An Experimental Study of Detecting and Correlating Different Intrusions

Ratna Deepika Kannan
An Experimental Study of detecting and correlating different Intrusions

School of Computer and Electronic Engineering
University of Essex
Name: Ratna Deepika Kannan
Email: ratnadeepika89@gmail.com
Supervisor: Dr. Martin Reed
# Table of Contents

**TABLE OF CONTENTS** .............................................................................................................II  
**FOREWORD AND ACKNOWLEDGEMENTS** ...........................................................................IV  
**LIST OF FIGURES** ................................................................................................................V  
**ACRONYMS** ..........................................................................................................................VI  
**MYSQL DATABASE TERMS:** .............................................................................................VII  
**ABSTRACT** ............................................................................................................................VIII  
1 **CHAPTER 1** ......................................................................................................................1  
   1.1 **Aim** .............................................................................................................................1  
   1.2 **Software Requirements** .............................................................................................1  
   1.3 **Scope of the Project and Document Structure** ........................................................2  
2 **CHAPTER 2** ......................................................................................................................3  
   **BACKGROUND INFORMATION ON SECURITY** ........................................................3  
   2.1 **Security Goals** ..........................................................................................................3  
   2.2 **Vulnerabilities and Threats** ....................................................................................4  
   2.3 **Measures to control Threats** ..................................................................................4  
   2.4 **Failure of Security Measures** ................................................................................4  
3 **CHAPTER 3** ......................................................................................................................6  
   **INTRUSION DETECTION SYSTEM** ...............................................................................6  
   3.1 **Functions of IDS** .....................................................................................................7  
   3.2 **Need for IDS** ............................................................................................................7  
   3.3 **Common Intrusion Detection Framework** ..............................................................7  
   3.4 **Classification of IDS** .............................................................................................8  
   3.5 **Challenges faced by IDS** .......................................................................................11  
   3.6 **Placing an IDS** .........................................................................................................12  
4 **CHAPTER 4** ......................................................................................................................14  
   **INTRUSION DETECTION USING SNORT** .................................................................14  
   4.1 **Components of SNORT** ..........................................................................................14  
   4.2 **SNORT Modes** ........................................................................................................16  
   4.3 **Experimental Setup** ...............................................................................................16  
      4.3.1 **Installing supporting packages on Operating System** ......................................16  
      4.3.2 **MySQL Installation** .......................................................................................17
5 CHAPTER 5.............................. 21

ALERT CORRELATION TECHNIQUES.............................. 21

5.1 Three-level Correlation Model .......................................................... 22
5.1.1 Contents of iphdr ................................................................. 23
5.1.2 First Level Correlation ........................................................... 24
5.1.3 Second Level Correlation ......................................................... 26
5.1.4 Third Level Correlation .......................................................... 28
5.1.5 Result ......................................................................................... 29

5.2 Correlation based on Signature and Timestamp Examination ................. 29
5.2.1 Correlation of Signature 32 ...................................................... 32
5.2.2 Correlation of Signature 43 ...................................................... 32
5.2.3 Result ......................................................................................... 33

5.3 Correlation based on ICMP Type and Code ........................................... 33
5.3.1 Correlation Process ................................................................. 35
5.3.2 Result ......................................................................................... 37

6 CHAPTER 6.............................. 38

CONCLUSION AND FUTURE WORK ........................................ 38

6.1 Conclusion ....................................................................................... 38
6.2 Future Work and Recommendations ................................................ 39

REFERENCES ................................................................. 41

APPENDIX ................................................................. 45
Foreword and Acknowledgements

I would like to thank Dr. Martin Reed, my supervisor for his guidance and help. It is worth mentioning that this project would not have been successful without his excellent supervision. I would also like to extend my gratefulness to the University of Essex and to the other staffs who directly or indirectly contributed to the success of my project.

Finally, I would like to thank my family and the Almighty for supporting me throughout the journey of my life, irrespective of all circumstances.
List of Figures

Figure 1: Suspicious events that would trigger alerts from an IDS [11] [3] ........ 6
Figure 2: C IDF Model [14] [3] ........................................................................... 8
Figure 3: Classification of IDS ............................................................................. 9
Figure 4: Possible outcomes of IDS [3] .................................................................. 11
Figure 5: Placing IDS in a Network [6] ................................................................. 13
Figure 6: SNORT Components [6][12] ................................................................. 15
Figure 7: Script to start SNORT and Barnyard ..................................................... 18
Figure 8: SNORT and Barnyard activated ............................................................ 18
Figure 9: Script to stop SNORT and Barnyard .................................................... 18
Figure 10: Alerts Generated by SNORT ............................................................... 19
Figure 11: Graphical Representation of the Attacks by SNORT ......................... 20
Figure 12: Script to Start SNORT, Barnyard and to run Tracefile ....................... 20
Figure 13: SNORT Database ............................................................................... 21
Figure 14: Three-Level Correlation Model .......................................................... 22
Figure 15: Contents of table ‘iphdr’ ..................................................................... 24
Figure 16: First Level Correlation Output ............................................................ 25
Figure 17: First Level Correlation of IP Address vs. Alerts ................................... 26
Figure 18: Second Level Correlation Output ....................................................... 27
Figure 19: Second Level Correlation of IP Address vs. Alerts ............................... 28
Figure 20: Third Level Correlation Output ........................................................... 28
Figure 21: Final Output of the Three-level Correlation Model .............................. 29
Figure 22: Contents of the table ‘event’ ............................................................... 30
Figure 23: IP Address and Signatures .................................................................. 31
Figure 24: Signature 32 separated ........................................................................ 32
Figure 25: Signature 43 Separated ....................................................................... 33
Figure 26: ICMP Packet Format [37] [38] ............................................................. 35
Figure 27: icmp table ......................................................................................... 36
Figure 28: ICMP Packets that must be blocked [40] ............................................. 36
Figure 29: ICMP Type 0 Code 0 Correlation ....................................................... 37
Acronyms

IDS: Intrusion Detection System
IPS: Intrusion Prevention System
CIDF: Common Intrusion Detection Framework
IDMEF: Intrusion Detection Message Exchange Format
IDXP: Intrusion Detection Exchange Protocol
IDIP: Intrusion Detection and Isolation Protocol
SASL: Simple Authentication and Security Layer
HIDS: Host Intrusion Detection System
NIDS: Network Intrusion Detection System
DoS: Denial of Service Attack
DDoS: Distributed Denial of Service Attack
PHP: Pretty Home Page
ICMP: Internet Control Message Protocol
POD: Ping of Death Attack
MySQL Database Terms:

Sid: SNORT ID

cid: Common ID

sig_id: Signature ID

ip_src: Source IP address field

ip_dst: Destination IP address field

ip_ver: IP version field

ip_hlen: IP header length field

ip_tos: IP Type of Service field

ip_len: IP packet length field

ip_flags: IP Flags field

ip_id: IP Identification field

ip_off: IP Offset field

ip_ttl: IP packet Time to live field

ip_proto: IP Protocol field

ip_csum: IP checksum field

icmp_type: Type of ICMP

icmp_code: Code of ICMP

icmp_csum: ICMP Checksum

icmp_id: ICMP Identifier

icmp_seq: ICMP Sequence Number

sig_id: Signature Identification number

sig_name: Description of the signature

sig_priority: Signature priority
Abstract

An Intrusion Detection System (IDS) has become a dominant security tool because of its ability to alert the System Administrator in the case of an intrusion. However, this simplicity in its functionality comes with a disadvantage. An IDS is highly sensitive and tends to report thousands of false positives (fake alerts). As a result, research has been initiated to develop the existing IDS by reducing the false positives and by increasing the ability to spot an actual intrusion.

In this project, correlation techniques are used to reduce the false positives in an IDS. Correlation or Aggregation refers to the process of identifying similar alerts and grouping them together. The false positives are identified by several means and are separated from the real, malicious alerts. Three correlation techniques have been proposed and have been tested using the DARPA Evaluation Data Sets 1999 on SNORT IDS. These correlation techniques namely, Three-level correlation model, Correlation based on Signature and Timestamp Examination and Correlation based on ICMP Type and Code have proved to increase the final efficiency of SNORT, by reducing the false positives up to 80%.
Chapter 1

Introduction

With the ubiquitous growth of the Internet, retaining its security is a difficult task. Two decades ago, computer systems were generally not connected to the Internet or were simply a part of a small network. At present, the era of e-commerce is flourishing, where all the business transactions are carried out via the Internet. All the computer systems globally rely on the existence of the Internet. This implies that the attacks on networks and systems have grown linearly with the growth of the Internet [9].

Due to the lack of necessary security measures, opportunities for exploitations arise. It is now possible to launch an attack on the network of an organization by remotely operating from any part of the world. Loopholes in the security of the Internet are searched for and if found, this discovery could result in anything ranging from a simple misuse of privacy to a mass destruction like bank robberies and terrorism. In simple terms, the search conducted to unearth a loophole is known as an intrusion and it is the duty of the system and network administrators to detect these intrusions.

1.1 Aim

An intrusion detection system (IDS), which is one of the most eminent security tools in detecting an intrusion, will be implemented in a vulnerable environment to aid in the process of logging valuable information regarding the intrusions. Even though the IDS can be utilised for intrusion detections, its efficiency cannot be graded as high due to the large number of alerts [1] [2] [3].

It is essential for the companies and the organisations deploying these intrusion detection systems to examine all of these alerts in order to avoid the chances of risking a malicious intrusion. Manual examination of millions of alerts on a daily basis can be an impossible task for system administrators. Therefore, the aim of the project is to implement alert correlation techniques in order to increase the performance of the IDS by segregating and classifying the large number of alerts into smaller and easily manageable groups.

