Network Forensics and Log Files Analysis: A Novel Approach to Building a Digital Evidence Bag and Its Own Processing Tool

Supervisors: Ray Hunt, Malcolm Shore

A thesis submitted in partial fulfilment of the requirements for the Degree
of Master of Computer Science and Software Engineering
at the University of Canterbury

Ahmed Qaisi

University of Canterbury
Department of Computer Science and Software Engineering
September 30, 2011

Abstract

Intrusion Detection Systems (IDS) tools are deployed within networks to monitor data that is transmitted to particular destinations such as MySQL, Oracle databases or log files. The data is normally dumped to these destinations without a forensic standard structure. When digital evidence is needed, forensic specialists are required to analyse a very large volume of data. Even though forensic tools can be utilised, most of this process has to be done manually, consuming time and resources. In this research, we aim to address this issue by combining several existing tools to archive the original IDS data into a new container (Digital Evidence Bag) that has a structure based upon standard forensic processes. The aim is to develop a method to improve the current IDS database function in a forensic manner. This database will be optimised for future, forensic, analysis.

Since evidence validity is always an issue, a secondary aim of this research is to develop a new monitoring scheme. This is to provide the necessary evidence to prove that an attacker had surveyed the network prior to the attack. To achieve this, we will set up a network that will be monitored by multiple IDSs. Open source tools will be used to carry input validation attacks into the network including SQL injection. We will design a new tool to obtain the original data in order to store it within the proposed DEB. This tool will collect the data from several databases of the different IDSs. We will assume that the IDS will not have been compromised.

Acknowledgement

First and foremost all thanks go to Allah. I would also like to thank every one who made this thesis possible starting from my supervisor to all relatives and friends.

I am indebted to my parents who deserve special mention for their inseparable support and prayers. It was my Father, Abdulrheem Jraibi Qaisi, who made learning a fundamental part of my character. Ever since I was a child it was he who showed me how a person must work towards his goals with passion and patience not only through his words but also by his own actions when studying and working. Two words he always told me which I will never forget; faith and sincerity. My Mother, Zahra Aljabali who lovingly raised me with her caring and gently love. Mohammed, Abkr, Norua, Abdullah, Safiya, Fatima, Somiah and Asmaa, thanks for being supportive and caring siblings. I also owe my deepest gratitude to my New Zealand mother Patricia Smith for the unlimited support and love she gave me when I was living with her and afterwards. I also thank her for proof reading the thesis an every word she had to check for me. Thank you Mum.

Words fail me to express my appreciation to my wonderful wife Amnah whose dedication, love and care, has taken such a load off my shoulders. I owe her so much for being supportive and patient during those long nights I stayed up working.

I am proud to express my sincere gratitude to Ray Hunt for his supervision, advice, and guidance from the very early stage of this research as well as giving me extraordinary knowledge throughout my time at Canterbury University. This thesis would not have been possible without Ray's help. Thank you Ray.

It is also pleasure to thank those who made this thesis possible especially my dear friend Muteb Alqahtani, Sachin Sebastian, Lim Lin and Michael

Paerce for sharing their knowledge. Many thanks also to those good friends who kept in touch and always asked about me all the time I have beenin New Zealand; Yahya Alameer, Mahmoud Hadad and Ali Sawadi. You are good friends.

Finally, I would like to thank everybody who was important to the successful realization of this thesis, as well as expressing my apology that I could not mention each of you personally one by one

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

etwork security and forensics include the action of monitoring and recording data in transit in order to discover potential attacks. Intrusion Detection Systems (IDS) tools are deployed within networks to perform these actions. The captured data is then examined and analysed possible extraction of the digital evidence. However, analysing such data to produce the evidence can be exceedingly difficult and time consuming. This is because data is always dumped to a log file or databases without a forensic standard structure. Providing auditing for training purposes can also provide difficulties. For instance, when performing a security analysis of the data, system administrators do not have sufficient time to manually record their actions which could be used for these learning and training purposes.

While there are several forensic tools available (mostly proprietaries), most of these must be done manually consuming time and resources. In this research, we aim to address this issue by combining several existing tools. These tools will consist of multiple entities (hosts, IDSs, databases, firewalls and attackers). The core purpose of this thesis is to archive the original IDS data into a new single container from which a unique Digital Evidence Bag (DEB) will be created. As well as these potential sources, command line tools

such as “Net-state”, “dig”, “nslookup”, “traceroute“, ”Whois“...etc can be also used to put information into the newly created DEB at the same time. The created DEB will have a structure based upon standard forensic processes. The aim is to develop a method to improve the current IDS database forensic function.

Since the validity of evidence is always an issue, a secondary aim of this research is to develop a new monitoring scheme. This is to provide the necessary evidence to prove that an attacker had surveyed the network prior to the attack. To achieve this, we will develop a testbed architecture for this thesis. Further, open-source tools will be utlised to obtain the original data from the IDSs' databases in order to store it within a combined database. Then a tool is designed to connect to the combined database from which a wrapped DEB is produced with digital evidence. This tool can collect the data from several databases of the different IDSs. These IDSs are deployed at different security levels of the testbed architecture. Their output (databases) will be compared for any potential matching that could strengthen the validity of the evidence produced in any court action against the attacker.

1.2 Motivation

The predominant motivation of this thesis is the shortage of research in the in the combined area of security and forensics. Lookingat them from a practical perspective it could be said they are even one entity. If a system needs to be highly secured it also needs to be structured in such a way to enable investigators to collect evidence. No entity is completely secure and so when security is violated, there must be a clear system for collection of the evidence. Not all systems will be the same and what system is appropriate will depend on the type of attack and the type of investigation required. However, investigators can take the initiative and build their own systems within a forensic environment.

1.3 Network Security and Forensics

Networks are exposed to internal and external threats. These threats can use the victim's network as a base for launching attacks on associated networks such as denial of service (DoS). Another potential threat can be an alteration of information through the victim's network. Internal trusted participants can also launch attacks on the network by abusing their level of privilege[1]. This requires the development and maintenance of proper situational awareness.

Network forensics is defined as the monitoring, recording, and analysis of network traffic and events. It is performed in order to discover the source of security breaches as well as providing information to assist in the response to recovery from attacks or other potential problems. One key role of the forensic expert is to differentiate repetitive problems from malicious attacks[1],[2],[3].

There are three groups of people who are interested in the network forensics area: law enforcement, operators of critical infrastructure and education systems. The two main types for network forensics are:

1. General network forensics
2. Strict network forensics

The purpose of general network forensics is to obtain malware attack signatures and utilises them for an intrusion detection system (IDS) and as an aid to firewall configuration. This is to enhance the security posture within the network; while the strict network forensics obtains the evidence to be used in a court of law[2],[4]. Currently, forensic methods are known as dead forensic (after the cybercrime fact); and live forensic (during the cybercrime fact). This thesis proposes to extend current network forensic techniques by the introduction of a DEB in order to enhance analysis techniques.This is shown in Figure 1.1.

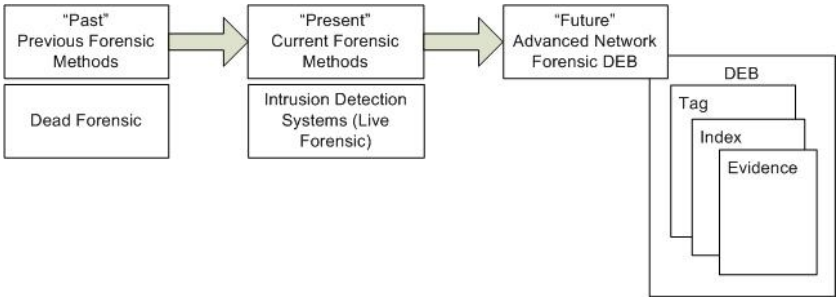


Figure 1.1: Logical Methods in the Development of Forensic Practices

The condition of multiple IDSs and network forensics has great potential to address the internal and external threats described above. In particular, the use of multiple IDSs can trigger network forensics tools in order to provide an analysis of data flows into and out of a network. IDS problems exist with the processes of aggregating audit logs and maintaining signatures. The overlay of network forensic onto IDSs will assist us in addressing these key problems[5],[6].

The limitation of many forensic audits lies in the lack of real-time (live) forensic analysis. Normal alerts can indicate the potential for more sophisticated attacks. For example, network scanning attacks are sufficient for attackers to collect useful information about the network which enable them to launch further attacks[5]. Such alerts should be categorised and reported at different stages of their occurrence.

1.4 Evidence Capturing Techniques

1.4.1 Live Incident Response Forensic

When acquiring files selectively from a live system, greater attention should be given than when creating a forensic duplicate. This is because when acquiring files from a live system action taken may alter the original evidence. Acquiring data from a system in a live-forensic mode is often necessary to

determine whether a malicious code has been installed. It is also important to gather volatile data at an early stage of a malware incident. This can provide valuable leads, such as details about remote servers that the malware is communicating with.

There are various tools in the operating systems themselves which can be used to obtain volatile data. For instance, Linux commands are useful for collecting volatile data from a live system. However, one can argue that if a system can be compromised by malware it cannot be trusted. Therefore, in reality it can be said no system can be trusted. This makes it necessary to use a toolkit of utilities for capturing volatile data that has minimal interaction with the subject operating system. The use of such utilities is critically important and can reveal hidden information by a rootkit.

In this thesis, an overall methodology for preserving volatile data will be briefly demonstrated on a Linux machine in a forensically sound manner.(See Chapter 5). This will be implmented by using a case example within the proposed testbed to demonstrate the strengths and shortcomings of useful linux commands from a forensic perspecive. These commands must be tested and assessed before trying to achieve the goal of combining them into one group.

Volatile Data Collection Techniques

Existing tools and commands can be utilised when conducting a live forensic technique to collect volatile data such as uptime, date, time, users logged into the system, open ports and listening applications, lists of currently running processes, registry information, attached devices (this can be important when having a wireless attached device which may not be obvious at the crime scene) and command history for the security incident. While conducting this process, it is crucial to generate the date and time to provide an audit trail. The following are some of the useful Linux commands that can be used to obtain live evidence although they tend not to perform in a foreniscally

sound manner.

Script: The majority of UNIX systems have a script utility that can record commands that are run as well as the output of each command, providing the supporting documentation that is the cornerstone of digital forensics. The script catches data in memory and writes the full recorded information when a process is terminated. By default, the script commands save data to the current location.

Who: Identify users logged onto the system. Use who or w to determine who is currently logged in. Verify that a legitimate user established each session.

Netstat: Determine network connections and activity. Use netstat to view open connections to the computer.

Ps: Use ps to view the processes running on the computer, and try to determine if any unusual processes are running.

Lsof: Use lsof to determine what files and sockets are being accessed.

1.4.2 Typical Capturing Evidence Processes

Selective Capturing Process

Selective imaging is a means of acquiring the evidence on a selective basis. When acquiring evidence, a decision should not be made to acquire the whole information of a media but only the required (relevant) amount [7]. This has not always been so, but now, according to official good practice guidelines of The Association of Chief Police Officers in the UK (ACPO) it is now recognised that "partial or selective file copying may be considered as an alternative" when it may not be practical to acquire everything. For instance, when the amount of data to be imaged makes this impracticable. However if

this approach is adopted, investigators should ensure they are capturing all relevant evidence [8].

According to [7], there are three types of selective imaging. They are:

1. Manual selective imaging: Is when the investigator manually specifies the required files in order to obtain them.

2. Semi-automatic selective imaging: Is when the investigator specifies a particular type of files or few categories. This can be based on the file extension, file hash or file signature (e.g hex signature).

3. Automatic selective imaging: Is when the investigator specifies the source and destination of the evidence required and commences the automatic capturing process.

When obtaining evidence by any of the aforementioned (selective imaging processes), one of the main difficulties encountered is recording the provenance of an item of information. In order to record such information there are some metrics that can be used to locate a file. This location can be specified by one of the following modes [7]:

1. The root folder including partition reference number
2. The logical cluster of the required file
3. The physical sector of the required file on the disk

One of the things to bear in mind with these modes is the attributes of provenance must meet the following criteria [9] and [7]:

1. Unique
2. Unambiguous
3. Concise

4. Repeatable

It can be argued that in its own way each method meets these criteria. It will depend upon the technical knowledge of the person trying to understand it. For example the general public, including even a judge or legal professional is likely to be more familiar with a folder location than a more technical disk sector or cluster reference.

