodology.

- **Probability of False Alarms**: This measurement determines the rate of false positives produced by an IDS in a given environment during a particular time frame. A false positive or false alarm is an alert caused by normal non malicious background traffic.
- **Probability of Detection**: This measurement determines the rate of attacks detected correctly by an IDS in a given environment during a particular time frame. The difficulty in measuring the detection rate is that the success of an IDS is largely dependent upon the set of attacks used during the test.
- **Resistance to Attacks Directed at the IDS**: This measurement demonstrates how resistant an IDS is to an attacker's attempt to disrupt the correct operation of the IDS. Attacks against an IDS may take the form of:

  1. Sending a large amount of non attack traffic with volume exceeding the IDSs processing capability. With too much traffic to process, an IDS may drop packets and be unable to detect attacks.
  2. Sending to the IDS non attack packets that are specially crafted to trigger many signatures within the IDS, thereby overwhelming the IDSs human operator with false positives or crashing alert processing or display tools.
  3. Sending to the IDS a large number of attack packets intended to distract the IDSs human operator while the attacker instigates a real attack hidden under the smokescreen created by the multitude of other attacks.
  4. Sending to the IDS packets containing data that exploit a vulnerability within the IDS processing algorithms. Such attacks will only be successful if the IDS contains a known coding error that can be exploited by a clever attacker. Fortunately, very few IDSs have had known exploitable buffer overflows or other vulnerabilities .

- **Ability to Handle High Bandwidth Traffic**: This measurement demonstrates how well an IDS will function when presented with a large volume of traffic. Most network-based IDSs will begin to drop packets as the traffic volume increases, thereby causing the IDS to miss a percentage of the attacks.
- **Ability to Correlate Events**: This measurement demonstrates how well an IDS correlates attack events. These events may be gathered from IDSs, routers, firewalls, application logs, or a wide variety of other devices.

- **Ability to Detect Never Before Seen Attacks**: This measurement demonstrates how well an IDS can detect attacks that have not occurred before. For commercial systems, it is generally not useful to take this measurement since their signature-based technology can only detect attacks that had occurred previously.

- **Ability to Identify an Attack**:This measurement demonstrates how well an IDS can identify the attack that it has detected by labeling each attack with a common name or vulnerability name or by assigning the attack to a category.

- **Ability to Determine Attack Success**: This measurement demonstrates if the IDS can determine the success of attacks from remote sites that give the attacker higher- level privileges on the attacked system. In current network environments, many remote privilege- gaining attacks (or probes) fail and do not damage the system attacked.

- **Capacity Verification for NIDS**: The NIDS demands higher- level protocol awareness than other network devices such as switches and routers; it has the ability of inspection into the deeper level of network packets.

- **Other Measurements**: There are other measurements, such as ease of use, ease of maintenance, deployments issues, resource requirements, availability and quality of support etc. These measurements are not directly related to the IDS performance but may be more significant in many commercial situations.

5.2 Challenges of IDS testing

There are several aspects of IDSs that make IDS testing challenging Mell et al. [41].

- **Difficulties in collecting attack scripts and victim software**: One problem that has inhibited progress in this field is the difficulty of collecting attack scripts and victim software. It is difficult and expensive to collect a large number of attack scripts. While such scripts are widely available on the Internet, it takes time to find relevant scripts to a particular testing environment. Once a script is identified, it takes roughly one person week to review the code, test the exploit, determine where the attack leaves evidence, automate the attack, and integrate it into a testing environment.

- **Differing requirements for testing signature based vs. anomaly based IDSs**: Although most commercial IDSs are signature based, many research systems are anomaly-based, and it would be ideal if an IDS testing methodology would work for both of them. This is especially important since we would like to compare the performance of upcoming research systems to existing commercial ones.

- **Differing requirements for testing network based vs. host based IDSs**: Testing host based IDSs presents some difficulties not present when testing network based IDSs. In particular, network based IDSs can be tested in an offline manner by creating a log file containing TCP traffic and then replaying that traffic to IDSs.This is convenient as all of the IDSs do not have to be tested at the same time, and the repeatability of the test is easy to achieve. Alternately, host based IDSs use a variety of system inputs in order to determine whether or not a system is under attack. This set of inputs changes between IDSs. Also, host based IDSs are designed to monitor a host as opposed to a single data feed (like network based IDSs). This makes it difficult to replay activity from log files in order to test a host based IDS. Since it is difficult to test a host based IDS in an offline manner, researchers must explore more difficult realtime testing. Realtime testing presents problems of repeatability and consistency between runs.

- **Four approaches to using background traffic in IDS tests**: Most IDS testing approaches can be classified in one of four categories with regard to their use of background traffic: testing using no background traffic/logs, testing using real traffic/logs, testing using sanitized traffic/logs, and testing using simulated traffic/logs. While there may be other valid approaches, most researchers find it necessary to choose among these categories when designing their experiments. Furthermore, it is not yet clear which approach is the most effective for testing IDSs since each has unique advantages and disadvantages.

    1. **Testing using no background traffic/logs**
    2. **Testing using real traffic/logs**
    3. **Testing using sanitized traffic/logs**
    4. **Testing by generating traffic on a testbed network**

5.3 Appropriate tools to use

This section highlight, the tools which can be used, when testing an IDS.