1.2 Software Requirements

The desirable Operating system chosen to conduct the intrusion detection would be Ubuntu v.10.04. Linux systems have always known to be more secure and tend to provide the power to control most of its aspects [4].
SNORT (v.2.9.0.5) is one of the most widely used intrusion detection system/intrusion prevention system (IDS/IPS) and typically runs on Linux Operating systems [5]. It performs intrusion detection at the network level by inspecting the signatures of the attacks, protocols and the anomalies as well [6]. Barnyard(v.2) is another tool, which is augmented along with SNORT to read and analyze binary logs generated by SNORT as a result of an intrusion [43].

The database in which SNORT logs its alerts will be the MySQL database. An independent webpage has been set up to log the number of attacks along with a description of its signatures and a graphical representation of the different types of intrusions (Refer Figure 10 and 11). Nessus (v.4.4.1) port scanner is used to scan the experimental setup from a different system in an attempt to check the working conditions of Snort, Barnyard and MySQL. The webpage will report a few alerts as a result of the port scanning and this will confirm the accurate working and logging functionalities of the IDS. Once the experimental setup is fully functional, the trace files from DARPA are used to synthetically generate different attacks from different IP addresses globally [44]. Recent trace files are difficult to obtain and hence trace files designed for 1999 evaluation process are used in this project. As a result, the MySQL database logs thousands of alerts along with other important details such as the time when the attack was executed, the source IP address, the signature of the attack, etc. These alerts comprise of a greater percentage of false alerts known as false positives and a lesser percentage of real and malicious attacks. Python v. 2.6 and SQL are used to carry out the correlation techniques.

1.3 Scope of the Project and Document Structure

This project aims to assist the reader in understanding the role played by an IDS in detecting intrusions and the ways in which alerts can be correlated. More emphasis has been given on the correlation techniques as this is the current area of research in the field of an IDS. In this project, an overview on Computer Security will be provided in the second chapter for better understanding. A detailed account on the different types of Intrusion Detection Systems, their needs and the general advantages and weaknesses will be provided in the third chapter. The fourth chapter deals with SNORT and its interaction with MySQL and Barnyard in this particular project. The fifth chapter comprises of the different correlation techniques that has been implemented in order to increase the performance efficiency of SNORT. The sixth chapter projects the results and conclusions of the tests that have been carried out to correlate the alerts. The final chapter suggests the future work that can be carried out. The document ends with an appendix containing the supporting material. The Appendix contains codes used and other relevant data. A list of references is provided for better understanding of the Intrusion Detection Systems and the Correlation Techniques.
Chapter 2

Background Information on Security

Security can be classified into two types namely Computer Security and Network Security. Computer Security deals with securing the information stored in a computer alone, whereas Network Security deals with securing the entire network infrastructure [7]. When broadly discussing security, the foundation can be broken down in terms of the security goals namely: confidentiality, integrity, availability, authenticity, non-repudiation, access control and accountability [7] [8] [45].

2.1 Security Goals

Confidentiality, in terms of Information Security refers to the prevention of improper disclosure of resource, data or information that has been branded as sensitive or private to unauthorised sources [10]. Information that is used for personal identification such as social security number or a bank pin number can be classified as confidential. Integrity refers to the prevention of modification of resource, data or information by an unauthorised source. For instance, a legitimate source can only have the rights to alter a piece of information and/or its functionality.

Availability refers to prevention of the resource, data or information being withheld from legitimate sources. All information available via Internet must be kept accessible to the rightful viewers/owners at all times. The Denial of service attacks are the quintessential examples of unavailability. Authenticity refers to verification of the legitimacy of the source trying to access or modify a data or information. Some of the most common authentication techniques are the password login into a system or an account, biometric analysis, etc. [9] [8] [7] [10].

Non-repudiation is the process of acknowledging that the corresponding person has committed an action. This provides evidence that a particular person was indeed in charge of the action that had been committed. Access control restricts the rights of accessing a resource or information. This security goal is especially useful in an organisation, where employees of different ranks need to work together without revealing information to the lower ranked individuals [3]. Accountability deals with the storage of records in order to trace a security breach. Logging of system activities is an example of accountability as it helps the forensics to determine the type of security breach and the possibilities of its mastermind [10]. A security breach occurs when one or all of the above security goals have been compromised.
2.2 Vulnerabilities and Threat

The question of protection and security arises only in the presence of an attacker, an asset and vulnerability [10]. An asset can be considered as valuable information or a resource that a company or an individual tries to protect. Vulnerabilities arise as a result of improper or inadequate protection of the asset. It is during this course of time, an attacker tries to take advantage of the existing vulnerability, in an attempt to access the asset illegitimately. A threat arises when the attacker foils the security measures and the critical data or asset is eventually assessed.

2.3 Measures to control Threats

A security policy comes into effect for any organisation comprising of a network of computers. It is a documentation of a set of rules on what is permissible and what is not permissible with respect to security. A security policy covers all possible and suitable measures for a company such as the firewall rules, acceptable use of hardware, encryption policy, regulation of activities over the network and the system, procedures for new installations and configurations, secure usage of VPN and wireless devices, server security, router security, etc.

A weak security policy often results in several vulnerabilities that make it easy for the attacker to launch an attack. Therefore, it is essential that organisations constantly review, test and update their security policies in order to maintain a strong and secure network infrastructure. Risk Analysis is an equally important task that prepares the companies for a possible attack by calculating the loss and cost of a new alternative [8].

Cryptography is not a security measure, but acts as an additional layer of protection along with the above-mentioned measures. This can prolong the duration of the attack if the attacker manages to access the encrypted data by crossing all other security barriers. For instance, it is easy to misuse an unencrypted password or bank login details that is sent over the Internet when compared to the very same that has been encrypted with a tough encryption cipher.

2.4 Failure of Security Measures

Even though the above measures such as having strong security policies or defining appropriate security techniques are the suggested remedies, these measures are not totally successful at eliminating intrusions. Security and privacy of an organisation also depends majorly upon the sensible usage of systems and the network by the employees. Social engineering is a simple method to steal passwords and information from a person without having to endure the process of intruding [9].
The other ways in which an attack can be initiated is by guessing an easy password by using dictionary attack [50]. One of the most common software used for a dictionary attack is “John the Ripper”. This software easily discovers passwords consisting of common names or dictionary words. Ciphers can also be deciphered within a couple of days with the help of computerised machines unless the key size or block size is large.

**IP Address Spoofing** [51] can easily overrule the security policy by faking an IP address to perform malicious activities. A simple ICMP ping packet can be turned into the **ping of death attack** by continuously flooding a server with ping packets. This is a well-known denial of service attack, which most attackers implement to intrude and collapse a server. A detailed explanation for ping of death is available in Section 5.3.

**Buffer Overflow** is an attack planted as a result of vulnerability spotted in a program or a piece of software code that has not been checked or tested [52]. The attacker who spots the vulnerability before the programmer or the software developer (zero-day exploit) can rewrite the actual return address of the program with another value so that a malicious code is fetched and executed instead. The malicious code that was injected into the actual program is also known as the **shell code** [49].

Another attack that can escape the clutches of the security measures is the malware. Inserting a virus or a Trojan in software is the easiest way to take control of the system and further perform malicious activities.

These are some of the attacks that have been implemented beyond the restrictions of the security measures implemented in an organisation. Since these security measures can fail, an Intrusion Detection system tends to be an additional protection and can be utilised to determine any symptoms of a possible intrusion.
3 Chapter 3

Intrusion Detection System

An intrusion can be defined as a successful attack that exploits vulnerability in a system or network infrastructure resulting in the violation of the security policy [3] [11]. However, an intrusion detection system can prevent attacks by detecting an intrusion through vigilant monitoring of networks and computers and reporting alerts in the instance of an attack. This can be compared with that of a real time burglar alarm, where the alarm alerts the owner of a possible burglary. When the Intrusion Detection System spots a violation of security policy, it immediately logs details of the event and presents it in the format of a report to the system administrator.

A diagrammatic representation of those events that must trigger alerts from an Intrusion Detection System is given below:

![Diagram of Intrusion Detection System]

Figure 1: Suspicious events that would trigger alerts from an IDS [11] [3]

According to Lindqvist, a good Intrusion Detection System must be able to detect intrusions belonging to the above category. ProbingProvocation is one of the most common intrusions attempted by attackers. This is generally carried out using a port scanner such as Nessus in an attempt to determine any ports that are unguarded.

Both Circumvention and penetration refer to accessing the internal resource by tricking or breaking through the security measures. A suitable example would be using a proxy server to access a restricted website or resource. Insider refers to the exploitation of data by a person within the organisation who does not hold the rights to access a particular resource.
3.1 Functions of IDS

Irrespective of the type, all the IDS typically have the same functionalities:

- **Observing and Monitoring**: An IDS observes the system and the network for any suspicious events. The criteria for observation depend on the type of IDS. A detailed description on the types of IDS and their modes of observations is provided under the Section 3.4.

- **Logging of Events**: On encountering a suspicious activity, the IDS records the information related to the observed activity. This is either performed locally by the concerned system (if the IDS has been installed on a single system) or by a centralised logging server (if the IDS has been set up for monitoring an entire network) [28].

- **Alerting System Administrators**: Once the events have been logged onto a database, the IDS can be set up to send alerts to the System Administrator. An IDS can send alerts through web pages, emails, messages, etc [28].

- **Reports on Intrusions**: A detailed report is prepared listing the details on the events, which had been captured and logged. These reports can be used by System Administrators to analyse the security setup for the organisation and for determining vulnerabilities.