Intelligent Capturing process

Intelligent imaging which is the process of embedding the knowledge of experts into an intelligent system is a good option for investigators who are not technically proficient. In order to obtain the relevant evidence this technique allows the investigator to select the type of inquiry that is being conducted [7]. However in the case of a fraud or intellectual property theft, the investigator may not be able to recognise the type or location of the required files. The question then becomes how can the expert's level of intelligence be recognised and thus be embedded in a tool. Another question is whether all the relevant data can be captured by the tool.

Digital Evidence Bags (DEBs)

The traditional forensic process is the process of capturing an image of the original material. This capture of the evidence can be either in static or real-time - 'live' forensics. A new concept for a container for digital evidence has been recently introduced. This can be used as a wrapper which provides professionally obtained evidence and an audit trail of previously performed actions.

When obtaining such evidence the actual forensic task is to capture an image of the original media. There are two problems that relate to the actual containers that contain the captured information:

1. The tool has to process and analyse the captured forensic image as a single entity.

2. The forensic utility captures the information into different format containers. That is not to say a single format container should not be used to capture evidence but the wrapper (which is the DEB) used must be consistent when capturing and storing information[10].

It is not uncommon to see a single log file from an architecture of the form being modeled in this thesis to be of 350 Gb of data over one week. This is compounded by the fact that forensic tools currently in use are being stretched beyond their capabilities. This results in the whole network forensic process becoming problematic [11],[10]. The situation is still difficult even when taking into account the diverse number of devices that process digital information and which are capable of having digital information extracted from them. This means that forensic practitioners have to learn, understand and use even more specialised applications in order to capture the required information[10].

In a networking environment, tracing attacks or analysing system problems could lead to losing valuable information. This could occur in a case of information that had been lost or stolen. Such a problem can happen when an incident investigator misses the opportunity of recording some valuable information that was discovered. This opportunity can often not be repeated. Therefore, people in charge of the system including System Administrator (SA)[1] is in a position of possibly neglecting to obtain important information [9].

System Administrators have various tools to investigate a system. These tools are mostly console application and run as command line utilities. The

[1]must be highly trained so that they can cover all the processes, tasks or operations performed by the system administrator, system operator, incident investigator, security administrator, network investigator. Anyone in the role of determining the problem, the cause or effect of any abnormal or unusual system behaviour

problem with these tools is is that they are not able to record either the preformed actions or the fact that these actions were taken or even the results obtained from them. From a forensic perspective these tools do not provide information in a forensically sound manner. One can argue that the output can be logged into a log file; however, this does not insure the integrity mechanism in transit. In fact, it is not common or an easy job to record every action taken or results obtained when examining a system [9]. Even taking notes can be extremely difficult when real live attacks are in process.

The concept of a DEB can be used in order to address the aforementioned issues. This is not to say all tools have to be changed but from a forensic perspective, this should take place. The DEB will enable Systems Administrator to record information and provide a continuity section to be maintained throughout the life of the investigation. DEBs were mainly designed for the traditional role of static digital forensic investigations [9]. Within this role DEBs allow more data capture methodologies including selective and intelligent capturing techniques [7]. Even more so, DEBs can be used to store forensic images of command line utility output, digital media, memory dumps, network packet captures and associated meta-data.

The DEB forensic incident response tool permits command line applications to be executed from a special dialogue box. When the command is executed the output from each command is captured in the DEB together with an integrity hash of the data and a time stamp of when the command commenced and completed. When all the relevant system information has been captured the DEB is closed and sealed [9].

Traditional Evidence Capture

In the real life of law enforcement, when a crime scene is being investigated, many elements are brought to the laboratory. It is, although not always, possible to take away a whole physical crime scene. However, it rarely happens

that the investigators dismantle bricks of a building for further investigation. In contrast, within digital life, forensic investigators are able to capture every potential evidence from the suspected crime scene. This is the advantage of the world of digital forensics. The data then is sealed at the scene with a seal number and, as well, a tag is attached with details such as the following:

1. Property reference number
2. Case/Suspect name
3. Brief description of the item
4. Date and time the item was seized/produced
5. Location of where the item was was seized/produced
6. Name of the person who is taking custody of the evidence
7. Incident/Crime reference

The tag may also include "continuity sections" in which the details of all the investigators involved are recorded at the time of their actual investigation of the evidence (chain of custody). This is to ensure that the data is being recorded from the time the item was seized to the time it is being shown as a legal evidence. This section shows the following details of the person who takes custody of the item:

1. Name/Rank
2. Signature
3. Date and time when the custody is being taken by the person.

Bags of different size, type and shape can be sealed at a crime scene. The actual number will depend on the size of the captured data and its type. However, using a consistent wrapper allows other specialist laboratories to process the item. For instance, some items may require DNA analysis while others may require fingerprint analysis or just an interpretation of their contents by a particular specialist. All of these depend on using a consistent bag

wrapper. The question becomes how to create a consistent bag wrapper in a radically changing digital world such a method can be applied in a radically changing digital world [10].

Digital Evidence Capture

At present, the processes of "dd"[2] image or the proprietary format produced by the forensic tool vendors, are the equivalent to the physical evidence capture process. Performing the "dd" file raw capture, there is no defined technique for attaching basic forensic details such as date and time of capture, name of the person who carried on the process or even any method for integrity check [10]. It is however possible to include such details manually or perform some integrity check separately (e.g. md5 hash). However, this can be extremely difficulty when dealing with a real-time evidence equation.

Some proprietary forensic tools allow users to enter details at the beginning of the capture process. They also enable users to generate a hash to maintain the integrity of evidence. These techniques tend to capture the whole evidence into a single file (one bag), which can become a problem if the size of evidence is too large. This problem requires a fragmentation of chunks from which the file is later backed up. Another problem could occur when the capacity of a single storage device is insufficient and will thus require using a number of devices for storage proposes. In order to be able to process the content of either of these types of data capture output, the fragmented files must be combined back together so that the application can process the evidence from the whole file [10]. The idea of dividing a file into chunks could be extremely challenging as one could argue that separating a content of a file into chunks and combining them again could violate the integrity of the original file (the evidence).

The above scenario becomes more complicated if data is being captured in real-time, for example as a network packet capture. This type of appli-

[2]dd is a common Unix program whose primary purpose is the low-level copying and conversion of raw data

cation is similar in principle of the 'dd' capture process but the difference is that the amount of data to be captured is unknown when the process is commenced.

1.5 Issues and Proposed Solutions

1.5.1 Strategies for Detection

The traditional intrusion detection system relies on a single IDS to monitor a whole network. If this IDS misses an attack, which could happen when receiving multiple attacks, then that attack can get through the network. Another shortcoming when deploying a single IDS is the possibility of having it crash due to the huge traffic of attacks. In this thesis, we evaluate the idea of having multiple IDSs with multiple servers and look into other multiple entities in the network. This enables us to have more protection advantages. For example, when one IDS reports that a bad patterns of attacks is under way we can therefore react to protect other servers in the network. This is further discussed in later chapters. (See Chapter 5 and 6)

On the other hand, many IDSs have been implemented only for security purposes and do not consider the need for forensically sound data storage. In other words, IDSs monitor attacks in order to prevent them rather than recording the performed steps for auditing purposes. For this reason, evidence may be not adequate when needed in the future. It will also create tremendous work for trainees who teach new users to relevant fields.

1.5.2 Forensic Solution

This thesis associates the concept of a DEB with the intrusion detection systems. It is conducted that a Proof of Concept (PoC) of having multiple IDSs with a ceneral database. This database is connected with a proposed DEB.

This DEB enables users to acquire evidence from IDSs and possibly other tools at the same time. The main idea is to have a DEB through which we can access other tools and obtain evidence through each and every step is being recorded. We can eventually complete the job producing a wrapped DEB being hashed with data (evidence), index and a tag. This includes all performed functions from the beginning to the last action taken. (See Section 4.3).

Chapter 2

Contemperary Approach IDS/SIEM Architecture

2.1 The Problem

2.1.1 Malware

Malware is hidden software designed to access a computer in order to manipulate it. It is also used to perform actions on a client's computer without the knowledge of that client. The particular actions of this software are designed by its creator rather than any particular features it may have. Malware includes computer viruses, worms, trojan horses, spyware, dishonestadware, scareware, crimeware, most rootkits, and other malicious and unwanted software or program. Malware can cause loss of data, personal information gathering and spam spreading. Few users intend to infect their own machines and in most cases users' machines are affected by others (attackers). Malware propagation varies from spreading by others to self-spreading into the network. These propagation methods include email attachments, worms (automatically propagating through the network), Trojan horses (a destructive software that is hidden or shown as a benign application) or drive-by-downloads which mean mean one of the following:

- Downloads which a person authorizes but without understanding the

consequences (e.g. downloads which install an unknown or counterfeit executable program, ActiveX component, or Java applet).

- Any download that occurs without a person's knowledge or permission.
- Download of spyware, a computer virus or any kind of malware that happens without a person's knowledge.

Modern Malware Propagation Techniques

Identifying malware propagation has become much more difficult than it used to be. IDS signatures cannot always detect new malware releases or polymorphic malware. At the early turn of the millennium, an entirely new breed of propagation techniques were released into the world with techniques spawned from the lessons learned from prior malware outbreaks.

Malware trends have evolved to such a point that we now depend on experts to predict potential new outbreaks or methods while old techniques may lead to innovations that decrease the damage caused by its predecessors. New techniques are built upon using system enhancements and feature upgrades of operating systems and applications against end users. The worms described in Table 2.1 use newer methods of infection and propagation and have been the source of significant outbreaks in recent IT history. As shown in Table 2.1, those worms use more recent changes in malware propagation methods. These changes have impacted the recent IT history by causing significant outbreaks.

The worm Conficker (also known as Downup, Downadup and Kido) [1] in itself has infected over 9 million computers within a week or even less of its introduction. It is important to evaluate the development of malware from custom targeted malware against organizations all the way down to

[1]Conficker: is a computer worm targeting the Microsoft Windows operating system that was first detected in November 2008. It uses flaws in Windows software and Dictionary attacks on administrator passwords to co-opt machines and link them into a virtual computer that can be controlled remotely by its authors.

simple client-side exploits that execute malicious code in order to remotely take control of victim computers. Although most of the examples are Microsoft Windows focused malware and are reported in the press, quantifying the entirety of all worldwide malware is still the key. All of the techniques used during malware's initial evolutionary period can be seen conceptually in today's malware releases. damage recount, was mainly due to result of the advances development of network and routing services which were designed to ease the network administrator's daily roles and responsibilities.

Early in the twenty first century, malware authors also started using techniques that have become increasingly difficult for forensics analysts and network defenders to identify and mitigate. Historically, their methods have ranged from very traditional straightforward ones to highly innovative approaches, which cause significant difficulties for administrators around the world. In fact, according to Click Forensics, about 16-17% of all advertising link clicks in 2008 were fake, of which a third was generated by botnets. A simple calculation will show that botnet owners made $33 million "for clicks".

2.1.2 Hacking

Hacking is a security vulnerability on a computer system that is being abused by an attacker. A security exploit is a prepared application that takes advantage of a known weakness. Common examples of security vulnerabilities are SQL Injection, Cross Site Scripting which normally result from weaknesses on the programming practices. Other exploits would be able to be used through FTP, HTTP, PHP, SSH, Telnet and some web-pages. These are very common in website/domain hacking.

In recent years hackers collaborate on groups and threaten not only individuals but also businesses and governments throughout the world. A good example for a hacking group is Anonymous (Group)[2] who are a group of

[2]Anonymous (used as a mass noun) is a group initiating active civil disobedience and

Malware	Year	Injection Technique	Propagation Techniques
StormWorm	2007	Email attachments/ File execution	File dropper Overwrite/deletion. P2P C2 structure and Fast Flux communication chaining.
AutoIT	2008	File execution	Copies generated onto removable drives by overwriting the autorun.inf
Conficer	2009	File execution	File transfer, file sharing, copying itself across network shares or shares with weak passwords.
Bacteraloh	2009	File execution	Disguised as a crack utility that (P2P network-based) a user downloads and executes locally.
Koobface	2009	Client-side exploit	Spread through social networking sites with a loaded URL linked to the malware through sites such as Facebook, MySpace, Friendster, and LiveJournal.

Table 2.1: Latest Changes of Malware Propagation Techniques

hackers from all over the world who individually attack one target at the same time. They announce a particular date for carrying out the actual operation. They also make it easy for other script kiddies by distributing tools throughout the internet together with demonstrations on how to use them.