5.3.1 Generating attacks

Here we list suggested available tools, which are able to generate attacks:

1. **BackTrack**: is a distribution based on the Ubuntu Linux distribution aimed at digital forensics and penetration testing use.
   BackTrack arranges tools into 12 categories: Information gathering, Vulnerability assessment, Exploitation tools, Privilege escalation, Maintaining access, Reverse engineering, RFID tools, Stress testing Forensics, Reporting tools, Services, Miscellaneous. [1]

2. **Kali Linux**: is a Debian-derived Linux distribution designed for digital forensics and penetration testing. It is maintained and funded by Offensive Security Ltd. developed by rewriting BackTrack, their previous forensics Linux distribution. [2]

3. **Metasploit**: is a computer security project that provides information about security vulnerabilities and aids in penetration testing and IDS signature development. [3]

4. **Pytbull**: is an Intrusion Detection/Prevention System (IDS/IPS) Testing Framework for Snort, Suricata and any IDS/IPS that generates an alert file. It can be used to test the detection and blocking capabilities of an IDS/IPS and to validate config. [4]

5.3.2 Generating background traffic

Here we list suggested available tools, which are able to generate background traffic:

1. **Ostinato** is an open-source, cross-platform network packet crafter/traffic generator and analyzer with a friendly GUI. Craft and send packets of several streams with different protocols at different rates. Ostinato aims to be "Wireshark in Reverse" and become complementary to Wireshark. [5]

[1] http://www.backtrack-linux.org/
[2] http://www.kali.org/
[3] http://www.metasploit.com/
[4] http://pytbull.sourceforge.net/index.php?page=home
[5] https://code.google.com/p/ostinato/

Looking at Botta et al. [42] there exist several generation platforms such as: Seagull, Timix, Rude/Crude, TG, Mgen, Kute, Brute, LiTGen, Network traffic generator, NetSpec, Netperf, Iperf, TCPivo, TCPreplay, TCPopera, ParaSynTG, UniLoG, Swing and Mace.

5.4 Suggested procedures

In this section we give a final procedure suggestion in order to give a best practice, when testing an IDS.
Firstly we sum up, the articles from section 4.2:

Alhomoud et al. [35] from section 4.2.1
Setup/environment: Use of physical hardware.
Metrics: Test of performance: different packet size, different speed,

Day et al. [36] from section 4.2.2
Setup/environment: Use of virtual environment.
Metrics: Evaluation of efficiency under stressful conditions. Following resources were monitored: CPU utilisation, memory utilisation, persistent storage bandwidth and network interface bandwidth

Alserhani et al. [37] from section 4.2.3
Setup/environment: Use of physical hardware.
Metrics: Test of performance.

Fu et al. [38] from section 4.2.4
Setup/environment: Use of virtual environment.
Metrics: Tested with different types of attacks, transmission rates, and various packet fragment sizes.

Based on the four articles which test Snort and the rest of this thesis, we can suggest the following check-list:

1: Determine the type of the target IDS.
We know that an IDS can only be passive, and not active, because then it would be an IPS. Another thing is that the IDS can also be either network-based or host-based, and knowledge-based or signature-based. For a more detailed overview you can look at figure 2.21.

2: Determine the testing setup/environment.
As seen in the four selected articles, there is no specific choice, if it should

be psychical hardware or virtual environment. It depends on, if there exist a budget or not which can give you the right resources for testing. It also depends on which metrics you decide to test for. Money or not and selected metrics's we will suggest a virtual environment, which hopefully can give you a realistic scenario. Some benefits are: Isolation, Standardization, Consolidation, Ease of Testing and Mobility [6]. Some drawbacks are: Virtual machine is not that efficient as a real one when accessing the hardware. When multiple virtual machines are simultaneously running on a host computer, each virtual machine may introduce an unstable performance, which depends on the workload on the system by other running virtual machines. [7]

3: Determine the testing/evaluation metrics.
This depends on the two previous points, and you can choice to re-examine metrics's which already has been tested/evaluated.

[6] http://www.devx.com/vmspecialreport/Article/30383
[7] http://www.serial-server.net/virtual-machine/

Chapter 6

Conclusion

In this project we wanted to look at IDS in an critical perspective, and the creation of this thesis has not convinced us, to be less critically regarding IDS.

The objective of this thesis was initially to select open source IDS, and find their limitations, pros/cons, and see how they complement each other.

During the project we discovered that our first plan, involving test with a dataset (KDD cup 1999), was not possible to fulfil. Some of the reasons for this outcome, were that the dataset had many shortcomings, it was not available as a replay file, and the data was not complete. When turning to other datasets, it showed that the availability of these datasets was lacking. Most of the datasets were not public available, not particular relevant with attacks, the project within the dataset was created was stopped, no documentation and numbers for amount of certain network traffic so you could make statistical calculations.

The main outcome of this thesis was later reconsidered, and the new direction was therefore to give an best practice of testing IDS.

The main contributions of this thesis are:

1. Proposed an overview of recent IDS techniques approaches, listing of attacks and threats, and explanation of challenges regarding IDS.

2. Explained the typical architecture of IDS, and used Snort as an example.
3. Explained how to evaluate IDS.
4. Proposed a best practice for testing an IDS.
5. Given a concluding remark about the role of NIDS, answers to the research questions, and provided suggestions and improvements.

We know that threats and attacks against the users computer system evolve rapidly every day, but the focus should be mixed. None of the viewed articles in this thesis look at attacks against the IDS itself. This issue just show that security in general is very complex, and the goal of a "secure" computer system is very hard.

Most of the articles which were deselected, proposed new methods but failed to prove that it actual worked. Besides that, some critic should be pointed at the used articles in this thesis. Not all of the articles are from the period 2012-2013, so their arguments could be outdated, because of new intrusions which evolve every day.

When working with rule based IDS, it turns out that the more rules the better protection, but is it really possible to check for everything, one could doubt this. In addition, testing an IDS has shown to be almost impossible because of the lack of available datasets and standard test procedures.