3.2 Need for IDS

According to Bace and Mell, an Intrusion Detection System can be implemented by an organisation for the following reasons [20] [12]:

- An IDS tends to act as an extra layer of protection along with the other security mechanisms in an organisation.
- It aids in the process of detecting intrusions and other malicious events when other security measures in the organisation fail.
- It can detect an attack in its preliminary stages when the attacker initiates a port scan to determine vulnerable ports.
- It can log events and present reports that can be used by the system administrator to determine the existing threats to an organisation.
- It acts as quality control tool and aids in strengthening the vulnerabilities in an organisation.
- It provides an easy technique for analysing the security measures of an organisation.

3.3 Common Intrusion Detection Framework

A Common Intrusion Detection Framework was proposed in order to describe the general functionality of an Intrusion Detection System [15] [3]. CIDF can be
thought of as a standard that has been used to define the basis of an IDS and its mechanisms. The Intrusion Detection Systems of the present day have been developed using the basic foundation of the CIDF.

The model suggested above consists of four major components namely: Event Generator, Storage Unit, Analysis Unit and Response Unit [13] [3]. These components combined together can perform the actual intrusion detection process. The event generator is responsible for the observation and monitoring process. It is the function of the event generator to make a note of all the suspicious occurring in the network and in the systems. All the events that have been captured by the event generator are then sent over to the analysis engine. The analysis engine analyses the events that have been produced. The storage unit stores in the details of the events as well as the results of the analysis for future retrieval. The response unit displays the results of the intrusion detection process. The results can either be a graphical representation of the number of attacks or a report of the attacks along with details of their times or a combination of both.

CIDF has also defined the protocols for the communication of its four basic components. An accurate design would be required in order to achieve the purpose of tasks defined for each component in the CIDF model. Similar to CIDF, there are other standards that support the foundation for an Intrusion Detection System like IDMEF, SASL, IDIP, IDXP, etc [16] [3] [17] [18].

3.4 Classification of IDS

An Intrusion Detection System generally works by detecting an intrusion while in progress and then triggers alerts, which can then be utilised by the System Administrator to take the necessary steps. Intrusion Detection Systems can generally be classified as Host Intrusion Detection System (HIDS) and Network Intrusion Detection System (NIDS). The HIDS is specifically used to detect intrusions in a particular sys-
tem alone. All the activities in a single system will be monitored by the HIDS and alerts are generated if the system is prone to attacks and intrusions. The NIDS is used to detect intrusions in a network comprising of many systems, routers, modems, hubs, printers, scanners, switches, etc. that are directly connected to the Internet by monitoring the network traffic. On detection of any signs of an attack, alerts are immediately reported to the System Administrator [19] [3].

This project makes use of a NIDS tool and therefore more importance shall be given to the description of NIDS and its functionality. NIDS can be further classified into two types namely: **rule (or signature) based NIDS** and **heuristic (or anomaly) based NIDS**.

A rule (or signature) based NIDS works by means of rules. Rules are either already defined or new rules can be created and inserted by the System Administrator, according to the security needs of the network. These rules are used by the NIDS to detect the signatures or patterns from the nature of the attack.

![Diagram of Types of IDS](image)

**Figure 3: Classification of IDS**

Some of the most common examples of the situations where a NIDS can be brought to effect are [19]:

**Port Scanning Attacks:** Attackers generally perform a port scanning before launching the actual attack. This provides the attacker with a good insight of which port might be vulnerable and open. For instance, an NIDS can be designed to monitor packets that are generated from a single IP address more than five times within a given period (assume five seconds).

**Packets with Malicious Scripts:** Sometimes, worms can randomly send malicious packets with hidden scripts in an attempt to exploit a vulnerable system. A rule in the NIDS can be written to monitor and search packets for textual strings or for symbols...
such as ‘&’, ‘!', ‘$’, etc. An alert will be triggered by the NIDS on encountering packets with such symbols.

Login Attacks: The rules in NIDS can be written to safeguard the security policy of an organisation. For instance, an NIDS may work towards determining repeated incorrect login attempts (assume more than 5 times) to a restricted machine and triggering alerts when the rule has been broken.

From the above-specified situations where a rule-based NIDS can be used, it is clear that this category can only detect intrusions that conform to the pre-defined rules. Intrusions for which the rules do not form a part of the NIDS can go unnoticed. Therefore, the heuristic (anomaly) based NIDS was formulated in a different way.

As the name suggests, the Anomaly-based NIDS reports alerts whenever anomalous behaviour is sensed. In other words, the normal and secure conditions of the network are “taught” to the NIDS and monitoring is carried out to detect anything that does not conform to the normal conditions. For instance, an abnormal or anomalous behaviour that could possibly trigger an alert from the NIDS is when a user, who has restricted access to the network tries to access a particular resource. In this case, previous modes of behaviour must be learnt by the system in order to tell the difference between a normal act and an event initiated by the attacker. Since this type of an NIDS is based more on logical algorithms, design of such NIDS takes the aid of Artificial Intelligence [19]. This project uses a signature-based NIDS tool in particular and more about this category will be discussed in detail later.

This is the most common method of classifying Intrusion Detection Systems. However, other attempts to classify an Intrusion Detection System have been proposed. The most popular modes of classification out of those are the ones proposed by Axelsson [22] and Debar et al [21] [23]. Debar et al. present four different methods by which classification can be carried out namely: detection mode, frequency of usage, audit mode and response techniques [22].

Passive alerting and active alerting differentiate the response techniques in an IDS [3]. A Passive alerting IDS performs the mere action of alerting the System Administrator by means of a report and does not perform any action to mitigate or prevent the intrusion. An active alerting IDS tries to prevent the effects of the intrusion apart from alerting the system administrator about the intrusion. An extension of the active alerting is the Intrusion Prevention System (IPS). In this project, the IDS is of passive alerting as it merely records the alerts and reports to the user.
3.5 Challenges faced by IDS

Even though, different classifications of IDS have been proposed, the problem remains the same in all the IDS. The three most important research questions that have been raised towards all the proposals of the IDS are [3]:

- How can the IDS be secured from attacks itself?
- How can its effectiveness be improved?
- What techniques can be used to improve the performance efficiency of an IDS?

There are four possible results that can be generated by an Intrusion Detection System as a result of an intrusion. They are true positive, true negative, false positive and false negative.

<table>
<thead>
<tr>
<th>Intrusive Event</th>
<th>Non-intrusive Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrusive Decision</td>
<td>True Positive</td>
</tr>
<tr>
<td></td>
<td>False Positive</td>
</tr>
<tr>
<td>Non-intrusive Decision</td>
<td>False Negative</td>
</tr>
<tr>
<td>(including missed events)</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Figure 4: Possible outcomes of IDS [3]

From the above table, the effectiveness of an Intrusion Detection System can be evaluated and the IDS can be graded for its performance.

- **True Positive**: When the IDS justifies a malicious event as malicious and reports an alert, then the alert is known as a true positive. For instance, a true positive outcome occurs when a real port scanning activity is detected by the IDS and an alert is sent to the System administrator. A good IDS should produce high percentage of true positives.
- **False Positive**: When the IDS justifies a non-intrusive event as malicious, then the alert is known as false positive. For instance, a false positive outcome occurs when a harmless ICMP ping packet is mistaken by the IDS for an intrusion and reports an alert message to the System Administrator. Most IDS suffer from low efficiency as a result of high percentage of false positives.
- **False Negative**: When a malicious activity is ignored by the IDS and when it fails to alert the system administrator about it, then the result is termed as false negative. For instance, a real attempt to intrude a system could be turned down by the IDS as a normal behaviour. This is the most dangerous outcome of an IDS and such an IDS is said to have a zero-percent efficiency.
- True Negative: When a non-intrusive event is correctly recognised as harm-
less by the IDS and when it ignores the event without reporting, then this can
be termed as a true negative.

As a result of these research questions, the weaknesses of an IDS have been rec-
ognised and listed out by Xu and Ning [31] [12]:

- The greatest disadvantage of an IDS is the generation of thousands of
false alerts along with the genuine alerts. It has been reported by System
administrators that 326 alerts were produced on one occasion, as a result
of an IDS being installed on a campus network on a weekly basis. Most
other IDS, installed on enterprise networks, produce much larger amounts
of false positives than on the above instance.
- Another drawback of an IDS is that 99% of these alerts turn out to be
false positives and with such a large number of false positives, it is over-
whelming for a System Administrator to identify the remaining 1% of
genuine alerts, on a daily basis.
- A much dangerous situation occurs when the IDS fails to detect the genu-
ine attacks and generates false negatives. This could mislead the System
Administrator into overlooking the security installed for the network.
This could also result in the System Administrator’s failing to understand
the present threats and vulnerabilities.

To increase the efficiency of IDS, the aim should be towards increasing the per-
centage of receiving true positives and true negatives as its outcome and reducing the
amount of false positives and false negatives. Many techniques were researched in or-
der to reduce the false positives from being generated. Research on achieving an effi-
cient and effective Intrusion Detection System is currently carried out by DARPA.

An initial attempt to evaluate intrusion detection process was begun by DARPA in
1998 [3] [24] [25] [26] [27]. The project was named as IA:AIDE (Information Assur-
ance: Automated Intrusion Detection Environment). This project was started with the
aim to discover secure technologies in the field of Intrusion Detection that can be util-
ised in military bases to communicate securely without risking any intrusions.