The consequences of these attacks are severe, ranging from business suspensions, that cause the loss of millions of dollars for commercial companies (e.g. Sony), to the publication of sensitive original documents from such organisations as Vanguard Defense Industries, a Pentagon and FBI contractor and Police (e.g. Italian cyber-police). One current potential attack is Facebook Operation (OPfacebook). The aim of this attack is to destroy Facebook. Anonymous hackers are claiming that Facebook sells users' private information to third parties. So far no solutions have been found to prevent Anonymous from launching their attacks (e.g. DDOS). This is mainly due to

spread through the Internet while staying hidden, originating in 2003 on the image board 4chan, representing the concept of many online community users simultaneously existing as an anarchic, digitized global brain. It is also generally considered to be a blanket term for members of certain Internet subcultures, a way to refer to the actions of people in an environment where their actual identities are not known.

the variety of attackers' demographics. Simply put, it is impossible to stop random people from all over the world visiting your website. We carry out a similar simualted attack on the testbed chapter. (See Section 5.3.3, Chapter 5.3.3).

2.1.3 Botnets

Botnets have become one of the greatest security threats to the web. The term Botnet can mean either a Bot or a BotMaster. The term Bot is an abbreviation of robot which is also called Zombie [3]

It is malware that compromises a host which can then be remotely controlled and manipulated by a BotMaster (the owner or the software source). The Botmaster runs commands into the bot through a remote command and control (C&C) infrastructure. Once the robot "Bot" software has been successfully installed in a host, this host becomes a zombie or a drone, unable to block the commands of the BotMaster. While the aim of all typical malware including viruses and worms is to attack the compromised host, Bots can receive commands from the BotMaster and can be used in a distributed attack platform. This type of BotMaster is also known as a BotHerder and is a user or a group of users who control the remote Bots. The number of these remote Bots can be either small or very large depending on the complexity and sophistication of the Bots used. Botnets are created by the BotMaster to setup a private communication infrastructure which can be used for malicious activities such as a Distributed Denial-of-Service [12],[13],[14]

Attacks commonly come through an establishment of an IRC from which the Bot's channels connect to or listen to the commands from a BotMas-

[3]A zombie computer is often described as "a zombie". It is a computer connected to the Internet that has been compromised by a cracker, computer virus or trojan horse and can be used to perform malicious tasks of one sort or another under remote direction. Botnets of zombie computers are often used to spread email spam and launch Denial of Service attacks. Most owners of zombie computers are unaware that their system is being used in this way. Because the owner tends to be unaware, these computers are metaphorically compared to zombies.

ter. HTTP-based Botnets are similar to the IRC-based ones. After infecting the host the Bots contact a web-based C&C server and notify the server with their system-identifying information via HTTP. This server sends back commands via HTTP responses. Although IRC and HTTP based C&C have been adopted by many past and current Botnets, both of them are vulnerable to a central-point-of-failure. This is when the central point is disabled which then disables all the relevant Botnets. However, attackers have removed this possibility by using a new type of Botnet utilizing decentralized C&C protocols such as P2P. This type of decentralised C&C infrastructure creates an enormous problem and makes it very difficult to detect or even mitigate the problem. Storm and Waledac are both good examples of a hybrid P2P Botnets which can spread through emails. [12],[13].

The current Botnet detection methods include honeypot and honeynet for capturing and analysing data for malicious behaviours, and detecting approaches for different C&C mechanisms such as IRC, HTTP, DNS, or P2P. However, these methods mainly look into the network traffic and acquire evidence of Botnet activities indirectly. For instance, the evidence for detecting the upgrade of Bot is acquired by identifying the upgrade binaries in the traffic, rather than directly obtaining them from the code server which logs the download event[15].

2.2 The IDS Solution

Two main approaches have been devised to analyse events to detect attacks: these are detection of signatures and detection of anomalies. The signature detection is the technique used by most commercial systems. This is where the signatures of attacks are known and thus detected. Whereas anomaly detection analyses unusual patterns of activity that are being investigated. The detection of anomalies is used by a small number of IDSs[16],[17].

Signature-based detectors look for events that could match predefined

patterns or signatures of a particular attack. The analysis is based on a comparison of patterns (pattern matching). The system has a database of attack patterns that is used to match similar patterns and when that happens an alarm will be triggered. The only issue with these types of detectors is the possibility of generating a large number of false positives which can waste a lot of time and money[16],[17],[18].

Anomaly detection checks if there is any kind of deviations from normal patterns. This can be divided into static and dynamic detection. The former detection is based on monitoring a portion of the system that must be fixed[18]. For example, operating systems software and data that bootstrap a computer never change. If any part of these systems changes, an alarm will be triggered. The latter operates on audit records or on monitored networked traffic data. This can be detected through the number of files accessed by a user in a given period of time or the number of unsuccessful attempts to enter the system[16],[17],[19].
A basic IDS Architecture is shown in Figure2.1.

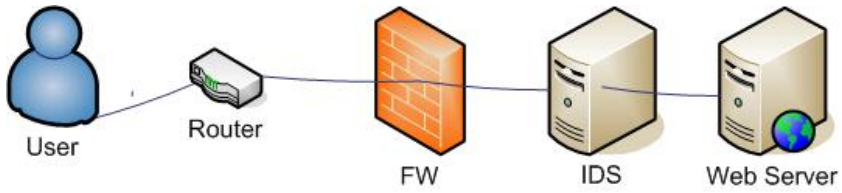


Figure 2.1: Basic IDS ArchitectureWhere the IDS Can Be Allocated According to the Network Peripheries

2.2.1 NIDS vs HIDS

Intrusion Detection is the process of identifying and responding to malicious activity targeted at computing and networking resources. In general IDSs are categorized into the two types network intrusion detection systems (NIDS) and host-based IDS. A NIDS monitors packets on the network wire and

attempts to discover if a hacker is attempting to break into a system. A typical example is a system that watches for large number of TCP connection requests (SYN) to many different ports on a target machine thus, discovering if someone is attempting a TCP port scan [20].

2.2.2 Honeypot and Honeynet

Honeypots are systems used to attract hackers' attention by enabling known vulnerabilities. Hackers are likely to spend time around these exposed vulnerabilities. These honeypots enable the forensic analyst to view the hackers' actions and log their activities and techniques. Once these techniques have been logged, actual servers can be used to trace the hackers' actions and so the security of the servers can be enhanced.

Honeypots can be built with different pathways and placed within many places. The honeypot should have common services running on it in order to make it appear to be a real server. These common services include Telnet server (port 23), Hyper Text Transfer Protocol (HTTP) server (port 80), File Transfer Protocol (FTP) server (port 21) and so on. It should be placed close to the production server so that the hackers readily assume that it is a real server. For example, if production servers have Internet Protocol (IP) addresses 192.168.10.11 and 192.168.10.13, you can assign an IP address of 192.168.10.12 to the honeypot.

Two or more honeypots within a network form a honey net which can then form a network of high communication honeypots. This is used when monitoring a large network which requires more than a single honeypot. Honeynets and honeypots are usually implemented as parts of larger network intrusion detection systems. A honeyfarm is a centralized collection of honeypots and analysis tools.

2.2.3 Client Honeypot

As with traditional server honeypots, there are two types of client honeypots: low and high interaction client honeypots. A honeypot acts in a particular way as a client and receives attacks. Client honeypots are classified in two main types: a low interaction, and a high interaction. The former uses a simulated client such as "wget"[4] that interacts with servers. While this type performs more quickly than the latter one it can produce false alerts or miss a malicious server, especially since it does not act as a "real" client and has programmatic limitations. This type may also fail to fully emulates all vulnerabilities in a client application. A high interaction client honeypot uses a dedicated operating system that has an actual vulnerable client interacting with malicious servers. It includes actual software that a client could be expected to run. Since no signatures are used, a high interaction client honeypot is able to detect unknown attacks.

We only briefly discuss honeypots as they are a small part of our project but we will carry out an experiment on a high interaction honeypot client (VMware) in order to incorporate their form of alerts into our proposed DEB. For more details about honeypots refer to [REF22] [REF120].

[4]GNU Wget (or just Wget, formerly Geturl) is a computer program that retrieves content from web servers, and is part of the GNU Project. Its name is derived from World Wide Web and get. It supports downloading via HTTP, HTTPS, and FTPprotocols, the most popular TCP/IP-based protocols used for web browsing.

Chapter 3

Requirements of Forensic Evidence

This chapter provides guidance on the management of electronic records that may be used as evidence in judicial or administrative proceedings. Such management applies whether the evidence is to be used by a plaintiff, defendant or for referral to appropriate authorities for investigation.

While this chapter provides guidance to the management of electronic records relating to litigation in New Zealand and Australia, the processes and procedures of that management of electronic records are consistent with global industry best-practice and will increase the value of digital evidence in many other jurisdictions.

It also discusses the main forensic structure that is required in the designing of the DEB architecture. The idea behind this chapter is to have a universal DEB structure that can be used by any digital investigator around the glob. A Proof of Concept (PoC) of a novel approach to a DEB design is further discussed and demonstrated in the tests and results chapter. (See Chapter 6, Section 6.4).

3.1 Overview

IT evidence is a broad term used to describe any records generated by, or stored on a computer system that may be used as evidence in court proceedings. IT evidence also encompasses computer-generated or stored records that detail management decisions which may be subjected to regulatory or judicial scrutiny. IT Evidence can be divided into three categories:

- records that are generated by computers;
- records that are merely computer-stored; and
- both generated records and stored records. The distinction hinges upon whether a human or a computer created the records' actual contents.

Records that are generated by computers refer to documents which contain the writings of users and which happen to be in electronic form. E-mail messages, word processing files and internet chat room messages are good examples. The main evidentiary issue is demonstrating that it is a reliable and trustworthy record of what was stated.

In contrast, computer-generated records contain the output of computer programs, untouched by human hands. Common examples are log files, telephone records, ATM transaction receipts. The key evidentiary issue here is demonstrating that the computer program generating the record is functioning properly. A third category of IT evidence is a combination of the previous two records that are both computer-stored and computer-generated. A common example is a financial spreadsheet that contains both human statements (input to the spreadsheet program) and computer processing (mathematical calculation performed by the spreadsheet program).

3.2 Principles for the Management of IT Evidence

The principles for the management of IT evidence only give assistance, not authority. Although individual jurisdictions will have specific evidentiary requirements, practitioners must ensure that the electronic records produced, collected, analysed are presented in accordance with these principles in order for them to be admitted and therefore accepted by courts. The following defines guiding principles for the management of IT evidence. These relate to:

3.2.1 The obligation to provide Evidence

Investigators have to keep updated with regulatory, administrative and best-practice in order to provide forensically sound evidence. It is also important to understand the steps by which the actual weight of the evidence can be maximised.

3.2.2 Design for Evidence

The following must be considered when using any tool to create the evidence necessary for a legal case of evidence:

- The capability of altering electronic evidence;
- The capability of authenticating electronic evidence;
- The reliability of tools generating such evidence;
- The time stamps and the date of creating, accessing and altering evidence;
- The chain of custody of who is taking care of the evidence; what do you mean by this and;
- The safe custody and handling of the evidence.

This also applies to the design or acquisition of new ICT systems or the upgrade of existing systems.

3.2.3 Evidence Collection

Collecting evidence must be stored in a forensically sound manner. Two elements must be considered when collecting evidence:

- The evidence must be technologically robust
- The evidence must be legally robust

3.2.4 Chain of custody

There must be a method of recording all access to and handling of evidence.

3.2.5 The original, copy and original copy

It is always crucial to have another copy of the original one in case any the computer and/or the information and evidence contained therein is damaged. It is also important to make sure that any performed actions on the original or a copy are appropriate and are appropriately recorded and documented.

3.2.6 Personnel

From a management perspective it is essential to ensure that personnel who carry the design, production, collection, analysis and presentation of evidence have appropriate training, experience and qualifications to confidently perform their roles.