6.1 The role of NIDS

Network security is often a primary concern when building a network infrastructure. Security management for networks is different for all kinds of situations. A home or small office may only require basic security while large businesses may require high-maintenance and advanced software and hardware to prevent malicious attacks from hacking and spamming.
By reading the article McHugh et al. [43], we can use some of its main points to support our discussion. First of all, we encounter the phrase "Defense in Depth", and the definition is:

"Defense in Depth is a strategy used by many corporations to maintain security. It is used to help prevent attackers from getting into the network by putting up multiple barriers around the network to slow down the attack. This strategy was developed by National Security Agent or NSA to help with security. This strategy would be using another idea called layered security, which would be using

Firewalls and other associated technologies to mitigate and prevent an attack. Depending on the technology these technologies would be used to defend against, malware, DDOS, spoofing, intruders, and many other types of attacks on a system. This also includes a plan on what would happen if an attack were to occur, and what the corporation should do in this type of event. However, Defense in Depth does not only apply to corporation it also applies to every day users. Typical users can also have a Defense in Depth strategy just in case something happens on their network or computer system." [1]

Figure 6.1: Defense in depth layers

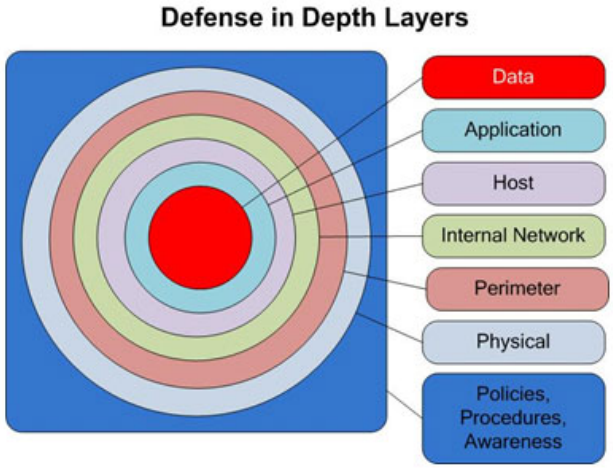


Figure 6.1 shows the different layers of Defense in Depth. NIDS can be deployed in one of the layers, and therefore complement other security measures. It should be considered as a 2nd line of defence, and a burglar alarm, which notifies the administrator.
When adding a NIDS to your network infrastructure it is not just a plug and play, where it works immediately as intended. No, it needs the right configuration and placement. Regarding placement the webpage [2] suggest three possible locations. The first is where the NIDS is outside the perimeter of the firewall, see figure 6.2. The next is where the NIDS is deployed such that it monitors the traffic that traverses any given link within the network, see figure 6.3. The last suggestion is where the NIDS is installed in every host, just like anti-virus. So every host has an inbuilt NIDS attached to all of its network interfaces.

[1] cited from `http://www.personal.psu.edu/dhl5025/Assignment6.html`
[2] `http://www.cse.wustl.edu/~jain/cse571-07/ftp/ids/`

Figure 6.2: A NIDS as an early detection system

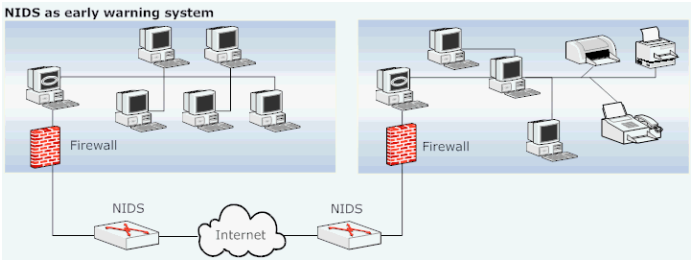
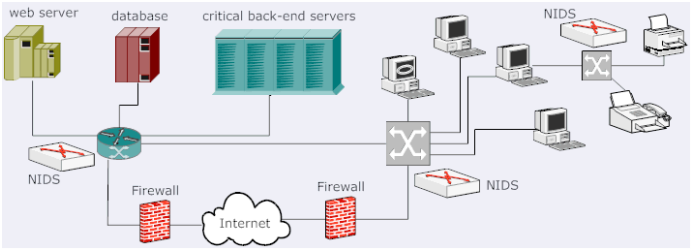


Figure 6.3: NIDS in complete deployment mode



When deploying an NIDS it requires a broad understanding of computer security. Besides that the use of technology alone is not sufficient to maintain network security. An organization must attract, train, and retain qualified technical staff to operate and maintain intrusion detection technologies. In today's market, qualified intrusion analysts and system/network administrators who are knowledgeable about and experienced in computer security are hard to find.

There will always be pros and cons when making a decision, and regarding the deployment of NIDS it is no exception. You must determine that the pros is weighted higher than the cons.

6.2 Answers to research questions

What is an IDS? - what is the typical architecture?

It is a device or a software application that monitors network or system activities for malicious activities or policy violations and produces reports to, for instance an administrator. As explained in section 3.1 about CIDF, it consist of four components: Event generators, Event analyzers, Event databases and Response units. These components serve as the base for modern IDS, and can be extended to fit the actual implementation.

What sort of techniques does the IDS use?
As explained in section 2.3, which consist of recent approaches within IDS, the techniques used has been categorized in this thesis. The categories are: Data Mining, Machine Learning, Hidden Markov Models, Honeypot, Genetic Algorithm and Fuzzy Logic.

How is the patterns represented and detected?
Looking at Snort as an example, an FSM is generated from the set of strings extracted from the Snort rule database. The FSM matches multiple strings at the same time based on the Aho-Corasick string matching algorithm. The single-keyword and multiple-keyword pattern matching algorithms which an IDS can use can be seen in section 3.3. Even though many IDS use multiple-keyword pattern matching it has been decided that it is properly not sufficient to use. When the workload is very high because of the inspection of every network packets, another possibility is to use hardware for the pattern matching.

What is the common test approach for IDS?
As summarized in section 5.4, about the articles regarding testing Snort, it shows that the choice of test environment is either physical hardware or virtualization. It depends on, the available resources, such as money.
Regarding evaluation metrics, they are adapted to the specific case, where the researchers use there own evaluation criteria to check the actual requirements of the new theory or extending existing IDS with new functionality.
So currently, there do not exist a common test approach for IDS.