3.6 Placing an IDS

For permitting maximum efficiency, an IDS needs to be positioned in the most
appropriate location. Choosing the right location is highly dependent upon the size of
the network as well as upon the type of intrusion detection required for the network. It
is very important for the System Administrator to analyse the entire network topology
carefully before deciding the exact spots to deploy an IDS. Some of the hotspots that
must be considered before the deployment are the access points, remote access points,
points separated within an intranet, etc [46].
For an enterprise having many networks connected to the Internet, an IDS should be placed in every possible entry point. This detects intrusions trying to enter the network of the enterprise. When an internal detection is required, i.e. if an attacker has gained access to the internal network of the enterprise, an IDS would be required to detect intrusions from within. Therefore, an IDS should be placed onto every segment in the internal network [6].

![Diagram of IDS placement in a network](image)

**Figure 5: Placing IDS in a Network [6]**

Another possibility is determining the *hot spots* in the network and placing IDS in only those locations. Hot spots are those areas that demand extra security and protection. This would reduce the cost of deploying several IDS in a network.

According to Chris Calabrese, the IDS sensor can be placed in following ways depending upon the network topology [47]:

- Single-segment IDS probe
- Multiple segment IDS probe

The multiple segment IDS probe can further be classified as *multiple single-segment probes* and *multiple multi-segment probes*. This determines the necessity for a single IDS or many IDS over a single segment or multiple segments.
4 Chapter 4

Intrusion Detection using SNORT

It was believed by much security and IT specialists that a firewall would be more than necessary to protect a network against malicious attacks. An IDS was questioned by many. However, the need for an IDS has been enlightened in the previous chapters.

SNORT is a signature-based Network Intrusion Detection System tool, developed by Martin Roesch in 1998 that will be used in this project. SNORT was specifically chosen as it is a light weight IDS that is suitable for a small network. The rules used in SNORT are predefined and they can easily be edited as per the security needs of the organisation. Currently, it is one of the most widely used IDS as it can be utilised to detect Denial of Service (DoS) attacks, buffer overflow attacks, port scanning, scanning worms and viruses, SQL injections, HTTP injections, etc [12].

One of the major advantages of SNORT is that it could be used along with an external database such as MySQL in order to log the alerts and an output system such as Barnyard. Barnyard can read the output of SNORT, which is in a special binary format called as ‘unified’ and send back the data into the database [43]. In general, SNORT plays the combined roles of a packet sniffer, packet logger and a network monitor.

4.1 Components of SNORT

In order to understand the functioning of SNORT as an IDS, it is important for the reader to understand about the different components of SNORT and the tasks performed by each. As described by Rehman, SNORT consists of five different components, which together can detect intrusions and log alerts. The five major components of SNORT are: packet decoder, pre-processor, detection engine, logging and alert systems and output module [6]. On close examination, it is evident that SNORT is designed based on the CIDF model.

The diagrammatic representation of the different components is shown below. All the packets originating from the network are initially sent to the packet decoder and depending on the nature of the packet, it is either dropped or an alert is reported to the System Administrator [6].
Packet Decoder: The packet decoder captures all the packets from the network and these packets are sent to the pre-processor.

Pre-processor: A pre-processor is used to aid the process of detection before sending the packets to the detection engine. An attacker can create a malicious packet in such a way that the IDS does not detect it. An attacker can disguise a packet to escape a signature. The pre-processor is responsible for detecting such disguised packets. For instance, if a rule has been written to detect packets with strings “httpd/conf”, the attacker can try to escape being detected by modifying it as “httpd/.conf”. It is the role of the pre-processor to identify the forged packet and rearrange to its original format before sending the packet to the detection engine. Similarly, an attacker can fragment a single packet into several smaller packets in order to hide the signature from being detected by the IDS. A pre-processor puts back the fragments as one single malicious packet.

Detection Engine: The detection engine is the most important component in the SNORT. The detection engine utilises the rules in determining whether a packet contains a signature or not. A rule can be divided into two parts namely the rule header and the options [29]. The rule header carries details about the action that needs to be executed by the rule and the criteria for matching an incoming packet. The rules are matched against all the incoming packets. The options field carries additional information that can be used to for rule matching against packets and also to determine which portion of the packet can be used to trigger the alert message. If a packet matches a rule, then an alert is generated by the next component (Logging and Alerting System) of SNORT, else the packet is dropped by the detection engine. In specific, a packet is dissected and the rules are matched against different parts of the

Figure 6: SNORT Components [6][12]
packet such as the payload, DLL header, IP header, Transport Layer Header and the Application Layer header [6].

Logging and Alert System: Once, the detection engine detects a malicious packet after matching against the rules, details about the type of packet, its signature, source and destination addresses, the time of generation, etc. are logged. The logs generated by SNORT are known as unified and are in a binary format. These can be stored as text files or tcp-dump-style files [6]. A report can be sent listing the number of alerts to the System administrator via email, message or a webpage.

Output Modules: An output module is used to decide how the generated logs are saved as an output. There are many options based on how the System Administrator wishes to view the output. In this project, SNORT is designed to log all its alerts to the MYSQL database.

4.2 SNORT Modes

SNORT uses three modes commonly to perform the act of intrusion detection namely, packet dump, packet logger and network IDS[12]. Packet dump mode is used to display all the packets that are captured by the IDS. The Packet logger or the packet sniffer mode is used to log all the captured packets to a file in order to serve as a report. In the network IDS mode, the actual analysis is performed by SNORT on those packets that have been captured to determine if they are malicious or not. As a result of this analysis, alerts are generated.

4.3 Experimental Setup

In the following sub-sections, a brief description of how the experimental setup had evolved has been provided. The SNORT Installation Guide by David Gullett is followed for installing SNORT and the other packages. The experimental setup would require a single system to install the IDS and the other supporting software. Nessus can be installed on a different system to check if the SNORT reports errors on performing a port scan.

4.3.1 Installing supporting packages on Operating System

Ubuntu 10.04 LTS Linux Operating System had been installed onto the hard disk. Apart from the major tools described in Section 1.2, a few more packages will be required to aid the experiment. This forms a part of the pre-installation process. The below packages can be installed using the Linux bash script ‘sudo apt-get install’ [41] (Refer Appendix for description of the packages):

- apache2
- php5
- php5-mysql
4.3.2 MySQL Installation

Once the basic packages have been installed, the next step is the installation of MySQL. A root user and password is provided in order to use the MySQL database. Similar to the above commands, sudo apt-get install is used:

- mysql-server
- libmysqlclient16-dev

4.3.3 SNORT Installation

The installation guide written by David Gullet for Ubuntu 10.04 has been utilised for the entire installation process. Before installing SNORT in the System, the SNORT report is downloaded and modifications are made in the srconf.php file [41].

The SNORT 2.9.0 package is downloaded and a separate SNORT database is created in the MySQL in order to log alerts directly in this location. Modifiers to the snort.conf file are made. The next stage is to download the latest snort rules. The rule-set for a 64-bit computer is downloaded as per the System requirements.

4.3.4 Barnyard2 Installation

Barnyard is mainly installed to reduce the load on the detection engine of SNORT. The binary output of SNORT, named unified is fetched by Barnyard 2 and entered into the MySQL database in a readable format. If the database is setup online and if it is not available for the moment, then the Barnyard would ensure that all the alerts are transferred to the database when it is available without generating any loss of alerts.

The latest version of Barnyard is downloaded and installed similar to all the other packages. Modifications are made to the Barnyard configuration file by including the user name and password details of MySQL [41].

4.3.5 Testing SNORT using Nessus

This step is carried out in this project to ensure that all the tools have been installed correctly and SNORT reports alerts in its webpage when an intrusion is generated using Nessus port scanning.
Nessus 4.4.1 is installed on another system. A port scanning is performed by specifying the source address (the address of the system where Nessus is installed) and the destination address (the address of the system where SNORT needs to detect the intrusion). Before scanning the destination system, the SNORT and Barnyard that has been installed on it must be activated. This is done using a shell script as shown below:

```
#!/bin/bash

# Start SNORT
/usr/local/bin/snort -u snort -g snort -c
/usr/local/snort/etc/snort.conf -r $1

# Start Barnyard
/usr/local/bin/barnyard2 -c
/usr/local/snort/etc/barnyard2.conf -d /var/log/snort -f
snort.u2 -w /var/log/snort/barnyard2.waldo
```

![Figure 7: Script to start SNORT and Barnyard](image)

The above script is saved as `.run-snort.sh` and is run on the console. The output of the script is as shown in the figure below:

![Figure 8: SNORT and Barnyard activated](image)

Now the port scanning can be launched from Nessus. Once the Nessus finishes the scanning, SNORT and Barnyard can be stopped by using a similar script:

```
#!/bin/bash

# Stop SNORT
killall snort
sleep 5
killall barnyard2
```

![Figure 9: Script to stop SNORT and Barnyard](image)
As a result of the port scanning, the SNORT database webpage logs hundreds of alerts with details of several unique signatures. This shows that the SNORT is functioning appropriately.

The total number of alerts logged as a result of a single port scan is 544 and each of these attacks belong to the one of the 26 signatures recorded. From the above figure, the total number of attacks unique of a particular signature has been listed in the column named Alerts. This shows how many alerts are of a specific type of an attack. The Common Vulnerabilities and Exposures (CVE) is a directory that lists and describes the most common vulnerabilities that the Internet Security had encountered so far.