3.3 IT Evidence Management Lifecycle

This section introduces the IT evidence management lifecycle and explains how the principles can be applied to each of the six lifecycle stages. While actual evidence is unknowingly generated when a criminal leaves his/her DNA

or fingerprints, digital evidence is generated by computer systems which have the capability of increasing their evidential value. In addition, the computer generated digital evidence has to be carefully processed and handled in order to increase its evidentiary weight. The IT evidence management life cycle is illustrated in Figure 3.1

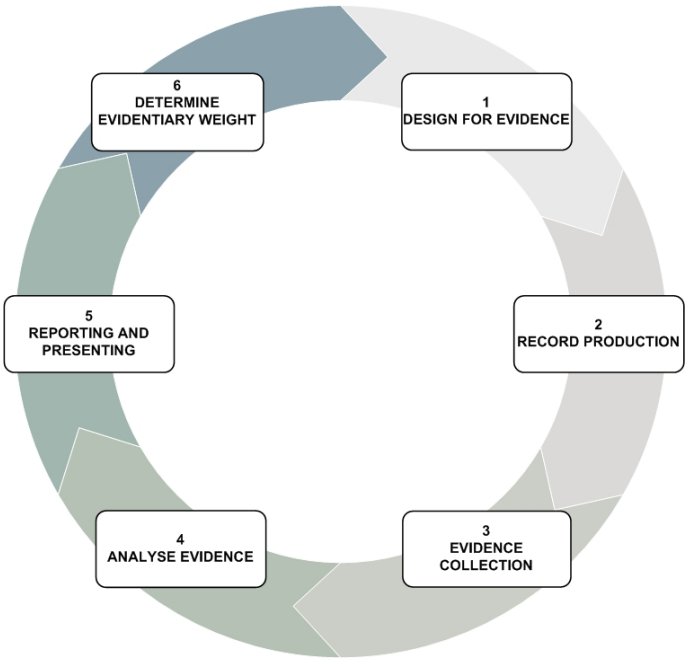


Figure 3.1: IT Evidence Management Lifecycle

3.3.1 Stage1: Design for Evidence

There are four objectives when designing a computer system to increase the evidentiary weight of electronic evidence

1. Electronic records must be able to be identified, available and usable;
2. The author of electronic records must be able to be identified;
3. The authenticity of the electronic records; and

4. The time stamps and the dates of creating, accessing and altering electronic records;

5. The reliability of computer programs must be able to be established.

Another important objective is the design of the procedures that are to be conducted by personnel for collecting, analysing and reporting digital evidence. Such procedures are discussed in the relevant stage of the lifecycle and should be;

1. Designed prior to them being necessary; tested to ensure that personnel are able to carry them out; and

2. The design of each procedure must be clear (unambiguous) and decrease the amount of decision-making.

The author of electronic records is identified

Identifying human author The author of a computer-stored record should be able to be identified electronically. Prior to recording the author's electronic identity, a user authentication system should be used. Authentication is any process by which users verify that someone is who they claim they are. This usually involves a user name and a password, but can include any other method of demonstrating identity, such as a smart card, retina scan, voice recognition or fingerprints.

The evidential weight of the recording of the author's identity will depend on the strength of the user authentication system. ISO 9798:1977— Entity Authentication, for example, specifies techniques used by authentication systems for corroborating user or computer identification.

Identifying the computer author Each computer program generating elements of the electronic record must be clearly identified in the record. This may be achieved by, for example:

- Clearly identified, unique and consistent labelling of file names; or

- Clearly identified, unique and consistent labelling within the record.

Human and computer authors Because electronic records may consist of both computer-stored and computer generated elements, both must be identifiable. For instance, a financial spreadsheet includes typical human numerical entries and the calculation formula. It also includes computer-created records derived by the spreadsheet program from the computer-stored records. Therefore, both the human author and the system author must be able to be identified.

Establishing the authenticity of electronic records

Two elements must be achieved in order to establish the authenticity of electronic records

1. The original electronic record must be able to be identified; and
2. Each alteration must be identified as to whether it was a human or computer author and recorded.

The time stamps of electronic records

As electronic records are being generated or altered it is important that management ensures that time and date stamps exist in their computer system and are maintained by the organisation. In order to achieve this, a timestamp must be activated at the time of creation of each record. As the electronic record is being altered the timestamp must be updated. RFC 3339—Date And Time On The Internet: this provides the timestamps which specifies a format for timestamps that may be used. Also see for example ISO/IEC 18014 - Time-stamping Services.

All computer system clocks must be synchronised to a central reference to ensure the right time is being conducted. Amongst others the Universal

Time Coordinated (UTC16)[1] provides a central reference for computer system clocks.

Establishing the reliability of computer systems

In order to ensure a precise recording of the author's statement it is important to establish the reliability of the computer systems that generate the electronic recordsand that those systems operate correctly and precisely. Their reliability must be demonstrated by the following:

1. the computer program was built correctly i.e. the output is: i) consistent with its design; ii) predictable; and iii) repeatable.

2. there were no fault or errors in the program when the electronic record was created, copied or altered. In other words the program was operating correctly during the capturing process.

Formal design criteria when designing the formal criteria for a software program the methodologies of, for example, ISO 15504-Software Process Assessment or by accreditation to the appropriate level of the Capability Maturity Model17 (CMM) should be adhered to [21]. When buying a software program the formal assessment criteria of the manufacture can be used to clarify the reliability the new software.

Source code In order to ensure the reliability of a software program, its source code has to be analysed by experts. When acquiring an open source software program or producing it, the source code should be retained. In order to enable a software demonstration from its source code. However, when buying a software program the buyer must ensure that a guarantee of its source code (same version) is available at any time [21].

[1] Also known as Universal Time Clock or Zulu time

3.3.2 Stage 2: Production of Records

At this stage of the life cycle critical operations are performed. The main objective of this stage is to be able to initiate the following:

- a particular software generated an electronic record ;
- for computer-stored records, the human author ;
- the timestamps of creation; and
- being able to make sure that the software is operating correctly when the electronic records are being created or altered.

When maintaining electronic records of evidential significance best-practice controls should be applied to all computer system operations. For instance, those indicated in ISO 17799 - Code of Practice for Information Security Management part 8 - Communications and Operations Management

In order to show that a particular software program was performing correctly when the electronic records were being captured, the following requirements have to be met:

- that the computer program was operating; and
- that the computer program is valid as to its reliability.

Circumstantial evidence may also be used to demonstrate that a computer program is operating correctly. For example, a statement by a person asserting that he/she was using a particular computer program at a particular time and that he/she observed certain things, could be strong evidence of the operation of a computer program that produces computer-stored records.

In addition when designing or using a computer software program which generates electronic records, a record of operational faults must be maintained. For further details on this matter see ISO 17799-Code of Practice for Information Security Management part 8.4 Housekeeping [21].

Stage 3: Evidence Collection

Relevant information (evidence) has to be obtained when securing the original copies of those obtained of this information.As stated above under the section Principles for the Management of IT Evidence the process of acquiring evidence must not be performed on the original.

Standards for evidence collection The standard of the evidence collected is one factor determining the evidential weight of electronic records.

Forensically Sound In order to ensure that the evidence presented is admissible, forensically sound procedures must be followed. These procedures must show the original and every action, whether human or computer generated, thereafter in order to be admitted as evidence.

Best Evidence The judiciary will decide whether the evidence is admissible. For example, in Australia the judiciary has significant discretion to recognise records as evidence even when the forensic specialists themselves have not collected the electronic records admitted as evidence. When forensic specialists get involved, the value of evidence does increase however, it is not wise to rely on this judicial discretion. By following the correct procedures at each and every stage the weight of the evidence will increase and speak for itself [21].

Contemporary notes Contemporary notes written at the time the evidence was obtained can be relevant even some years later when the investigators or personnel who made those notes are called to appear before the Court to give evidence. This may happen years after the evidence collection process was performed. For this reason contemporaneous notes are very important and must record any actions that were performed whether on the original or other copies. These contemporaneous notes may include the process of decision-making such as why those decisions were taken, persons consulted and authorities sought. It is necessary that contemporaneous notes include

facts such as actions performed and observations made. These observations must not be opinions. It is also far more important to ensure that those notes do not interfere with the evidence presented [21].

Chain of custody Any personnel who have gained access to a particular electronic record at any given time must be identified. This is from the creation of the electronic records, to the presentation of the evidence. The electronic records evidential weight will be substantially reduced if the chain of custody cannot be adequately proved or is discredited. This is to avoid any potential allegations of data tampering or misconduct which can compromise the Court case.

Evidence copy When relevant information is produced as evidence, a copy of the evidence will be provided to the Court and the other party so the chain of custody can be demonstrated. An individual can be responsible for the chain of custody and so monitor all access to it. The copy of evidence may be created by:

1. regenerating the electronic record of evidence as a hard copy (a printed document)

2. copying the evidence to another "offline" media (e.g. floppy disk, CD-rom, backup tape, external hard drive); or

3. utilising system access privileges to control access. When an electronic record of c) evidence is copied, it is a must provide proof that the copy has not been tampered. It is recommended that a number of evidence-copies should be created and a chain of custody be established for each copy.

Custody log The individual in charge of the evidence copy must maintain a log recording of:

1. Users who access the evidence;

2. The time stamps, date and purpose for each user's access; and

3. When any copy of the evidence is removed, the time and date of removal and return must be logged.

4. All activity related to the digital evidence should be documented according to the planned procedures for the custody of evidence.

Non-readable electronic records There is data which may be stored within non-readable files (or even readable) that is evidentially useful but which can be easily altered or deleted by certain computer software. For example, the slack space of a disk drive may include deleted files or encrypted files that may contain key electronic records. This can be an issue when reaching the lifecycle stage of analysing electronic records.

Since non-readable files can be easily altered or deleted by computer software, investigators need to pay more attention when locating those non-readable files so as not to tamper with their contents when collecting evidence.

Limitations When collecting evidence, investigators must follow rules that control the access of or declaration of particular information. If any of these rules are violated, the credibility of evidence will be comprised. This could decrease the weight of the evidence and in the worst case scenario even prevent the evidence from being admitted in Court. Personnel who do not follow the rules may expose themselves to penalties. For example: The Telecommunications (Interception) Act (1979) provides for criminal penalties for the unauthorised interception of a "communication". Evidence collectors must be able to determine if an electronic message (e.g. e-mail or IRC19) constitutes a communication or if it is merely data [21].

Stage 4: Analyse evidence

The objectives of this stage of the lifecycle are to:

1. assemble from IT evidentiary records material facts;

2. deduce from IT evidentiary records opinions relating to those facts; and

3. determine what other IT evidence is lacking that will assist the enquiry.

Use evidence copy In order to analyse an electronic record, an evidence copy of the original must be used while the original remain in a safe condition (untouched). Only original electronic record is used to certify

- if copies are duplicates of the original; or
- if the original has been altered.

Personnel qualifications The analysis process of the IT evidence should be conducted by professional people who are appropriately qualified for the function they are performing. It is important to decide whether an ordinary or expert witness is required. While ordinary witnesses' analysis is on matters of fact only, expert witnesses may provide matters of opinion from the IT evidence [21].

Completeness of evidence IT evidence is circumstantial. Specialists conducting analysis of IT evidence must be provided with details of:

- why the evidence is required
- the circumstances in which the electronic records were created; and
- the computer systems creating the electronic records.

Material electronic records may be neglected or their significance diminished without a thorough understanding of the background.

Stage 5: Reporting and presentation

In this stage of the life cycle, the aim is for investigators to convince decision-makers (management, lawyer, Judge,standards for evidence collection forensic...etc) of the validity of the facts and opinions retrieved from the evidence.

For most IT evidence, the original electronic record consists of electronic impulses stored on media. It must be converted into human readable format prior to presentation, either by computer print out or by using a computer program.

If IT evidence is to be used in legal proceedings, the investigator will be advised of the suitable manner and form in which the evidence should be reported and presented.

Stage 6: Determine evidentiary weight

The objective of this stage is to assess the evidentiary weighting of the electronic records and the reports. Assessment of the evidentiary weighting of electronic records occurs during all stages of the lifecycle. In earlier stages of the lifecycle (i.e. one through five) assessment is performed by the participants or stakeholders such as lawyers. A final assessment is performed by an independent arbitrator who may be a Magistrate or Judge; a member of a tribunal or an arbitrator; or senior organizational management [21].

Two criteria are used to measure the evidential weight of electronic records

1. Probative value: Has the electronic record relevancy, authorship, authenticity, correct operation and reliability been established?; and
2. Rules of evidence: Has the electronic record been collected and handled correctly in accordance with the rules?

Each of these criteria encompasses many factors.

Probative value Records must relevant and all relevant electronic records must be presented and more importantly, records must be relevant to the matter at hand. Organizations must demonstrate that the procedures used to collect electronic records were reasonable and robust enough to discover obvious, lost or hidden material. The following must be satisfactorily established

1. Authorship ;

2. Authenticity ; and

3. Correct operation and reliability of the computer program.

Rules of evidence With some exceptions, the aim of the rules of evidence is to exclude evidence that is either irrelevant or unreliable. If organizations collect and handle IT evidence in accordance with this handbook, they will minimise the risk of having such evidence excluded by operation of any applicable rules of evidence.