Does an IDS cover all potential intrusions?
We have looked at NIDS, and my opinion is that it is not possible to detect all potential intrusions. For instance the placement of the NIDS determines the role, and what its purpose is. It depends on the circumstances, and the NIDS can't fully detect zero-day exploits. Another weakness of the NIDS is when the network speed increases, the NIDS might discard/drop packets.

What is the future prospects of IDS?
Its a question that is hard to answer, because non of the used articles explains this.

"Is it possible to make an trustworthy investigation of an Intrusion Detection System which finds its limitations?"

Answer: No, but if you have money and therefore the required resources it might be possible. In addition, specific researchers might have a private dataset available, which represent a realistic network scenario.

"Is it possible to make an trustworthy best practice for testing Intrusion Detection System?"

Answer: In this thesis we have tried to give an best practice, and the focus has been on, giving some guidelines on what to decide, when testing an IDS. The best practice has been based on the knowledge we have gained by the articles in this thesis.

6.3 Suggestions and improvements

When reading the relevant articles for this thesis, and by using tools, we have given it a thought, and thereby listed some suggestions and improvements:

Central IDS web community
: It was very annoying that relevant articles about IDS were hard to find, where some was found by luck. It could be very helpful and informative to have a community on the web, which contained categorized relevant articles about IDS, the state of the art IDS information, and for instance links or reviews about how to test IDS.

De facto standard (shared) dataset for testing IDS
: As mentioned in earlier section it was not possible to get a relevant and useful dataset for testing Snort for its limitations. It could be nice if someone could create a dataset which were accessible for every relevant researchers, and for the public. People could therefore contribute to this dataset, and get closer to handle all of the modern intrusions/attacks.

Runnable security check of Snort
: One could be in doubt, whether Snort was configured the right way, and therefore had the correct security level. It could be nice if the vendors of Snort, had made a security check script that an user could run. The script could check for specific intrusions that as a standard the Snort program should detect.

Improvements for Pytbull
: There should be an included file or on the homepage, about how and what it tests for. The next version of Pytbull is under development, and it should have the functionality of exporting or save previous results.

GUI for Snort Rules
: Currently the rules for Snort is placed in regular text files. It could be

convenient to have the rules placed in a database. In addition a GUI could be used to get an overview of the rules, and maybe create new ones or modify existing ones.

Documented test of Snort

As a student which is interested in a more detailed explanation of the Snort architecture, it could be helpful to have documentation of this on the Snort webpage. Besides that it could be good if the vendors of Snort had made documentation of testing Snort, so it could convince the users, that it really could help detecting intrusions.

Another issue related to the documentation is that there are many potential customers for the results of quantitative evaluations of IDS accuracy. Acquisition managers need such information to improve the process of system selection, which is too often based only on the claims of the vendors and limited-scope reviews in trade magazines. Security analysts who review the output of IDSs would like to know the likelihood that alerts will result when particular kinds of attacks are initiated. Finally, R and D program managers need to understand the strengths and weaknesses of currently available systems, so that they can effectively focus research efforts on improving systems, and measure their progress.

A standard for test methodology for IDS

It would be helpful for the researches whose research field is IDS, to have a standard test methodology which they could follow. The standard could just be a base for own extending procedures.

A standard for evaluation metrics for IDS

It would also be helpful for the researches for IDS to have a standard for evaluation metrics for IDS. In this way, they could have specific performance requirements that the current IDS should comply with, when for instance extending with new functionality.