Distinguishing true positives from false positives can also be based on studying the signatures in detail and determining whether it forms a part of those that have been listed on the CVE directory. A good System Administrator must ensure that he or she subscribes to the CVE newsletters and updates the security mechanisms of the systems and the network accordingly.
DARPA has been actively participating in the attempts to evaluate the performance of the IDS in the years 1998 and 1999 respectively [9] [32] [33] [34] [35]. These tests involved generating different types of attacks against an experimental IDS and testing its efficiency in producing accurate results as an output. In order to generate a combination of these attacks, data sets or test files were created, which were capable of producing the results equivalent to a port scanning, denial of service attack and many other similar intrusions.

Though the tests on IDS did not produce the intended outcome, these test files, ever since, had become a standardised way of evaluating an IDS/IPS. DARPA started releasing more of these trace files as these were adopted by many commercial organisations in developing IDS/IPS. However, recent DARPA trace files are not available easily and testers commonly use trace files designed for the 1998 and 1999 evaluations.

A synthetically packet-generating trace file from the 1999 DARPA tests is downloaded and is run against SNORT to produce attacks. Similar to the steps followed by the Nessus port scanning, the SNORT and Barnyard tools are activated and started. The script includes a statement to run the trace file once the SNORT and Barnyard tools have been initiated.

```bash
#!/bin/bash
echo "I am starting snort on $1"
/usr/local/bin/snort -u snort -g snort -c
/usr/local/snort/etc/snort.conf -r $1
echo "I am starting barnyard"
/usr/local/bin/barnyard2 -c
/usr/local/snort/etc/barnyard2.conf -d /var/log/snort -f
snort.u2 -w /var/log/snort/barnyard2.waldo
```

As a result, 57,912 alerts have been recorded in the MySQL database with 63 unique signatures as in the case of the Nessus scan.
5 Chapter 5

Alert Correlation Techniques

Out of the many techniques being researched in an attempt to increase the efficiency of an IDS, alert correlation is the most prominent one. As the name suggests, alert correlation is the process where all the alerts logged in the database are considered and a more succinct detail of the intrusions is provided. The main goal of correlating alerts is to eliminate most of the false positives, if not all.

The major alert correlation technique that has been chosen for this project deals with using MySQL statements and Python programming. The MySQL database can be accessed using the username and password that had been set during its installation. The SNORT database is chosen and the tables stored in the SNORT database can be viewed. An insight into the contents of the tables in SNORT is provided in the figure below:

Correlation techniques can generally be controlled using a combination of any of the above mentioned tables from the SNORT database. In this project, the correlations will revolve around the `event`, `iphdr`, `signature` and `icmp` tables in particular. Therefore, it is important for the reader to understand the contents of each of these tables and about the columns in those tables, which would be considered for the process of correlating the alerts. Three correlation techniques have been proposed namely:
- Three-level Correlation Model
- Correlation based on Signature and Timestamp Examination
- Correlation based on ICMP type and code

These techniques are used to increase the efficiency of the SNORT IDS by reducing the number of false positives.

5.1 Three-level Correlation Model

In this section, a three-level correlation model has been proposed to correlate the alerts logged by the SNORT. Correlation is carried out in three stages. The main objective of the Three-level Correlation Model is to eliminate all those IP addresses, which generate a single alert. There is a high probability that the IP addresses generating the single alerts could be a harmless packet. However, this assumption by the Three-level Correlation Model should be tested by examining the signatures in detail. The three stages of the Three-level Correlation Model have been explained in the flow diagram below:

![Flow Diagram](image-url)

Figure 14: Three-Level Correlation Model
First Level – In this level, the total number of alerts reported from each IP Address is counted and displayed. A graphical representation of output details the alerts generated from each IP address.

Second Level – In this level, the output of the first level is displayed in an ascending order with the IP addresses generating the least alerts displayed first and those generating the highest displayed towards the end of the table.

Third Level – In this level, the output from the second level is used and those IP addresses, which have generated a single alert, are separated from the rest of the IP addresses and are displayed.

5.1.1 Contents of iphdr

The correlation is carried out by utilising the MySQL database to re-arrange the required columns. In order to proceed with the Three-level Correlation technique, the iphdr table would be required.

The iphdr table generally provides information about the IP header of the packets that have been captured by the SNORT. Examination of the IP Packets for details of an attack is considered very important as it reveals the source address from which the attacks have been initiated. It is with the Source IP address that the first step of correlation will be initiated. A brief description of contents of iphdr and the SQL statement required to access the table is provided below:

```
select * from iphdr;
```

SID - is used to add SNORT ID to the SNORT rules. The SID value can be used by the output module component in the SNORT to identify the particular rule.
CID - is similar to a serial number, logging all the alerts in an order.
IP_SRC - denotes the IP addresses from which the attacks have been generated (IP addresses are represented in hexadecimal in the MySQL database and not in the dotted decimal notation).
IP_DST - denotes the IP addresses to which the attacks were intended. Since SNORT is a NIDS, the attack packets have been intended for more than a single destination address. Similar to IP_SRC, IP_DST are represented in hexadecimal notation.
IP_VER - denotes the version used by the IP packets. In this case, the packets belong to the standard IPv4.
IP_HLEN - denotes the total length of the IP header alone. The minimum permissible length of an IP header is 5.
IP_TOS - represents the type of service used by the IP packet. In this instance, the Type of Service used by the IP packet is 0, which symbolises that it falls under the
category ‘**routine**’. All those packets whose transmissions originate electrically use an
IP packet with TOS as routine.

**IP_LEN** - represents the total length of the IP packet including that of the IP header
length.

**IP_ID** - involves the Identification field, which acts like a sequence number and is
used to identify to which particular datagram the fragmented IP packets belong.

**IP_FLAGS** - stands for reserved (0), don’t fragment (1) or more fragments (2).

**IP_OFF** - refers to the offset of a particular fragment of IP datagram.

**IP_TTL** - represents the time taken for a packet to live and is used to prevent lost
packets from looping in the network.

**IP_PROTO** - represents the protocol used by the IP datagram.

**IP_CSUM** - represents the checksum used by the destination to determine if the
packet arrives with an error [37].

![Table of iphdr](image)

**Figure 15: Contents of table ‘iphdr’**

### 5.1.2 First Level Correlation

The column that will be of concern to the correlation process would be the
IP_SRC column from the iphdr table. A simple SQL statement is used to enable the
Python script to count the number attacks generated. *(Refer Appendix for Code to
Count Number of Attacks)*.

```sql
select ip_src, count(*) from iphdr group by ip_src;
```
As a result of executing this code, the output is displayed as below. The code has displayed all the IP addresses present in the IP_SRC column of the iphdr table and the number of times each IP address has generated an alert message. This is not the actual correlation process, but tends to provide information on how many alerts messages were generated from each IP address.

![Figure 16: First Level Correlation Output](image)

A graph can be plotted as a result of the output displayed above between the IP addresses and the number of alerts that were generated from each of the IP addresses. This would provide the System Administrator with a clear insight of which IP addresses produce the peak alerts and which IP addresses produce the least alerts. A cutoff range of 2 alerts is provided below which alerts can be considered as false positives and above which a further investigation can be made.
The graph above represents the IP address, 3256613653 (hexadecimal notation) as having the highest alerts count of 14000. This could be the possibility of Denial of Service of Attack, where a substantial amount of packets are generated and steered towards a particular server. The other IP addresses generating a similar amount of alerts are 3317062633 (11000 alerts), 2259747619 (9500 alerts) and 32348288594 (3500 alerts). It is possible for all these alerts to be a part of the same attack i.e. using different systems to increase the traffic being sent over to the destination. This proposal can be determined by considering the period over which all the attacks from the above-mentioned IP addresses commence. If these attacks are generated over a short period, there is very high probability that the same attacker had constructed these attacks.

Another way in which, the above proposal can be determined is by examining the signatures of the attacks originating from the above-mentioned IP addresses. If the attack packets belong to a particular class of signature or a closely resembling signature, then these attacks can be grouped together and labelled as an attack originated by the same attacker. The two methods have been attempted in the Section 5.2.

5.1.3 Second Level Correlation

From the above output, it is possible to judge which IP address could be the actual attack generating system. However, this process would take a little longer as the output contains a mixture of high and low alerts generating IP addresses. Therefore, another SQL command can be used inside a python script to group all those IP
addresses generating lower number of alerts to a higher number of alerts as shown below (Refer Appendix for Code to Order Alerts According to their Number):

```
select ip_src, count(*) as number from iphdr group by ip_src order by number;
```

As a result of executing the python script, the IP addresses and their number of attacks are displayed in an order starting from the least number of attacks to the most number of attacks. This output can be utilised to segregate those IP addresses generating one or two alerts from the rest. As specified before, there is a high probability that these alerts can be false positives. However, further correlation processes can be executed to prove that they are genuinely false rather than assuming.

The figure below shows all those IP addresses which generate a single attack and this is followed by those addresses generating two attacks and so on. The list ends with IP addresses generating the highest number of alerts, which is 14,000, from the graph represented in Figure 13.

![Figure 18: Second Level Correlation Output](image)

A graph is plotted between the IP Addresses and the alerts generated. It is now simple to carry out the final step in correlation, which involves eradication of those IP addresses with the least alerts.
The final step in the Three-Level Correlation Model involves splitting IP addresses producing the least count of alerts from the rest of the database. The SQL statement that can be used to group all the IP addresses producing single alerts is given below (Refer the Appendix for code to segregate IP addresses producing a single alert):

```sql
select ip_src, count(*) as number from iphdr group by ip_src having number<2 order by number;
```

![Figure 19: Second Level Correlation of IP Address vs. Alerts](image)

![Figure 20: Third Level Correlation Output](image)
5.1.5 Result

As a result, all the three levels of correlation have been successful. The final output obtained from the three-level correlation model is shown below:

![Third Level Correlation](image)

Figure 21: Final Output of the Three-level Correlation Model

The output produced is all those IP addresses whose alerts count is 1. This correlation technique has removed 169 alerts from 57,912 alerts. The remaining 57,743 alerts can be correlated using the other two techniques suggested below. Not all these single alerts can be harmless. There is a possibility that one out of all the single alerts could be malicious. To extend the analysis on this output in the future, the signatures of the above attacks can be examined. Any signature whose alert appears false is considered and a correlation filter can be designed to filter out all those alerts with the same signature out of the 169 alerts.