Chapter 4

Design of Analysis Engine and DEB Structure

4.1 Design of Intrusion Detection System Correlation Engine

4.2 Fuzzing

Web application fuzzing is a specialized form of network protocol fuzzing. A network protocol fuzzer is a fuzzer that communicates with its target through a particular form of socket. While the network protocol fuzzer can mutate any type of network data, a web application fuzzer only mutates in HTTP packets. Web application technologies include: CGI (Common Gateway Interface), PHP (Hypertext Preprocessor), Flash, JavaScript, Java and ASP.NET [22].

Web application fuzzing can indicate not only the web application vulnerabilities but also other associated components' vulnerabilities such as the database server or web server [22]. Such vulnerabilities that can be identified include SQL injection, Cross-site scripting (XSS), Directory Traversal and Buffer Overflow.

We useed fuzzing techniques in this thesis to save time finding potential vulnerabilities. We also wanted to try all possible ways in order to trigger alerts of the designed intrusion detection system coloration engine.The aim was to make our plan more efficient and save time in terms of carrying attacks.

4.2.1 Relevant Attacks

The input validation attacks category is a serious issue for web application security. Examples of such attacks are SQL injection and Cross-Site Scripting (XSS). In this thesis, we look at some Input Validation attacks including SQL injection and XSS.

Although the majority of web vulnerabilities are easy to understand and to avoid, many web developers are, unfortunately, not security-aware. Before commencing these attacks, a test has to be made to know whether these web applications are vulnerable to such attacks. These tests can be implemented with certain methodologies (See Section 4.2.2).

SQL Injection

SQL injection is a vulnerability that enables malicious codes to be inserted into strings that are later passed to an SQL Server for parsing and execution. SQL injection attacks can occur if a web application does not properly filter user input. There are many varieties of SQL Injection attacks. The typical unit of execution in the SQL language is the use of query the aim of which is to retrieve data from the database server. These queries can also be used to manipulate records in the database server. This could lead to obtaining, modifying or even deleting data from the database server. Figure 4.1 shows a simple digram for an SQL Injection attack.

Normal SQL Injection Attacks: A web application is vulnerable to an SQL Injection attack if an attacker is able to insert SQL statements into the web application. This is usually achieved by injecting malicious in-

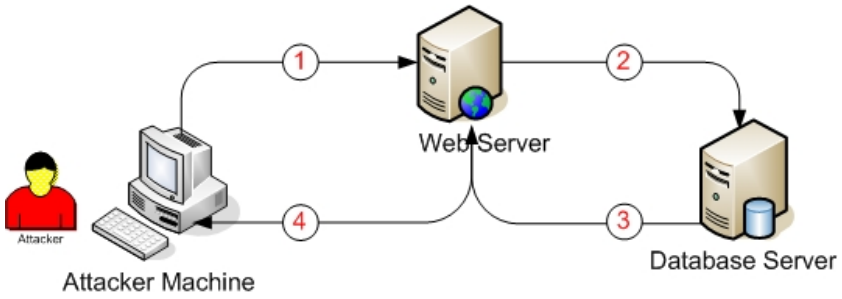


Figure 4.1: SQL Injection Process

puts into user fields or website addresses that are used to compose the query. For example, The following C# code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user [23],[24]. See Figure 4.2.

```
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = "'"
                                + userName + "' AND itemname = '"
                                + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
```

Figure 4.2: A C# code executes a SQL query to find a specified name

```
SELECT * FROM items
WHERE owner =
AND itemname = ;
```

Figure 4.3: Shows the query that Figure 4.2 code intends to execute

However, because the query is constructed dynamically by concatenating

a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name Smith enters the string "name' OR 'a'='a" for item Name, then the query becomes as Figure 4.4

```
SELECT * FROM items
WHERE owner = 'Smith'
AND itemname = 'name' OR 'a'='a';
```

Figure 4.4: An attacker enters the string "name' OR 'a'='a"' as bad inputs

The addition of the OR 'a'='a' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query as shown in Figure 4.5.

```
SELECT * FROM items;
```

Figure 4.5: An attacker enters the string "name' OR 'a'='a"' as bad inputs

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Blind SQL Injection Attacks: Blind SQL injection occurs when an attacker is able to execute SQL injection attacks and thus receive an error response from the server stating this was a SQL Query's syntax error. A blind SQL injection attack is similar to a normal SQL injection attack; however, instead of receiving a useful error message in the latter situation, a generic page specified by the developer appears. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker is

still able to steal data by asking a series of True and False questions through SQL statements. This is carried out through simple statements to combined and very sophisticated ones in order to determine the server's response and so build a structure to achieve the attacker's intended goal which might be, for instance, stealing the administrator's user name and password [25],[24].

An attacker may verify whether a sent request returned True or False in the following ways:

(in)visible content Having a simple page, which displays an article with a given ID as the parameter, the attacker may perform a couple of simple tests if a page is vulnerable to SQL Injection attack. For example, the following website link: http://example.com/items.php?id=2 (See Figure 4.6)

```
http://example.com/items.php?id=2

This link sends the following query to the database:

SELECT title, description, body FROM items WHERE ID = 2
```

Figure 4.6: An attacker visits http://example.com/items.php?id=2

The attacker may try to inject any query, even an invalid one, which should cause the query to return no results. (See Figure 4.7)

```
http://example.com/items.php?id=2 and 1=2

This modification will change the SQL query so it will look like this:

SELECT title, description, body FROM items WHERE ID = 2 and 1=2
```

Figure 4.7: An attacker visits http://example.com/items.php?id=2

The query is not therefore going to return anything. Even if the web application is vulnerable to SQL Injection, it probably will still not return

anything. To make sure, the attacker will certainly then inject a valid query. (See Figure 4.8)

If the content of the page is the same, then the attacker is able to differen-

```
http://example.com/items.php?id=2 and 1=1
```

Figure 4.8: An attacker injects a valid SQL query into a website link

tiate whether the query is True or False. The only limitations are privileges set up by the database administrator, different SQL dialects and finally the attacker's imagination [26],[27],[28],[29],[30].

The scenario of figures 4.9, 4.10, 4.11, 4.12 and 4.13 show the process of potential Blind SQL Injection attack on a back-end database server. The first step is when the attacker checks whether the back end server was vulnerable to SQL Injection attack. If the back-end server was vulnerable, the attacker then checks the infected columns from which further information in the database can be retrieved. Then the attacker reveals the databases' names that are stored in the back end server. This step is followed with revealing the actual database in use by the web application at present. As the attacker send requests and receives responses through the browser (HTTP protocol), he/she records valuable information that can be used in the following steps to complete the attack. Eventually, the attacker reaches a stage where he/she can retrieves the tables' names and looks for a table which may include sensitive information such as administrator's user name and password. If this attack was completed the attacker would be in control of all the website contents.

This attack was performed for testing and demonstrating the actual attack without completing it. The targeted website was informed about their SQL Injection vulnerability. A similar attack is performed later on this thesis. The aim of performing such attack is to find a way of recording the attacker behaviour and record it before the attack actually strikes on the

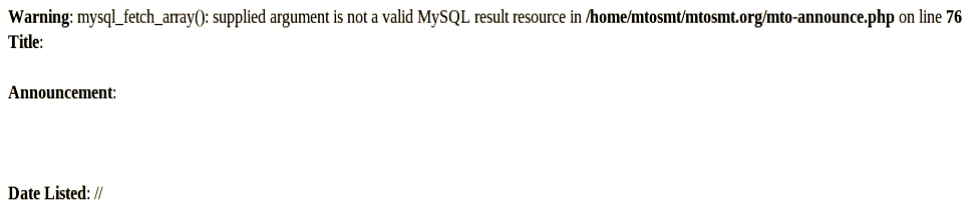


Figure 4.9: Step 1: The Attacker Sent SQL Injection Inputs and Received the Response Above, Showing an Error Message Which Indicates an SQL Injection Vulnerability

Figure 4.10: Step 2: The Attacker Proceeded Further Exploiting the SQL Injection, Retrieving the Number of the Infected Columns in the Database

Figure 4.11: Step 3:The Attacker Proceeded Even Further Retrieving the Databases' Names

database server. This can be used to prevent similar attacks from actually happening and also more important to provide valid digital forensic evidence against the attacker if needed in the future. (See Chapter 5)

Figure 4.12: Step 4: The Attacker has Revealed the Database Name that is in Use by the Web Application

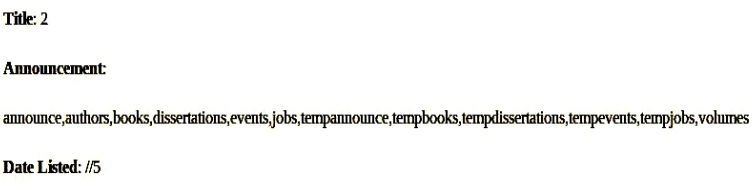


Figure 4.13: Step 5: The Attacker Utilised the Already Retrieved Information on Revealing the Available Tables on the Database

Cross-site scripting (XSS)

Cross Site Scripting (XSS, sometimes also abbreviated as CSS) refers to a range of attacks in which the attacker injects malicious client-side script into a web application or web pages viewed by other users [2, 9]. XSS can occur when an attacker intentionally injects a malicious script into a web page. This malicious script in most cases is sent through a web request. XSS can also occur when data in a dynamic content is sent to a web user without that data being checked as to whether it contains a malicious code [31],[25],[32], [33].

That malicious code is often in a JavaScript or it can include HTML, Flash or any other form of code that can be executed in the web server. The code can result in sensitive data including cookies, session information, or other details retained by the browser being revealed. The attacker could also redirect the victim to a controlled web content or perform other suspicious

operations on the targeted computer[25].

XSS scripts can also rewrite the content of the HTML page which can result in significant damage. These attacks are easy to execute but difficult to prevent. XSS attacks can generally be categorized into two categories: Stored and Reflected. There is a third category, a much less well known type of XSS attack called DOM Based XSS. The third one will not be discussed in this thesis.

Reflected XSS attack (also known as first-order XSS or Type XSS); and Stored XXS attack (also known as second-order XSS, persistent, or Type 2 XSS) [34],[30]. These two are further explained below.

Reflected XSS vulnerabilities result from the application inserting part of the user's input in the next HTML page that it renders. This happens when an attacker encourages a user to click on a (disguised) URL which contains malicious HTML/JavaScript code. The user's browser then displays HTML and executes JavaScript that was part of the attacker-crafted malicious URL. This can result in the theft of browser cookies and other sensitive user data.

Stored XSS Attack vulnerabilities result from the application permanently storing the attacker's input within the target servers, such as in a database, in a message forum, visitor log, comment field, etc. This input will then be inserted into a web page and thus is displayed to all users who gain access to it (e.g., in an online bulletin board application).

The Reflected XSS Attack is the most common attack of web application at present[35]. A good example is when a user accesses the popular www.myonline-banking.com web site to perform sensitive operations (e.g., online banking). Unfortunately, input validation of the search form is not checked by the web site. Therefore, when a search query is entered that does not return any results, the message displayed also contains the unfiltered search string. For example, if the user enters a search string

```
<i>Hi There<i>
```

The italics markers are not filtered, and the browser of the user displays "No matches for *Hi There*" (note that the search string is displayed in italics). This is how a Reflected XSS vulnerability has been uncovered in this web application.[34],[30] The uncovered vulnerability can be then exploited by an attacker using the following steps:

1. The attacker writes a small JavaScript through which a victim's sensitive details will be passed to the attacker such as cookie address or session ID. This script is executed in the victim's browser.

2. The attacker then encourages the victim into visiting a website (a link) that points him/her to the form which contains the malicious script as an URL (GET) parameter. By emailing the link or sending it while chatting (e.g. on social networks) to the targeted user. This can include a few words that encourage the targeted user to click on the link which will enable the attacker to achieve his/her goal [**?**], [36]. When the user clicks on this link, the vulnerable application receives a search request similar to the previous one, where the search term was "Hi There". However, this time the search term is not "Hi There" any more, but a malicious script written by the attacker. Instead of a harmless phrase in italics, the victim's browser now receives malicious JavaScript code from a trusted web server. Therefore, the server will executes the malicious JavaScript code.

3. Consequently, the user's cookie, which can contain authentication credentials, is sent to the attacker. This is why this type of attack is called Reflected; the malicious code comes back to the victim's browser after being Reflected back by the server.

Apart from cookie stealing there is, potentially, another way for attackers to exploit Reflected XSS vulnerabilities. Assuming the web page mentioned in the previous example includes a login form along with the search form with JavaScript, the location to which a form sends the collected data can be modified. Therefore, the attacker can modify the malicious JavaScript snippet in order to redirect the login form to his/her server. When the user enters his/her name and password into the compromised login form and submits them, her/his credentials will be transmitted to the attacker [34],[35].