Appendix A

Pytbull and snorby results

Figure A.1: Report from pytbull

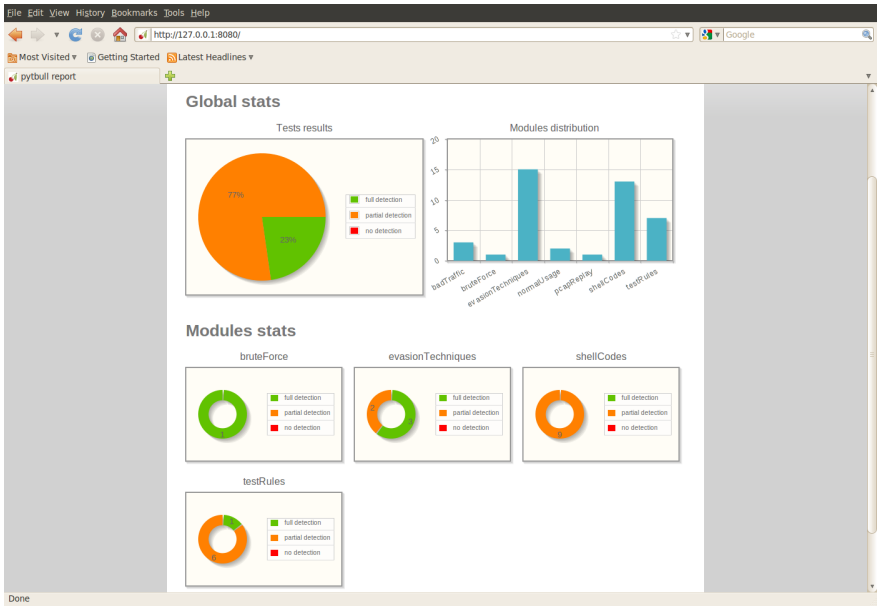


Figure A.2: Details from pytbull

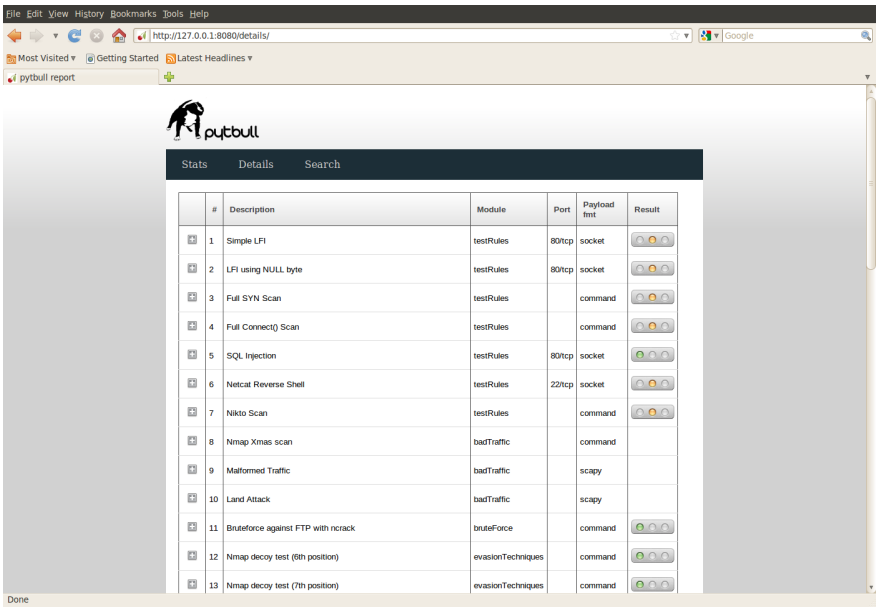


Figure A.3: Details from pytbull

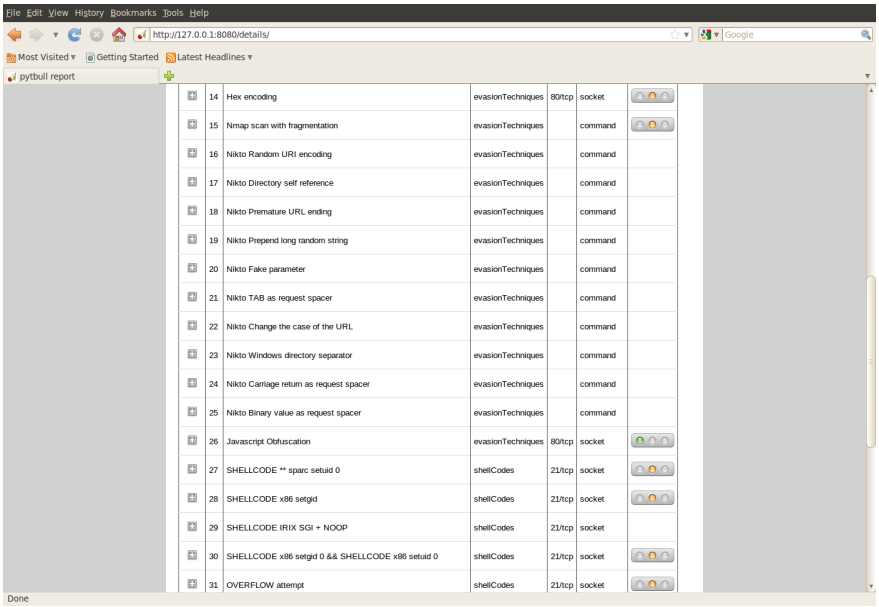


Figure A.4: Details from pytbull

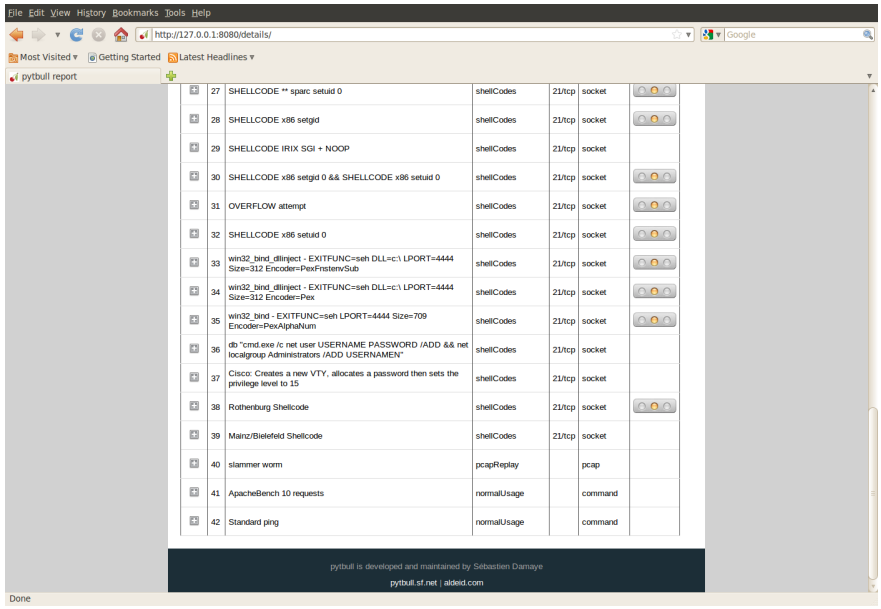


Figure A.5: Signature results from snorby

Top 15 Signatures

Signature Name	Percentage	Event Count
frag3: Fragments smaller than configured min_fragment_length	83.29%	2198
Snort Alert [1:1228:0]	4.32%	114
GPL SHELLCODE x86 inc ebx NOOP	4.24%	112
ET POLICY FTP Login Successful	1.63%	43
ET POLICY Suspicious inbound to mySQL port 3306	1.17%	31
ET SCAN Potential VNC Scan 5900-5920	0.87%	23
PROTOCOL-SNMP AgentX/tcp request	0.68%	18
ET SCAN Potential VNC Scan 5800-5820	0.61%	16
ET POLICY Suspicious inbound to Oracle SQL port 1521	0.57%	15
ftp_pp: Telnet command on FTP command channel	0.53%	14
Snort Alert [1:1748:0]	0.34%	9
ET SCAN Nmap Scripting Engine User-Agent Detected (Nmap Script...	0.34%	9
ET SCAN Potential FTP Brute-Force attempt	0.27%	7
PROTOCOL-SNMP request tcp	0.23%	6
ET POLICY Suspicious inbound to PostgreSQL port 5432	0.23%	6
ET POLICY Suspicious inbound to MSSQL port 1433	0.15%	4
ET SCAN Potential SSH Scan	0.15%	4
ET SCAN Potential SSH Scan OUTBOUND	0.15%	4
SERVER-WEBAPP robots.txt access	0.11%	3
ET POLICY Unsupported/Fake FireFox Version 1.	0.11%	3