5.2 Correlation based on Signature and Timestamp Examination

Another method of correlation is investigating the nature of the signature and the time at which an attack is generated. The idea proposed is to combine the columns ‘signature’ and ‘timestamp’ from the table ‘event’ along with the column ‘ip_src’ from the table ‘iphdr’. The column signature lists out numbers, which are the same as the numbers displayed in the column ‘sig_id’ of the table ‘signature’.

The ‘signature’ table lists all the general signatures that SNORT is capable of identifying and capturing alongside the sig_id. (Refer the Appendix for the signature table). Another important component of the signature table is the ‘sig_priority’ which lists out which signature has more priority in terms of being malicious. It is possible to correlate alerts based on signature priority as well. The figure below describes the general columns contained in event. This table can be accessed using the following SQL statement:
This table lists all the events and stores the exact time when they were generated. A description of all the columns in the table is provided below:

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>signature</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7133</td>
<td>12</td>
<td>1999-04-01 14:00:05</td>
</tr>
<tr>
<td>1</td>
<td>7134</td>
<td>13</td>
<td>1999-04-01 14:00:05</td>
</tr>
<tr>
<td>1</td>
<td>7135</td>
<td>24</td>
<td>1999-04-01 14:00:01</td>
</tr>
<tr>
<td>1</td>
<td>7136</td>
<td>48</td>
<td>1999-04-01 14:01:00</td>
</tr>
<tr>
<td>1</td>
<td>7137</td>
<td>24</td>
<td>1999-04-01 14:00:01</td>
</tr>
<tr>
<td>1</td>
<td>7138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7142</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7156</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 22: Contents of the table ‘event’

The sid and cid field perform similar functions to that of the iphdr table. Signature - lists numbers which are the same as the sig_id field in the signature table. The corresponding description of each sig_id can be fully obtained from the signature table. (Refer Appendix for the signature table containing the description of each signature).

Timestamp - signifies the date and time at which the alerts were generated for each alert. (DARPA datasets were created to produce synthetic alerts for the tests that were performed in the year 1999 and hence the year 1999).

On examining the signatures and their priorities from the signature table, a few signatures could be termed as generating false positive alerts. For instance, the ICMP Destination Unreachable Port Unreachable (sig_id=2), ICMP Timestamp Request (sig_id=3), ICMP Timestamp Reply (sig_id = 4), ICMP Destination Unreachable Host Unreachable (sig_id = 32), ssh: Protocol Mismatch (sig_id = 43), etc. could be those signatures which might give rise to false positives in most cases. However, these ICMP packets could also be used to initiate an attack. To determine whether these signatures, in this particular test environment are true positives or not,
the timestamp can be examined along with the number of times a particular IP address
generates alerts due to the same signature.

```sql
select event.signature, iphdr.ip_src from event natural join iphdr;
```

The SQL command above is used to join the signature column from the event
table along with the ip_src column from the iphdr table. In other words, it lists the
source IP addresses and the signatures of their respective alerts as shown below:

![Figure 23: IP Address and Signatures](image)

For experimental purposes, the signatures 32 and 43 are taken into consideration
and examined in detail. The other signatures can be examined and correlated as a fu-
ture extension of this project. An ICMP Destination Unreachable message is generally
issued when the packets sent are not received by the destination. On observing the
complete database, it is evident that signature 32 occurs only a few times (6 times) at
different periods from two IP addresses (3319443781 and 3232236823) as shown in
Figure 24. Therefore, there is a high probability that alerts generated by signature 32
could be false positives. The alerts from signature 32 can be separated from the entire
database using the command listed below.

It is also observed that signature 43 (ssh: Protocol mismatch) has resulted in
the highest number of alerts (47,595) within a small time frame. The protocol mis-
mismatch alerts are generated when the SSH versions used by the Client and Server do
not match. This situation could occur when a Client and Server using two different
versions of SSH try to connect or when a non-SSH Client tries to connect to an SSH
Server. This is a bug in the current versions of SNORT which produces a huge
amount of Protocol mismatch alerts when the auto-detect feature is turned ON [39]. Therefore, these alerts can be grouped together and eradicated from the entire list.

5.2.1 Correlation of Signature 32

The statement below is used within a Python script to separate the alerts generated by the signature 32:

\[
\text{select event.signature, iphdr.ip_src from event natural join iphdr where event.signature=32;}
\]

The resulting output provides a list of IP addresses whose alerts have been recorded on account of the signature 32. These could be eradicated from the list as false positives.

![Signature 32 separated]

Figure 24: Signature 32 separated

5.2.2 Correlation of Signature 43

This correlation process is carried out by combining three columns: the signature column from the event table, the timestamp column from the event table and the ip_src column from the iphdr table. The corresponding SQL statement is provided below (Refer Appendix for program):

\[
\text{select event.signature, event.timestamp, iphdr.ip_src from event natural join iphdr where event.signature=43;}
\]

On examining the database of the output displayed below, it is evident that a huge number of false positives (47,595) have been generated from two IP addresses 3276379954 and 2265831615 continuously.
As a result of correlating Signatures 32 (6 alerts) and 43 (44,693 alerts), a large number of false positives has been removed from the remaining SNORT database of 57,743 alerts. By the end of this correlation, the remaining alerts are reduced to 13,044. Nearly 80% of the false positives have been reduced using correlation techniques based on signature and timestamp examination. The remaining can be correlated using the technique suggested in the next section.

5.3 Correlation based on ICMP Type and Code

As the remaining alerts majorly comprise of the ICMP packets, a correlation technique to shortlist ICMP types and codes is proposed. The Internet Control Message Protocol (ICMP) is one of the most important protocols of the IP Layer in the TCP/IP Protocol suite. It functions as one of the adjunct protocols, alongside the Internet Protocol. One of the main functions of the ICMP is for the IP in one router/gateway to send error messages and other control messages to the IP in another router/gateway [37]. Some of the common error and control messages are destination router cannot be reached or service is unavailable or a simple ping packet.

Based on the purpose and function of the ICMP messages, they are identified by a particular type and code. When it comes to Intrusion, an attacker can intrude a system using an ICMP packet. Therefore, it is quite essential for an IDS to report the in-
trusions based on ICMP. However, most of these ICMP packets could be nothing more than a harmless echo reply and these ICMP packets could end up being a false positive.

A ping packet is generally blocked by firewalls as an attacker can easily find out more about the network and the ports. When a system is pinged by an attacker, the reply from the system provides valuable information about whether the port is open or not. Some ping packets come back with additional information such as the version of the OS being used, the time take for the ping to reach the destination and to send back a response and so on. This information would hint the intruder with ideas and ways in which he could break all the security measures.

There are two other ways in which an ICMP packet can be used to perform dangerous tasks, apart from gaining normal information about an open or closed port, as explained above. These are the ping of death and smurf attacks [8].

Ping of Death (POD) – This is a type of Denial of Service attack and is known as ICMP Ping Flood Attack as well. In this attack, the attacker has more bandwidth than the receiver does. A ping packet is an ICMP Echo Request packet and is generally generated to a system in order to receive a response known as the ICMP Echo Reply packet. An attacker who is attempting an ICMP Ping Flood DoS Attack generates a substantial amount of ICMP Echo Request (ping) packets that are targeted towards the destination. The destination system, which has a lesser bandwidth than the attacking system, is overwhelmed by such a high volume of packets and tries issuing ICMP Echo Reply packets for every ping. Therefore, both the outgoing and the incoming bandwidth of the destination system would have been consumed to its fullest capacity. As a result, the destination system that had been a targeted victim slows down in its operation and at the worst case, denies its service to anybody to requests. This has been one of the most commonly used methods by several attacking groups to initiate a Denial of Service Attack against the servers of many reputed websites [8].

Smurf Attack – A smurf attack is another type of a Denial of Service Attack where spoofing is required. Spoofing is the process of masquerading as another system by replacing the true IP address with that of targeted system. The attacker sends an ICMP Echo Request packet with the spoofed IP address of the victim to the IP broadcast address in the network. The broadcast address sends an ICMP Echo reply packet to the actual victim. The victim tries to send a Refresh message to the IP broadcast address suspecting a security breach. However, the attacker blocks the victim from sending a response to the broadcast IP address and continues sending a flood of ping packets. The broadcast IP address continues to send ICMP Echo reply packets to the victim and exceeds the incoming bandwidth capacity of the victim. As a result of this, the victim slows down and denies access to the legitimate users who try to access this machine [8].
As can be seen in the figure above, the ICMP Packet is 32 bits in total and comprises of three major fields namely type, code, and checksum.

- **Type** – this field describes what the function of an ICMP packet is. There are 41 types of ICMP packets in total and each ICMP packet is allotted a number ranging from 0-40 based on its type [42]
- **Code** – this field describes further describes what precise action each type performs in particular. Each type has varying numbers of codes [42]
- **Checksum** – this field is used to detect error on the ICMP packet by the destination side.

This ICMP datagram is encapsulated within an IP datagram and sent to inform the destination system of a possible error or a control message.