The difference between the Reflected XSS attack and Stored XSS attack is that the malicious script is not immediately reflected back to the victim by the server. A Stored Attack is recorded inside the targeted web application for later retrieval.

XSS Attack Consequences: The consequences of an XSS attack are the same regardless of what type the attack is All the three types of XXS attacks (Reflected or Stored or DOM Based) have the same consequences. The only distinction is the way of transmitting payloads to the server. XSS attacks do not exclude "read only" or "brochureware" site. An XSS attack can cause consequences for the victim from mere annoyance to complete account compromise. One of the most severe XSS consequences is when a user's session cookie is revealed which allows the attacker to hijack the victim's session ID and thus be able to manipulate their logged account (e.g. bank account) [25].

Other damaging attacks include the disclosure of the victim's files, the installation of a Trojan horse programs, redirecting the user to some other page or site, or modifying the presentation of its content. An XSS vulnerability allowing an attacker to modify a press release or news item could affect a company's stock price or lessen consumer confidence. An XSS vulnerability on a pharmaceutical site could allow an attacker to modify dosage information resulting in an overdose[26],[37],[27],[28],[30].

In this thesis, XSS attacks are carried out in order to have real-live data of the attack patterns. This attack is carried out manually and by automatically utilising some open-source testing tools. The aim is to be able to monitor a real world XXS traffic and be able to detect it for potential evidence recording. This attack is carried on the installed open-source web applications of our testbed. (See Chapter 5). A simple process of XSS attack is shown in Figure 4.14. However, the whole picture of the attacks' streams are shown later on in Chapter 5

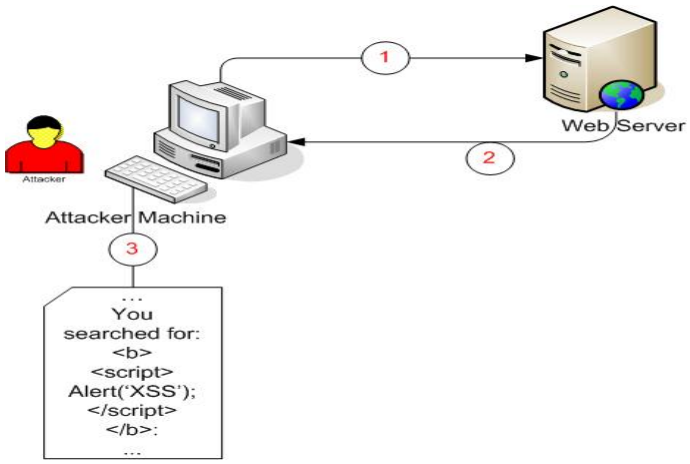


Figure 4.14: Simple XSS Attack Digram

4.2.2 Methodology Approaches

Before commencing web application attacks, a few tests can be made to uncover any potential vulnerabilities. These tests can be carried out with three main approaches. These approaches follow different mechanisms. While no mechanism can uncover all the possible vulnerabilities for a targeted web application, no one approach is better than the other is. The differences between them are based on the accessed resources availability to the tester.

White box: In the White box approach requires programming skills and an access to the targeted application resources including the source code and design specifications, in order to find the defective or vulnerable lines of code and fix them. This operation is often integrated into the development process by creating add-on tools for common development environments. [38], [39], [29]

Black box: The Black box does not require an access to the source code or any other resources and only sends random, unexpected or invalid data to the web application inputs [40].

Gray box: the Gray box testing is floating in between the white and black boxes. It provides more information than the Black box analysis through reserve code engineering (RCE) of available binaries. We will not discuss the Gray box as it is beyond the scoop of this thesis [22].

White Box Testing

A source code can be obtained either manually or automatically using certain tools that include time checkers, source code browser or automated source code auditing tools. An automation tool is a better way, as code lines can be numerous and searching manually would be time consuming. However, these automated tools can only suggest what suspicious codes need to be reviewed by a security specialist. The security specialist then needs to review the suspicious code and fix it [39], [29], [30].

A suspicious C++ code might include, for instance, "the strcpy function" which could cause a Buffer Overflow vulnerability if not used properly. The usage of strcpy() can be misused in C++. This is when a programmer misses checking the buffer which receives the inputs [22],[41]. The following example demonstrates a C++ weak code that can cause Buffer Overflow.

```
#include <stdio.h>
```

```
#include <string.h>

int main (int argc, char**argv) {
   int authentication (0);

  char cUsername[10], cPassword[10];
  strcpy (cUsername, argv[1] );
  strcpy (cPassword, argv[2] );
  if (strcmp(cUsername, "admin") == 0 && strcmp(cPassword, "adminpass") == 0){
  authentication = 1;
  }
if(authentication){
    printf("Access granted");
  } else {
    printf (\Wrong username and password");
  }
    Return 0;
  }
```

This code will not check the input and therefore can cause a Buffer Overflow attack. As is shown, only 10 bytes are allowed as inputs. So if we enter within the username field more than 10 bytes; for example 20 bytes e.g. (AAAAAAAAAAAAAAAAAAAA). This will cause a buffer overflow (overwriting) to the array in the password field of the memory. The entered input could be even bigger and bigger. Hence, automated tools will only indicate the suspicious parts of the code while it is the tester who must address its vulnerability. In such a case, the lengths of input must be checked to prevent such vulnerabilities[42]

Black Box Testing

Black box testing is considered as blind testing due to the randomness of fuzzing inputs to a web application [43], [28]. The end user determines the contents of that Black box as inputs, and subsequently he/she observes the

emerged output from the targeted application. While this process is happening, the end user has no clue what is going under the hood of that target. Black box testing can be carried out manually or with assistance of automated tools "Fuzzers"[44], [29] [30].

The technique of Black box testing will be used in the design of the thesis testbed. A certain number of black box Fuzzers will also be implemented to test the targeted web applications for uncovering potential vulnerabilities. This will depend on human analysis of the fuzzer's responses. Uncovering such vulnerabilities will enable us to carry out some attacks on the targeted web application. See Chapter 5 for more details. The following figure will show a simple process for a Fuzzer4.15.

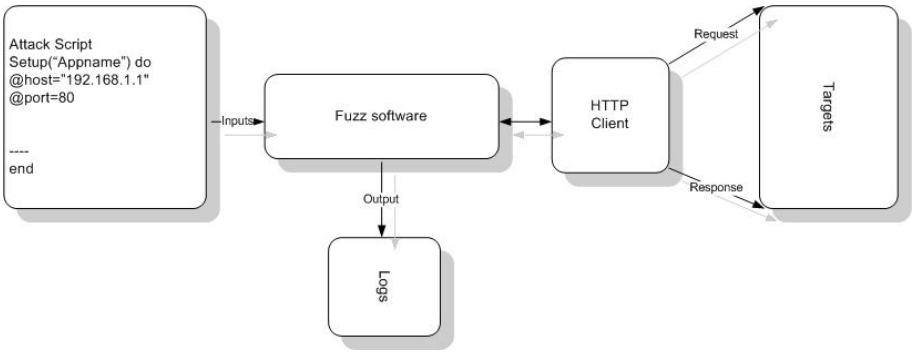


Figure 4.15: shows the process of a Fuzzer.[9]

Manual Testing Since Black box testing can be carried out manually, ordinary web browsers can be deployed to explore a web site hierarchy and insert dangerous inputs throughout the observed areas. For example to conduct an SQL Injection vulnerability adding the following characters: (' or '1'=1 –) will attempt to achieve this [26]. For uncovering traversal vulnerability, inserting the following: (../ ../ ../) and so on.

Automated Testing (Fuzzing) Fuzzing is a brute force technique that simply inserts random or invalid inputs into a web application. A Black

Box testing can be carried out as the researcher requires no knowledge of the internal workings of the application. Fuzzing must proceed through certain phases which start by identifying the specific target to fuzz and end with the determination of exploiting the found vulnerability. These phases are further discussed and implemented in the design of testbed chapter. (See Chapter 5, Section). The following figure demonstrates all the fuzzing phases (Fig2).

4.2.3 Methods

Before fuzzing any web application, the target environment must be set up to target the subsequent selection of the input vectors. "web applications present some unique challenges on both fronts. By design, the architecture of web applications can be deployed across multiple networked machines. Although this enables the scalability necessary when deploying such applications in production environment, it can also create performance hits that might not be desirable during fuzzing."[22] Far more important to understand is that data comes in various forms and can always be masked. Hence, web application inputs can always be "disguised" in many different ways while all lead to the same "vulnerabilities." This should be considered when defining inputs that are to be fuzzed [22].

Fuzzer's Ability

It is crucial to set up a capable fuzzing environment. When fuzzing a web application a large amount of inputs need to be generated and then fuzzed to the targeted web application. At the same time it is necessary to wait for them to be processed by the targeted web application while also listening for any reaction. A fuzzer must be able to do this easily and without any problems occurring. To achieve this we have to ensure a supported environment for the fuzzer. The supported environment can be evaluated from two perspectives: 1) fuzzing a local web application that is in same server of the fuzzer; or 2) fuzzing a web application within a network. The "bottleneck"

within the first tends to be CPU cycles and hard drive read/write times whereas the bottleneck within the second is caused by the transport of the network packets.

Fuzzer's Inputs

We have identified the various inputs of the targeted web application in order to fuzz them. (See Section**??**). This includes not only normal inputs to fill forms, but also URL (Uniform Resource Identifier) itself, cookies and HTTP headers. From a fuzzing perspective, all inputs that can be fuzzed and interpreted by the web server must be considered [22].

GET and POST Methods

The GET and POST methods are the core functions within an HTTP server. They are used to transfer data from a web page and display it. The GET and POST methods use name-value pairs for obtaining data from a web server. The name-value pairs can be submitted to the web server by both GET and POST. While GET submits the name-value pairs within the request URL, POST submits them as HTTP headers. The POST method enables unlimited size of data to be sent but it does not allows the URL to be shared to direct another user to an already generated page. There are also other valid methods including HEAD, DELETE, PUT, TRACE, CONNECT and OPTIONS [22].

4.2.4 Fuzzing Areas

Request-URL

After stating the method, the request-URL can be sent to the web server in order to identify the resource of the requested page. This is explained more in

the following example: “http://www.example.com/page.html”. Each component of the URL can be fuzzed as it is shown in the following example: /dir/page.html?name1=value1&name2=value2. This can be segmented as follows:

/[path]/[page].[extension]?[name]=[value]&[name]=[value]

While fuzzing data decoding should be considered, it can limit the input validation process. As it has been shown above, the Path component can uncover vulnerabilities that could cause Buffer Overflows or Directory Traversal vulnerabilities. These vulnerabilities can be identified by fuzzing a large amount of data, and fuzzing successive ../ character sequences respectively.

The page component can uncover vulnerabilities such as Information Leakage or Buffer Overflow. It can be tested by fuzzing random or common page names that could ultimately reveal sensitive pages. The Extension component can uncover vulnerabilities that are associated not with known page extensions like .php or .html but with unknown extensions.

Within the Name component, sending unexpected variables could lead the targeted application to malfunction if it does not handle errors properly. Conversely, vulnerability within this component can be uncovered by fuzzing common names that could result in undocumented variables, which can be eventually fuzzed to the server.

Within the Value component, random data can be fuzzed especially when they are different from what was expected to be an input. For example if the value length=50, then we can try to fuzz it with a higher or lower length or even negative. We also can fuzz it with an increasing number to see whether a Buffer Overflow would occur. The last component within the above example which can be fuzzed is the Separators that include (/, ?, =, &, %, $, ., :, etc). While fuzzing these characters, they can be exploited only when the

server does not have enough error handling [22].

HTTP Headers, Protocols & Cookies

There are also many other components that can be fuzzed within web applications including Protocols, Headers and Cookies. The HTTP protocol version can uncover server vulnerabilities. This can be fuzzed through fuzzing supported and unsupported versions. The most current popular version is HTTP1.1. This is clarified as HTTP/ [major version]. [minor version]. web application Headers can also be fuzzed to ensure whether the application supports them. Three potential components within a header can be fuzzed; Header name; the separator (:) and Header's value. The Header's format is [Header name]: [Header value]. Cookies can also be a fuzzing target. Since cookies are locally stored when a request is made to the web server, then they can come back as an HTTP header. The format of a cookie looks like: Cookie: [Name1]=[Value1]; [Name]=[Value]. The same fuzzing procedure that applied to HTTP Header can be also applied here.