Figure A.6: Graph from snorby

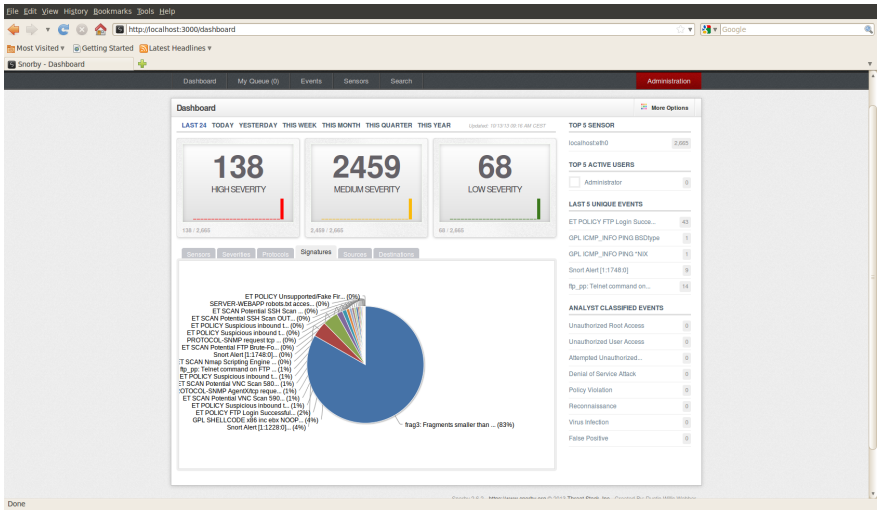


Figure A.7: Highest classification of events from snorby

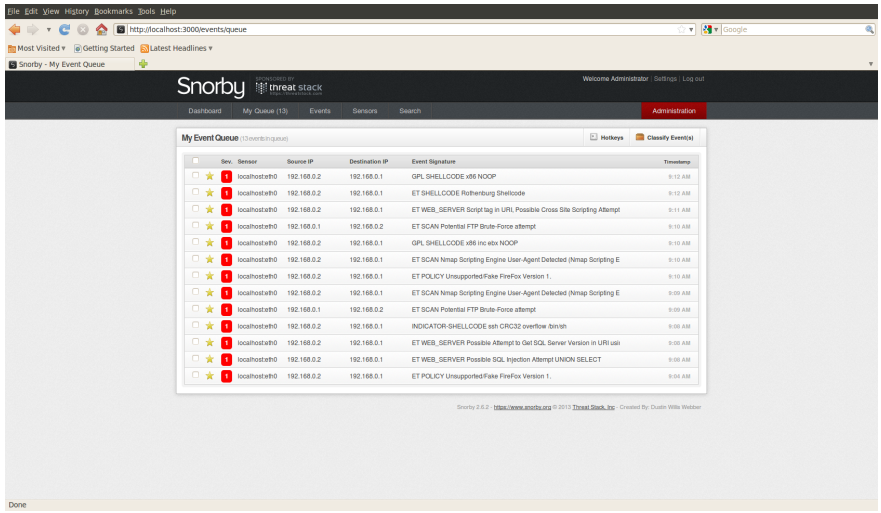


Figure A.8: Middle classification of events from snorby

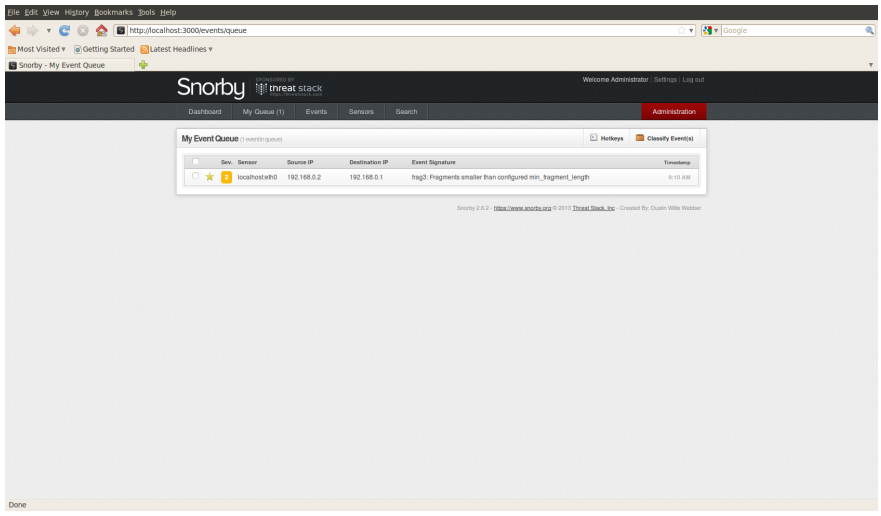


Figure A.9: Report from pytbull

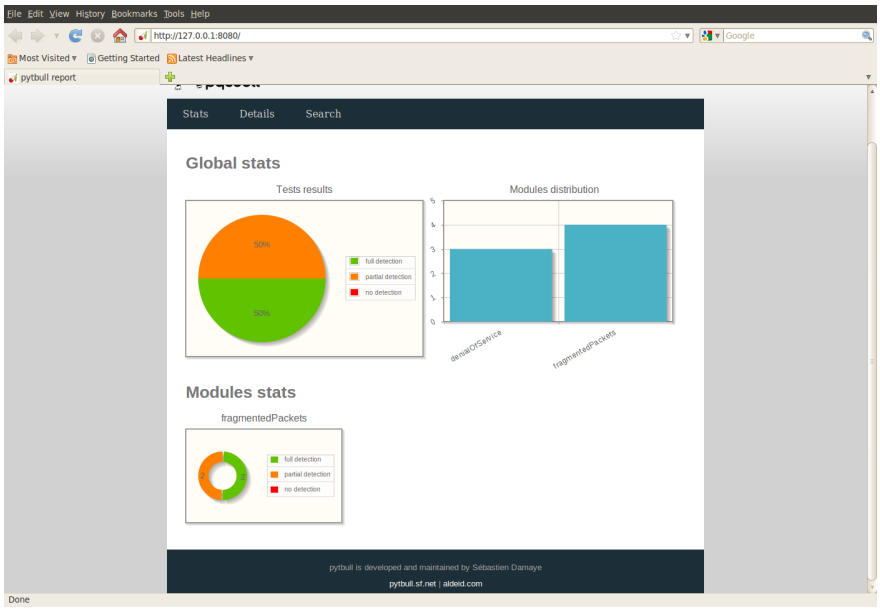


Figure A.10: Details from pytbull

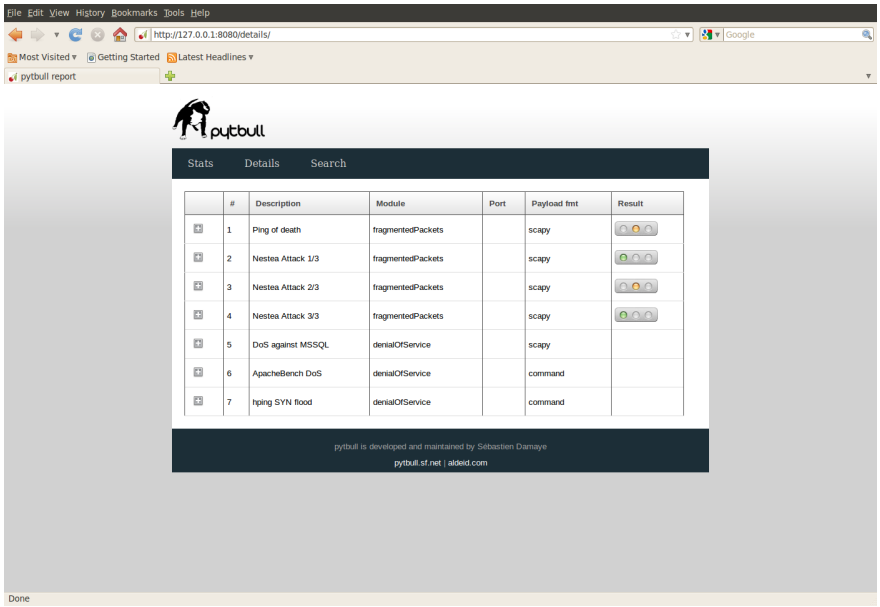
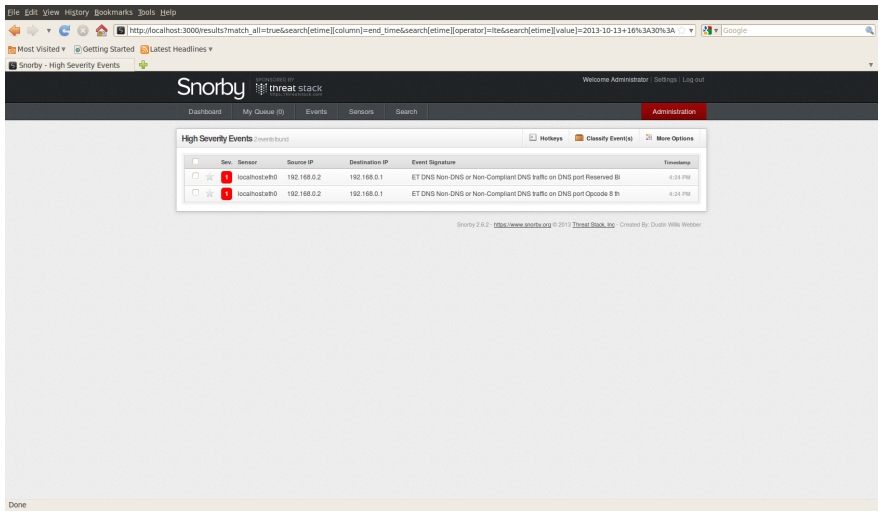


Figure A.11: Signature results from snorby

Top 15 Signatures

Signature Name	Percentage	Event Count
stream5: Data on SYN packet	98.04%	1000
ET POLICY FTP Login Successful	0.78%	8
ET POLICY Suspicious inbound to MSSQL port 1433	0.49%	5
Snort Alert [1:499:0]	0.2%	2
ET DNS Non-DNS or Non-Compliant DNS traffic on DNS port Reserv...	0.1%	1
ET DNS Non-DNS or Non-Compliant DNS traffic on DNS port Opcode...	0.1%	1
frag3: Fragmentation overlap	0.1%	1
frag3: Short fragment, possible DoS attempt	0.1%	1
ET SCAN Behavioral Unusual Port 1433 traffic, Potential Scan o...	0.1%	1

Figure A.12: Middle classification of events from snorby



Bibliography

[1] Hackmageddon.com, "Cyber attacks statistics," http://hackmageddon.com/2013-cyber-attacks-statistics/, Aug 2013.

[2] PWC, "Cybercrime protecting against the growing threat," *unknown*, 2004.

[3] I. Corporation, "Ibm qradar security intelligence," 2013.

[4] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers and Security*, 2004.