### 5.3.1 Correlation Process

The MySQL database logs details regarding the ICMP packets in a separate table named as icmphdr. The figure below displays the SQL statement that can be used to view the details in the icmphdr table:

```
select * from icmphdr;
```
The icmphdr table below displays the contents of the ICMP header of the packets logged as alerts by SNORT. The columns of relevance and importance to this correlation technique are the icmp_type and icmp_code. It is based on these fields that a decision would be made if an ICMP packet could be malicious or otherwise.

![Figure 27: icmp table](image)

There is a set of ICMP Types and Codes, which must not be allowed to enter or exit the system. These are listed below:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP_ECHO</td>
<td>8</td>
<td>0</td>
<td>Ping</td>
</tr>
<tr>
<td>ICMP_ECHOREPLY</td>
<td>0</td>
<td>0</td>
<td>Ping response</td>
</tr>
<tr>
<td>ICMP_UNREACH</td>
<td>3</td>
<td>4</td>
<td>ICMP Unreachable with don’t fragment</td>
</tr>
<tr>
<td>ICMP_TIMXCEED</td>
<td>11</td>
<td>0</td>
<td>TTL Expired in Transit</td>
</tr>
</tbody>
</table>

![Figure 28: ICMP Packets that must be blocked](image)

Routers and firewalls must be configured to restrict the flow of the ICMP packets of the above type and code back and forth. However, in this project, firewalls have not been enabled and hence, a correlation would be required to segregate the real dangers from the rest. MySQL Statements to separate the above types and codes for further investigation will be proposed in this section of correlation.

For instance, a SQL statement has been suggested to separate icmp_type=0 and icmp_code=0 from the rest of the database. As an extension to the current proposal, a future research can be carried out to investigate these segregated ICMP ping packets based on the number of alerts logged, the timestamp of these alerts and the ip_src address from which these have originated.

```sql
SELECT icmphdr.icmp_type, icmphdr.icmp_code, iphdr.ip_src FROM icmphdr natural join where icmp_type=0 and icmp_code=0;
```
The above statement gives rise to the list of IP addresses that generate the ICMP packets of the type and code 0. A total of 356 alerts are correlated which belong to type 0 and code 0 and can be sent in for further examination. Similar to this example, the other types and codes mentioned in the Figure 28 can be separated from the rest.

![Figure 29: ICMP Type 0 Code 0 Correlation](image)

5.3.2 Result

As a result of this correlation process, the ICMP packets, with type=0 and code=0 that could be true are separated from the remaining harmless ICMP packets. Further investigation on these packets could be attempted.
6 Chapter 6

Conclusion and Future Work

As described earlier, the different components of SNORT functioned appropriately to achieve the detection of intrusions when the DARPA trace file was executed. Each packet was synthetically produced by the trace file and was intended to trigger SNORT into generating alerts. Referring back to the timestamps in the output, the year of alert detection is 1999, which shows that the packets are synthetic for evaluation purposes.

As a result, random packets, inclusive of both malicious as well as harmless packets were fabricated by the trace files to evaluate the accuracy in detection of true positives. The latest DARPA trace files are not easily available and hence the ones used for 1999 evaluation had been considered for testing SNORT. Since the effectiveness of SNORT in detection, as a result of intrusions via trace files is not very high, correlation techniques were implemented to increase efficiency as a part of the post-detection scenario.

A summary of the results shown in the previous chapter along with a brief description of how each correlation technique had played a part in filtering false positives is explained in the next section. A few other correlations that were not proposed in the previous chapters, but were attempted, have been listed for future recommendations. An insight into how these correlation techniques can be attempted by future researches has been proposed instead in the future recommendations and appendix sections.

6.1 Conclusion

The SNORT IDS was installed and its working and detection capacity was tested using Nessus. The DARPA 1999 Trace files were further used to evaluate the detection efficiency of SNORT. 57,912 alerts were produced in total, as a result of a single test. The large amount of alerts proved the weakness in the detection efficiency of SNORT by allowing false positives to be included.

This project proposed three correlation techniques, which were used to increase the efficiency of SNORT as an aftermath of its detection process. The first two correlation techniques dealt with separating the false positives from the entire database and the final technique dealt with separating the true positives from the alerts left behind after the first two correlations. The first technique, which was the three-level
correlation model, showed a systematic grouping process of the alerts present in the MySQL database and eventually, separating all the IP addresses involving a single alert generation. This technique filtered 169 single alerts from the entire database of 57,912 alerts. This implies that 169 IP addresses were responsible for generating 169 single alerts. These alerts were removed from the database as they were considered false positives. A future work to the three-level model is suggested in the next section.

The second correlation technique is that which is based on Signature and Timestamp examination. In this method, signatures were closely inspected along with their time of generation, in order to justify whether the alerts were false or not. Signatures 32 and 43 were inspected and based on their timeframes and IP addresses, they were termed as false positives. Therefore, another 6 alerts generated by two IP addresses and 43,693 alerts generated by another two IP addresses were removed from the database. By the end of this correlation process, a total of 44,693 alerts have been removed from the database, leaving behind only 13,044 alerts. This removes nearly 80% of the false positives. A further expansion of this idea is provided in the next chapter.

The third correlation technique involved justifying which alerts could be true positives by examining the ICMP packet type and code. The ICMP types and codes listed in Figure 28 should generally be blocked with the help of a firewall as these could end up as malicious probe packets. Therefore, correlation is performed to separate ICMP packets belonging to these types and codes from the rest of the ICMP packets. The rest of the ICMP packets are discarded as false positives, whereas the separated ones can be further explored and researched. 356 alerts were correlated for Type 0 and Code 0 alone. The same technique can be employed to correlate the other Types and Codes listed in Figure 28 as true positives.

The three suggested correlation methods have been attempted with another DARPA trace files used for the 1999 evaluation process. The total alerts recorded were 58,052. After implementing the correlation methods, approximately 76% of alerts were correlated as false positives. This proves that the correlation techniques work for all trace files and are not biased to a particular data set.

6.2 Future Work and Recommendations

Suggestions on how the current correlation techniques can be improvised in the future have been postulated below:

- The three-level correlation model can be further enhanced by individually experimenting the signatures of all the single alerts that have been separated out. Any unique signature that appears to be false can be used to further correlate
within the single alerts database to separate out those with the same signature. This correlation would aid in determining, which single alerts are genuinely true positives and which ones are genuinely false.

- The second correlation process can be further enhanced by taking the IP addresses and timestamps into consideration. An attempt had been made to determine how many attacks had been initiated for every minute (Refer Appendix for Output and Code). This information can be used to further determine which of the IP addresses generate the alerts in the first minute, second minute and so on. Based on the timing, correlation can be carried out to determine if the alerts constitute a part of the same attack or a different one. The signatures can be used as well to determine, if the alerts within a minute describe different attacks or the same one.

- The same correlation techniques can be tested on other datasets, apart from DARPA to determine whether the proposed methods result in similar outcomes.

- The proposed correlation techniques were used only as a post-detection process and not as a part of the detection. Future research can be carried out to integrate the correlation techniques as a part SNORT, so that it can correlate the alerts while it detects and produces a correlated output.

- Correlation can be based on the priority of the signature. An attempt has been made to correlate alerts according to their priorities. All the IP addresses whose priorities are 1 are separated from the rest. Further analysis can be made to study the signature of the correlated IP addresses in detail. (Refer Appendix).
References


[19] Dr. M. J. Reed, CE823 Network Security Notes, 2010-2011, Computer Science and Electronic Engineering (School of), University of Essex.


[37] Dr. N. Zachlenuik., CE707 IP Networking and Applications, 2010-2011, Computer Science and Electronic Engineering (School of), University of Essex.


[52] Dr. M. J. Reed, CE708 Computer Security Notes, 2010-2011, Computer Science and Electronic Engineering (School of), University of Essex.

[53] Dr. M.J. Reed, “Scripts for Barnyard, SNORT and Correlation”, Computer Science and Electronic Engineering (School of), University of Essex.

Appendix

Description of packages supporting SNORT:

Apache 2: The Apache server is required so that the alerts that have been logged in the MYSQL database can be viewed in a web page in the form of a report with graphical representation of the types of alerts in a user-friendly format [41].

Php5: The Pretty Home Page Scripting Language acts along with GD and Apache 2 in displaying the alerts in the form of a report in a webpage. The PHP version being used in this project is 5 [41].

Php5-gd: The GD Library is also known as the Graphic Display library and is used for the graphical setup of the SNORT report webpage.

Php5-mysql: This command is required to provide PHP access to the MySql database.

libpcap0.8-dev: pcap represents packet capture. The library used to perform this low-level network monitoring is known as libpcap. The version being used is 0.8.

libpcre3-dev: The libpcre package is used for regular expression pattern matching and follows a syntax similar to Perl 5 [30].

Bison: Bison is a language parser that is required along with flex and groups words into sentences and sentences into paragraphs. This, along with Flex, aids SNORT during rule matching and analysis.

Flex: Flex is similar to Bison and acts as a code-generating tool. It performs the reverse operation of Bison by splitting characters into tokens.

Wireshark: Wireshark is a packet-capturing tool. Installation of wireshark is required in order to determine the rate at which the malicious traffic enters the port.