4.2.5 Inputs Identification

In this section, we will discuss how inputs' fields can be conducted in order to identify all the legitimate input values. Identifying inputs can be carried out through a manual method or through an automated process. When fuzzing an application either way the following inputs should be identified; web pages, Directories, Method supported by pages, web forms including name-value pairs and hidden fields, Headers and Cookies.

The aforementioned can be simply identified through viewing the page source using the web browser. This allows anyone to identify the inputs included in web forms. Some applications include hidden input fields. Unfortunately, lazy developers try to implement security through obscurity by writing these kinds of input fields. This is a very weak control because hidden

fields can be seen through the page source code. Since the headers' request and response cannot be viewed through the page source, a particular type of software is needed. We have designed C++ software in order to interpret HTTP headers. This software allows us to view the request/response of headers; modify them and eventually post them to the application server. See Section 5.4.3.

HTTP Headers can be viewed through web browsers such as Mozilla Firefox using Add-ons features. To trace a web application hyperlinks fuzzing can be sufficient. This can be applied using a web spider (also known as web crawler). A web spider is an automated software that identifies all the visited pages within a web application. web spiders help to identify the inputs and thus enables them to be fuzzed.

4.2.6 Vulnerability Approaches and Attack Injection

In this section we discuss both uncovering vulnerabilities and security mechanisms that can be applied to web applications. The methodology for these mechanisms is based on the injection of realistic vulnerabilities and the subsequent controlled exploitation of the vulnerabilities that attack the system. This provides a practical environment that can be used to test counter measure mechanisms (such as IDS, web Application vulnerability scanners, firewalls, etc.), train and evaluate security teams, estimate security measures (like the number of vulnerabilities present in the code), among others.

We, however, have used completely vulnerable web applications (e.g. WebGoat). In order to define where a real vulnerability is likely to be located in the source code, we must find the difference between a vulnerable and a non- vulnerable piece of code, and sometimes how the vulnerability manifests itself.

This is carried out by utilising some manual methods as well as some

open source automated tools such as Fuzzers (e.g. JbroFuzz and Exploit Scanner). We also have used proxies to intercept HTTP requests as well as mySQL queries. The attacks are carried out by using other functions in these tools or by using other particular and more sophisticated functions. Most Fuzzers or Vulnerability Scanners include finding the vulnerability and injecting potential attacks automatically at the same time. (See Section 5.4).

The aforementioned tools (JbroFuzz and other tools) are used to inject attacks in a web application source code file (Figure 1). They start by analysing the source code of the target file while searching for locations where vulnerabilities can be injected (e.g. user-name and password parameters). While some tools require the user to specify the vulnerability targets, others recognise them automatically; they follow the actual patterns in the web application. Once a tool finds a possible location, it performs a specific code mutation in order to inject one or more attacks in that particular vulnerable location. This was carried out while we observed the results of such actions on the web application at the same time it was being intercepted by the HTTP proxy and IDS system. (See Chapter 6 on tests and results).

When attacking a particular vulnerability certain attributes must be followed. These attributes include the location pattern and the vulnerability code change. The location pattern defines the conditions that a specific vulnerability type must comply with [45],[46] . The vulnerability code change specifies the actions that must be performed in order to inject attack patterns (e.g. bad inputs) to the specified vulnerability. Where the vulnerability is to be injected depends on its environment. The following example clarifies the concept of simulated attacks through Fuzzing. For example, a web application has the following php code: "$id=intval($_GET['id']);" in its source code. Since the variable "$id" can be used in a query, then the Fuzzer will remove this function from the source code in order to inject a simulated attack (e.g SQL Injection with the input of "1 or 1=1"). By assigning the value "1 or 1=1" to the "$id" variable, the SQL query will execute without the effect of the "where" condition; therefore, affecting every row of the query[45].

The automated attack of a web application when one vulnerability is injected is done in two stages by the attack injection tool (e.g. JbroFuzz & HTTPFuzzer). The first stage is to collect all possible links of the target web application. This is called web crawling or spidering. The next stage is when a new interaction with the web application is performed but, this time, a collection of attack payloads is also injected in order to exploit the vulnerability by altering the SQL query sent to the database server of the web application. The interaction with the web application is always done from the web client's point of view (the web browser) and the payload is applied to the variables (the text fields, combo boxes, etc, present in the web page interface) [45],[46]. At the end of the attack, the fuzzer assesses if the attack was successful. This success is equivalent to the "error" state in a traditional fault injection technique.

However, whether the attack succeeds or not, the main point of this thesis is to monitor attack patterns, alerts and other relevant data from a forensic perspective. This is in order to record them in the database and therefore store them in the proposed DEB. The consequences of the performed attack are mostly dependent on how valuable the data stored in the database actually is. For those who are interested to know the consequences of performed attacks, the database itself can be checked after the attack was executed (e.g. SQL Injection).

4.3 Design of Digital Evidence Bag

4.3.1 A New Approach

While it is not possible to completely ignore all the current existing forensic tools, their techniques must be made more flexible. This should eventually lead to a new way of capturing and processing digital information in a forensically sound manner [10],[11]. During the last few years, the concept of

a Digital Evidence Bag (DEB) has been introduced and this should address these issues. A digital evidence bag is a wrapper for any type of digital based evidence or information [10],[9],[7]. Although the bag has a potentially infinite capacity, practically, the size will be limited and will also depend upon the user's requirements to obtain evidence in both a static and a real-time environment (the scope of this research). Each bag also has its own tag of information, complete with integrity assurance information and "continuity sections".

4.3.2 DEB Structure

The core structure of a digital evidence bag may contains three main files:

- Tag file;
- Index file;
- Bag file.

Both the bag file and the index file are called evidence unit (EU) [10],[7]. The EU (bag and index files) coupled with an index definition which enables the flexibility of the DEB frame work. This flexibility is required to use the selective and intelligent imaging techniques. At present a selective capturing process can be created in both manual and semi-automatic modes. In the manual processing capture the investigator captures information to both single and multiple EUs. This mode allows the category of the content to be recorded with an arbitrarily defined label [7].

Tag File The tag file contains details about the person who captures the evidence and the general information about the corresponding DEB. This could include: a reference number of the DEB; the details of the captured evidence; the name of the person who is taking custody of the capturing process; the date and time when the capturing process commenced. A unit is included containing the other two files: the index file and the bag file.

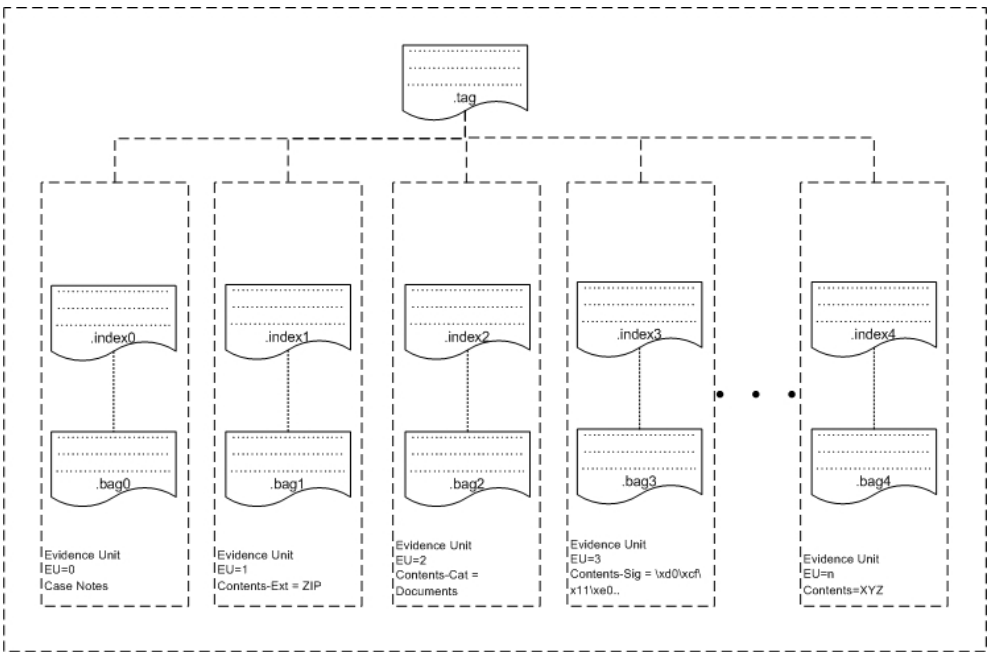


Figure 4.16: Digital Evidence Bag Framework

There is also a hash for integrity maintenance; a tag seal number that is equivalent to the traditional seal number; a continuity section that reflects the history of operations performed on the DEB and finally a definition of the index file [10]. The tag file is a plain text file comprising four main sections [7]:

1. DEB header
2. Evidence units
3. DEB footer
4. TCB

The first section of the DEB tag file is the header that contains information such as investigating officer, time-stamp of when the DEB was created, and description of what, where and when evidence was captured. The default content of the DEBs index files are specified within the DEB header " Index Format". The file format is defined by a sequence of meta-tags. This

enables EUs to be customisd within the DEB; therefore, allowing a DEB to contain information from several types of devices [7]. There are a wide range of index file meta-tags that can be used to present the stored information. These come in four categories:

- Labels: file name and path (F), origin description (P), file attributes (Fa), command (C)
- Timestamps: last modified/completed (Tmod), accessed (Tacc), created/ started/commenced (Tcre)
- Numeric: physical sector (PS), logical cluster number (LCN), file logical size (Fls), file physical size (Fps)
- Integrity – MD5 hash (Hmd5), SHA hash (Hsha)

The second section of the DEB tag file is the evidence units that is used to record all the created EUs in the DEB. These EUs have integrity hashes for their contents (both the index and bag files.) The first evidence unit (Unit 0) is used for recording the meta-data of the DEB as a whole. This meta-data can include creation time, capturing process, hash integrity of the used application, revision number, capturing process configuration file and details of capturing selection criteria. Other forms of meta-data can also be included such as photographs and free-form text.The third section of the tag file is the DEB footer section that records the number of EUs within the DEB. The DEB, subsequently, will be sealed and tagged with an integrity hash.While the last section of the DEB tag file is the tag continuity blocks (TCB). This section records the application function, signature and time-stamp of when the DEB was last accessed or modified [7].

Index File The index file should contain details about the bag file such as a list of file names, folder paths, and time-stamp information relating to the contents of the digital information in the bag. Alternatively it may contain details of the physical device.

Bag File The bag file should contain the actual evidence captured. This can be a structured text (e.g. from network packet capture), files (e.g from logical volume acquisition) or categorized files (e.g. several bags, each bag contains file of one type such as text files, MS word docs, JPEG files etc.) The bag files within each EU contain a link to each entry in the corresponding index file [7].

Chapter 5

Design and Implementation of a Testbed

5.1 Overview

This chapter describes the tools that have been used in the testbed to attack Web server and Database server systems and how other tools collect digital evidence resulting from these attacks. The novel approach of this thesis involves multiple attacks on multiple servers. The testbed architecture includes multiple IDSs, firewalls and databases and is used to monitor and gather data from different locations of the network. The data obtained is analysed and forensically stored within a collective database. This data, subsequently, is extracted by a DEB tool and stored in a Digital Evidence Bags (DEB) format. Open source tools are implemented for both data collection and data analysis. There are two reasons for choosing open-source tools: the first is cost-efficiency in an academic research environment. The second is that generally open-source programs are easier to deploy in heterogeneous networks because it is possible to adapt the software to the specific requirements of the environment.

The first two sections describe the tools used to set up the testbed and the network model for security and forensics tests. (See Chapter 6, on evaluation and results).

A list of the version or build numbers for the software used for this thesis is given towards the end of the chapter. The chapter ends with a discussion of unsuitable tools; that is software or hardware that have not been used in the experiments within the security architecture testbed or network simulation.

5.2 A Novel Approach

In this section, we have performed some practical experiments in the live testbed. Open-source tools that operate at different system levels(application, database and network), have been utilised in this testbed. In order to perform practical tests, a range of attack scenarios were performed on the testbed. Vulnerability and attack injection were applied in a set up based on web applications of different sizes and complexities. Results are further discussed in Chapter 6.

5.2.1 Motivation

One factor that motivated us to do this particular experiment was the idea of performing multiple attacks on multiple servers. We were very interested to have a testbed which consists of multiple servers and most importantly multiple intrusion detection systems.

Another factor was that there is also a shortage of research in the combined area of security and forensics. Looking at them from a practical perspective it could be said they are even one entity. If a system needs to be highly secured it also needs to be structured in such a way to enable investiga-

tors to collect evidence. No entity is completely secure and so when security is violated, there must be a clear system for collection of the evidence. Not all systems will be the same and what system is appropriate will depend on the type of attack and the type of investigation required. However, investigators can take the initiative and build their own systems within a forensic environment.