[5] A. J. Deepa and D. V. Kavitha, "A comprehensive survey on approaches to intrusion detection system," in *Procedia Engineering*, 2012.

[6] W. Stallings, "Introduction to network-based intrusion detection," http://www.informit.com/articles/article.aspx?p=782118, Aug 2007.

[7] M. D'silva and D. Vora, "Comperative study of data mining techniques enhance intrusion detection," *International journal of engineering research and applications*, 2013.

[8] Q. Zhou and Y. Zhao, "The design and implementation of ids based on data mining technology," *Research journal of applied sciences engineering and technology*, 2013.

[9] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," *unknown journal*.

[10] P. Natesan, P. Balasubramanie, and G. Gowrison, "Improving attack detection rate in network intrusion detection using adaboost algorithm with

mutiple weak classifiers," *Journal of information and computational science*, 2012.

[11] D. Ariu, R. Tronci, and G. Giacinto, "Hmmpayl: An intrusion detection system based on hidden markov models," *Computers and Security*, 2010.

[12] H. Farhadi, M. AmirHaeri, and M. Khansari, "Alert correlation and prediction using data mining and hmm," *Information Security*, 2011.

[13] V. S. Bhumika, "Use of honeypots to increase awareness regarding network security," *IJRTE*, 2012.

[14] Wikipedia, "Genetic algorihm," http://en.wikipedia.org/wiki/Genetic_algorithm, Aug 2013.

[15] B. S. Dhak and S. Lade, "An evolutionary approach to intrusion detection system using genetic algorithm," *ijetae*, 2012.

[16] Wikipedia, "Fuzzy logic," http://en.wikipedia.org/wiki/Fuzzy_logic, Aug 2013.

[17] R. Shanmugavadivu and D. N. Nagarajan, "Network intrusion detection system using fuzzy logic," *IJCSE*.

[18] U. S. G. A. Office, "United states faces challenges in addressing global cybersecurity and governance," *unknown journal*, 2010.

[19] FireEye, "Advanced targeted attacks," *unknown journal*.

[20] O. Kolesnikov and W. Lee, "Advanced polymorphic worms: evading ids by blending in with normal traffic," 2004.

[21] S. Paul and B. K. Mishra, "Polys: Network-based signature generation for zero-day polymorphic worms," *International journal of grid and distributed computing, vol.6, no. 4*, 2013.

[22] P. Li, M. Salour, and X. Su, "A survey of internet worm detection and containment," *1st Quarter 2008, Volume 10, No. 1*, 2008.

[23] M. M. M. Hassan, "Current studies on intrusion detection system, genetic algorithm and fuzzy logic," *International Journal of Distributed and parrallel Systems*, 2013.

[24] J. Pieprzyk, T. Hardjono, and J. Seberry, "Fundamentals of computer security," http://books.google.dk/books?id=NH-m8L0R3rYC&printsec=frontcover&hl=da#v=onepage&q=CIDF&f=false, 2003.

[25] unknown author, "White paper on deep packet inspection," *unknown journal*.

[26] T. AbuHmed, A. Mohaisen, and D. Nyang, "Deep packet inspection for intrusion detection systems a survey," *unknown journal*.

[27] J. ichi Aoe, "Computer algorithms string pattern matching strategies," *unknown journal*, 1994.

[28] S. Hasib, M. Motwani, and A. Saxena, "Importance of aho corasick string matching algorithm in real world applications," *IJCSIT*, 2013.

[29] Z. A. Khan and R. K. Pateriya, "Mutiple pattern string matching methodologies a comparative analysis," *unknown journal*, 2012.

[30] Syngress, *How to cheat at configuring open source security tools*. Syngress, 2007.

[31] S. Sen, "Performance characterization and improvement of snort as an ids," Lucent Technologies, Tech. Rep., 2006.

[32] Po-ChingLin and Jia-HauLee, "Re-examining the performance bottleneck in a nids with detailed profiling," *Journal of Network and Computer Applications 36*, 2012.

[33] B. Kang, H. S. Kim, J. S. Yang, and E. G. Im, "Rule indexing for efficient intrusion detection systems," *unknown journal*, 2012.

[34] G. Nadiammai and M. Hemalatha, "An enhanced rule approach for network intrusion detection using efficient data adapted decision tree algorithm," *Jorunal of theoretical and applied information technology*, 2013.

[35] A. Alhomoud, R. Munir, J. P. Disso, I. Awan, and A. Al-Dhelaan, "Performance evaluation study of intrusion detection systems," *The 2nd international conference on ambient systems, networks and technologies*, 2011.

[36] D. J. Day and B. M. Burns, "A performance analysis of snort and suricata network intrusion detection and prevention engines," *The Fifth International Conference on Digital Society*, 2011.

[37] F. Alserhani, M. Akhlaq, I. U. Awan, J. Mellor, A. J. Cullen, and P. Mirchandani, "Evaluating intrusion detection systems in high speed networks," *Fifth international conference on information assurance and security*, 2009.

[38] T. Fu and T.-S. Chou, "An analysis of packet fragmentation attacks vs. snort intrusion detection system," *International Journal of Computer Engineering Science (IJCES)*, 2012.

[39] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A methodology for testing intrusion detection systems," *17th National computer security conference in Baltimore*, 1996.

[40] J. O. Nehinbe, "A critical evaluation of datasets for investigating idss and ipss researches," *Proceedings of the 2011 10th IEEE International Conference On Cybernetic Intelligent Systems, September 1-2, London, UK*, 2012.

[41] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An overview of issues in testing intrusion detection system," *U.S. Department of Commerce*, 2003.

[42] A. Botta, A. Dainotti, and A. Pescape, "A tool for the generation of realistic network workload for emerging networking scenarios," *Elsevier*, 2012.

[43] J. McHugh, A. Christie, and J. Allen, "The role of intrusion detection system," *IEEE Software*, 2000.