Python Program for First-Level Correlation: [54]

```
#!/usr/bin/python
# apt-get install python-mysqldb

import MySQLdb as mdb
import sys

try:
    conn = mdb.connect('localhost', 'snort', 'university', 'snort');
```
cursor = conn.cursor()
cursor.execute("select ip_src, count(*) from iphdr group by ip_src")
rows = cursor.fetchall()

for row in rows:
    print row

cursor.close()
conn.close()

except mdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
sys.exit(1)

Python Program for Second-Level Correlation: [54]

#!/usr/bin/python
# apt-get install python-mysqldb

import MySQLdb as mdb
import sys

try:
    conn = mdb.connect('localhost', 'snort', 'university', 'snort');;
cursor = conn.cursor()
cursor.execute("select ip_src, count(*) as Number from iphdr group by ip_src order by number")
rows = cursor.fetchall()

for row in rows:
    print row

cursor.close()
conn.close()

except mdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
sys.exit(1)

Python Program for Third-Level Correlation: [54]

#!/usr/bin/python
# apt-get install python-mysqldb

import MySQLdb as mdb
import sys

try:
conn = mdb.connect('localhost', 'snort', 'university', 'snort');
cursor = conn.cursor()
cursor.execute("select ip_src, count(*) as Number from iphdr group by ip_src having number<2 order by number")
rows = cursor.fetchall()

for row in rows:
    print row

cursor.close()
conn.close()

except mdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
sys.exit(1)

Script to Clear Alerts from MySQL Database:

#!/bin/bash
# This is a simple script that will prune all but the last 20 days of
# data from the snort DB.

/usr/bin/mysql --user=snort --password=university snort <<EOF
USE snort;
DELETE FROM event WHERE timestamp < DATE_SUB(NOW(),INTERVAL 20 DAY);
DELETE FROM data USING data LEFT OUTER JOIN event USING (std,cid) WHERE event.std IS NULL;
DELETE FROM iphdr USING iphdr LEFT OUTER JOIN event USING (sid,cid) WHERE event.sid IS NULL;
DELETE FROM tcp hdr USING tcp hdr LEFT OUTER JOIN event USING (sid,cid) WHERE event.sid IS NULL;
DELETE FROM udp hdr USING udp hdr LEFT OUTER JOIN event USING (sid,cid) WHERE event.sid IS NULL;
DELETE FROM opt USING opt LEFT OUTER JOIN event USING (sid,cid) WHERE event.sid IS NULL;

#DELETE FROM acid_event USING acid_event LEFT OUTER JOIN event USING (sid,cid) WHERE event.sid IS NULL;
#DELETE FROM ag USING acid_ag_alert AS ag LEFT OUTER JOIN event AS e ON ag.ag_sid=e.sid AND ag.ag_cid=e.cid WHERE e.sid IS NULL;

#OPTIMIZE TABLE event, data, iphdr, tcp hdr, udp hdr, opt, acid_event, acid_ag_alert;
OPTIMIZE TABLE event, data, iphdr, tcp hdr, udp hdr, opt;
EOF

Figure (i) [49]
### Signature Table:

<table>
<thead>
<tr>
<th>sig_id</th>
<th>sig_name</th>
<th>sig_class_id</th>
<th>sig_priority</th>
<th>sig_rev</th>
<th>sig_sid</th>
<th>sig_gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POLICY download of a PDF with Operation object</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>10862</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>ICMP Destination Unreachable Port Unreachable</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>452</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>ICMP Timestamp Request</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>453</td>
</tr>
<tr>
<td>4</td>
<td>ICMP Timestamp Reply</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>451</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>ICMP Address Mask Request</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>388</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>SPYWARE-root Hacker Tool Tidbitsk pro runtime detection - udp port 467</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>35817</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>SCAN: Amadeo client version request</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>234</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>FTP GLOBAL client request</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>1432</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>FTP Outbound GLOBAL client request</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>556</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>SHELLCODE x86 inc exo NOP</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>12</td>
<td>1364</td>
</tr>
<tr>
<td>11</td>
<td>SHELLCODE x86 inc exo NOP</td>
<td>11</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>1366</td>
</tr>
<tr>
<td>12</td>
<td>ICMP FIND</td>
<td>12</td>
<td>2</td>
<td>5</td>
<td>384</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>ICMP Echo Reply</td>
<td>13</td>
<td>3</td>
<td>3</td>
<td>480</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>ICMP FIND undefined code</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>365</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>ICMP Echo Reply undefined code</td>
<td>15</td>
<td>2</td>
<td>3</td>
<td>480</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>ICMP Information Request</td>
<td>16</td>
<td>2</td>
<td>3</td>
<td>417</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>WEB-NICE HP Overview NWM freeDesires.mpl Unix command execution attempt</td>
<td>17</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>8698</td>
</tr>
<tr>
<td>18</td>
<td>stream: Web outside window</td>
<td>18</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>http_inspect: NON-FPC DEFINED GMR</td>
<td>19</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>http_inspect: U ENCODING</td>
<td>20</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>sensitive.data: sensitive data global threshold exceeded</td>
<td>21</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>sensitive.data: sensitive data - email addresses</td>
<td>22</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>Short Alert [1.21.322-28]</td>
<td>23</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>15382</td>
</tr>
<tr>
<td>24</td>
<td>http_inspect: NO CONTENT LENGTH OR TRANSFER ENCODING IN HTTP RESPONSE</td>
<td>24</td>
<td>18</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>stream: TCP Timestamp is missing</td>
<td>25</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>stream: ACK number is greater than prior FIN</td>
<td>26</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>ATTACK RESPONSES invalid URL</td>
<td>27</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>1298</td>
</tr>
<tr>
<td>28</td>
<td>ftp_g: FTP parameter length overflow</td>
<td>28</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>29</td>
<td>ftp_g: FTP malformed parameter</td>
<td>29</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>ftp_g: invalid FTP command</td>
<td>30</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>ftp_g: Possible string format attempt in FTP command/parameter</td>
<td>31</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>ICMP Destination Unreachable host unreachable</td>
<td>32</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>388</td>
</tr>
<tr>
<td>33</td>
<td>http_inspect: LONG HEADER</td>
<td>33</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>stream: TCP window opened before receiving data</td>
<td>34</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>Short Alert [1.21.547-48]</td>
<td>35</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>36</td>
<td>stream: TCP Small Segment Threshold Exceeded</td>
<td>36</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>CMT TFC nick change</td>
<td>37</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>CMT TFC channel join</td>
<td>38</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1728</td>
</tr>
<tr>
<td>39</td>
<td>CMT TFC message</td>
<td>39</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1463</td>
</tr>
<tr>
<td>40</td>
<td>ATTACK RESPONSES 403 Forbidden</td>
<td>40</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>1291</td>
</tr>
<tr>
<td>41</td>
<td>SHELLCODE x86 inc exo NOP</td>
<td>41</td>
<td>4</td>
<td>1</td>
<td>20</td>
<td>668</td>
</tr>
<tr>
<td>42</td>
<td>WEB CLIENT Microsoft Internet Explorer Script Engine Stock Exhaustion Denial of Service attempt</td>
<td>42</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>17467</td>
</tr>
<tr>
<td>43</td>
<td>sst: Protocol mismatch</td>
<td>43</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>XIIC logor</td>
<td>44</td>
<td>18</td>
<td>3</td>
<td>7</td>
<td>1226</td>
</tr>
<tr>
<td>45</td>
<td>ICMP FIND DEFType</td>
<td>45</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>368</td>
</tr>
</tbody>
</table>

Figure (ii)
Correlation of Signature 43: [54]

#!/usr/bin/python
# apt-get install python-mysqldb

import MySQLdb as mdb
import sys

try:
    conn = mdb.connect('localhost', 'snort', 'university', 'snort');;
    cursor = conn.cursor()
    cursor.execute("select event.signature, iphdr.ip_src from event natural join iphdr where event.signature=43")
    rows = cursor.fetchall()

    for row in rows:
        print row

    cursor.close()
    conn.close()

except mdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    sys.exit(1)

Note: The same code is used for correlating signature 32 as well by changing the signature number.

Extension to Present Work:

(i) Python Program to Count Number of Attacks in One Minute [54] [53]

#!/usr/bin/python
# apt-get install python-mysqldb

import MySQLdb as mdb
import sys
import datetime
from datetime import timedelta

try:
    conn = mdb.connect('localhost', 'snort', 'university', 'snort');;
    cursor = conn.cursor()
    cursor.execute("select event.timestamp from event")
    rows = cursor.fetchall()

    count = 0
    total = 0
    startime = rows[1][0]
print('start time is',starttime,'/n')
for row in rows:
    print row
    alerttime = row[0]
    total = total + 1

    if alerttime > starttime + timedelta(minutes=1):
        print (count)
        count = 0
        starttime = alerttime
    else:
        count = count + 1

cursor.close()
conn.close()
print("total alerts=",total)
except mdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
sys.exit(1)

Output:

![Image of output]

Figure (iii)

(ii) SQL Statement for Correlation based on Sig_Priority

```
select signature.sig_priority, iphdr.ip_src from signature natural join iphdr where sig_priority=1;
```

Figure (iv)
## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Course</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Amsterdam August 2020 Part 1</td>
<td>Amsterdam, NL</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS FOR508 Canberra August 2020</td>
<td>Canberra, AU</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam August 2020 Part 2</td>
<td>Amsterdam, NL</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020</td>
<td>Virginia Beach, VAUS</td>
<td>Aug 31, 2020 - Sep 05, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Philippines 2020</td>
<td>Manila, PH</td>
<td>Sep 07, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London September 2020</td>
<td>London, GB</td>
<td>Sep 07, 2020 - Sep 12, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Baltimore Fall 2020</td>
<td>Baltimore, MDUS</td>
<td>Sep 08, 2020 - Sep 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Threat Hunting &amp; Incident Response Summit &amp; Training 2020</td>
<td>New Orleans, LAUS</td>
<td>Sep 10, 2020 - Sep 17, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>OnlineUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>