5.2.2 Concept

The Proof of Concept of this research shows how security and forensics are connected and why they should always be considered as one component. It also shows the importance of being secure and of always being aware of the potential for a future need for digital evidence. This statement can be reduced to the main goal of building a highly secured and forensically sound system. It is proposed in this thesis that the design of a Correlation Engine and the proposed Digital Evidence Bag framework are able to collaborate in a security and forensic implementation.

5.2.3 Testbed Architecture

In order to have a practical environment, the testbed consists of three subnetworks. There were two reasons for this. The first reason was that most organisations have a model of three subnetworks model (external, DMZ and external). The second reason was that in this research we wanted to be able to demonstrate and actually perform the concept of multiple IDSs among various networks. This was to allow us to have a few IDSs (slaves) which enabled us to collect their feedback from these subnetworks into a single IDS server (master). See Figure 5.1

External subnetwork: this subnetwork is directly connected to the Internet through the main router. It is connected to a central network which is in fact the DMZ subnetwork which is the joint between the external and

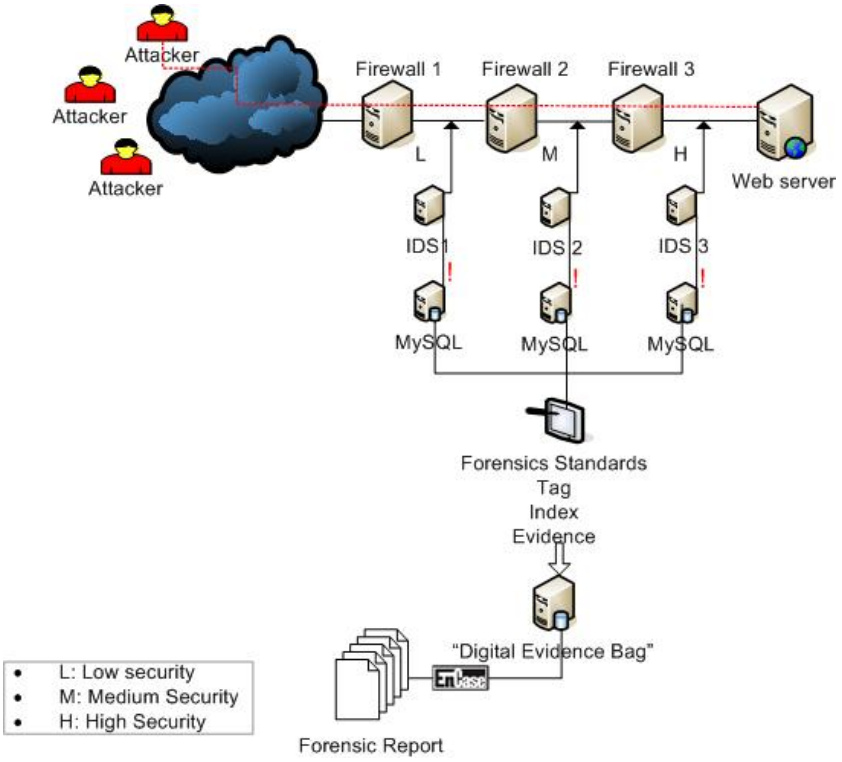


Figure 5.1: Testbed Architecture

the trusted networks

DMZ subnetwork: this is the central subnetwork in our testbed which utilises another computer as an additional AP. The purpose of a DMZ is to add an additional layer of security to the internal local area network (LAN); an external attacker only has access to equipment in the DMZ, rather than any other part of the network. DMZ is widely used in real world scenarios. This subnetwork includes a few services like HTTP, FTP, DNS...etc. We also use this subnetwork as the centre for our testbed as it incorporates the data correlation and collation machine. This data is actually the IDSs alerts which result from potential internal or external attacks.

Internal (trusted) subnetwork: this is connected to the DMZ via a firewall but it cannot communicate with the external (untrusted) subnet-

work. This is the most secured subnetwork among all three. It cannot be subjected to any external attacks; however, it can be subjected to internal attacks. For this reason this subnetwork interface was monitored but with fewer attack rules compared to the most vulnerable subnetwork which is the external one.

5.3 Attack Scenarios

As described in Chapter , a range of invalid injection attacks were performed in order to present this thesis. This included SQL Injection and Cross Site Scripting attacks. We also performed some Denial of Service attacks. These were designed to keep the monitoring processes apart from an ongoing attack. Because the Administrators do not have enough time to trace all of the attacks our aim this was to divert their attention from the monitoring process. See Section 4.2.1, Chapter 4.

5.3.1 First Scenario

This scenario represents an attacker's attempts to take advantages of the vulnerability of an SQL Injection in a web server. The attacker starts his attacks by checking whether the targeted website is vulnerable. This can be carried out in three ways: automatically, semi-automatically and manually. The following figure shows the main diagram of the data flow in our testbed. When looking at it, it may be helpful to consider it as an extracted piece of the whole testbed picture. (See Figure 5.2).

One way to exploit an SQL Injection vulnerability is using hacking tools. This is the most sophisticated but easiest and fastest way of uncovering SQL Injection vulnerabilities and exploiting them. Hacking tools can be exceedingly sophisticated but utilising some of them does not require a major human analysis. In fact, most script kiddies learn how to use them and start exploiting SQL Injection vulnerabilities without even knowing the basic ar-

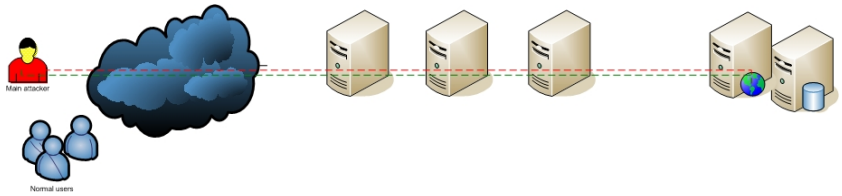


Figure 5.2: The Data Flow of The Attacker Trying The First Scenario

chitecture of the a database. They even can misuse penetration testing tools in order to exploit SQL Injection vulnerabilities and thus gain credential details from the back end database server.

Another way of uncovering SQL Injection vulnerability is to deploy a fuzzing tool and watch the web server responses (See Section 4.2, Chapter 4). This way is faster than the manual one but still also requires a further human analysis to complete the attack. One more way to discover an SQL Injection vulnerability is manually, the attacker misuses the web application input fields by entering bad inputs. This can include any type of Blackbox data. It can range from using a simple code to building a complete hacking tool.

In our testbed we have used all the above ways while monitoring them and recording attacking attempts in a forensically sound manner. The attacker attempts to find whether the web application is vulnerable by using any of the above ways. While the attacker is doing this the IDS starts alerting these suspicious activities. It is from this point that the forensic recording must start. In case the attacker gets through, there would be valuable evidence of his attempts before the attack was completed. Of course there is a possibility of having some false negative alerts but that would not be an issue when an operator is undertaking normal actions. For example if a user started typing bad inputs by mistake, this would not last long before the user realised their mistake and altered their inputs. It would be clear whether he/she was making normal mistakes or was actually attempting to break into the database

and obtaining sensitive data (e.g. admin username and password)

This scenario raised two elements in our experiment which can be utilised in security and forensic research. The first element was that being able to collect valuable evidence before an attack actually strikes on the web server. The second element was that being able to reduce the possibility of of Zero-day attacks from happening. This is further discussed in more dtails towards the last chapter (See Section 7.2, Chapter7).

5.3.2 Second Scenario

This scenario shows how multiple attacks were carried out simultaneously. This can happen when an attacker(s) (e.g. three hackers) collaborate to launch attacks at a certain time or date. For example, we set up a group of two attackers (virteual machines) that attacked the web server with different attacks (DOS, SQL Injection and XSS). The basic idea behind simulating this attack is to have a few attacks underway while only one of those attacks is intended by the attacker. This is to lure the monitoring processes way from the main attack while hiding the pattern of that main attack or at least diverting the administrators' attention because they do not have enough time to trace all the attacks. These different attacks may come from different sources. We used automated tools to attack the targeted web server while we concentrated on the main attack. (See Figure 5.3).

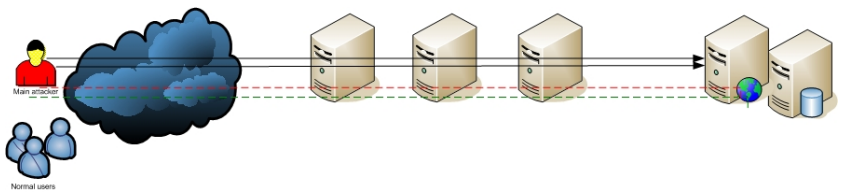


Figure 5.3: The Data Flow of The Attacker Trying The Second Scenario

5.3.3 Third Scenario

This scenario presents the most sophisticated attack that web servers have suffered in recent years (DDOS). This scenario shows the complexity of such attacks with suggestions that can be used in order to mitigate the severe consequences of these attacks. The idea behind this scenario is to have a large number of attackers from different locations launching the same attack at the same time.

Since this scenario is near to impossible to simulate, we tried a similar scenario by inviting some outsiders to attack our web server in the DMZ network. Attackers from hacking forums and social networks were invited to hack into our website. These attackers were encouraged to use open-source tools including Slowhttptest[1], Nmap and fuzzers for their attack on the web server. We also performed some DDOS from internal virtual machines. See Chapter 6 for results. For this scenario data flow in our testbed (See Figure 5.4).

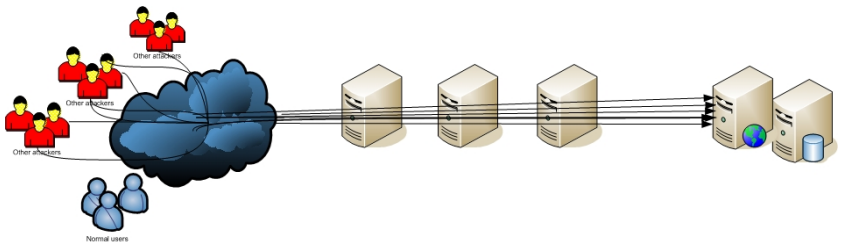


Figure 5.4: The Data Flow of The Attacker Trying the Third Scenario

In this testbed we found that the second scenario is very similar to the third scenario. However the third scenario consumed a lot of CPU resources

[1]Slow HTTP DoS attacks rely on the fact that the HTTP protocol, by design, requires requests to be completely received by the server before they are processed. If an HTTP request is not complete, or if the transfer rate is very low, the server keeps its resources busy waiting for the rest of the data. If the server keeps too many resources busy, this creates a denial of service. This tool is sending partial HTTP requests, trying to get denial of service from target HTTP server.

compared to the second one. It was even hard at times to navigate between applications. The difference between the second and third scenario was that the second scenario focused on different attacks coming from one source or only a very small number of sources; whereas, the third focused on one type of attack coming from a large number of sources.

5.4 Testbed Tools

This section describes the tools used inside our testbed. It describes the main machines we have used and both their hardware and software components. Further explanation is completed on each tool and the reasons we chose it in this thesis.

5.4.1 Virtual and Physical Machines

Virtualisation has made the development and evaluation of many different types of software, from operating systems to graphical user interfaces and even malware, much easier. VMware Workstation allows a user to clone existing virtual systems (VMware images). Therefore, we used virtual machines in order to save time setting up identical or nearly identical operating systems compared to the physical computers. Additionally, the internal state of these images were observed at some stages for malicious activities and were reset to a healthy state when needed.

At times an IDS could not recognise some virtual network interfaces so physical machines as well as virtual ones were at times used in our testbed for the sub networks. However, every single sub network has its own physical machine in the testbed. Each physical machine includes some virtual machines when required. Most of the machines were Linux based but only a few were Windows. This enabled us to experiment on different operating systems and have the flexibility of ultlising different software. For example

some tools only work on windows while we need to use them (e.g. Visual Studio) so virtual machines sloved this problem in our experiment.

5.4.2 Hardware Tools

The testbed was established with a main router (D-Link, DSL-2730B), three PCs and 5 NIC. Each PC had (Linux Ubuntu 11.04) loaded. The first two PCs were utilised and turned into routers in order to have enough CPU power and RAM storage. This enabled us to perform complex functions with devices able to handle them smoothly. These functions include packet analysis and loading any software that was required.

5.4.3 Software Tools

In Chapter 4, we described why we chose certain attacks (SQL Injection and XSS).(See Section 4.2.1, Chapter 4). In this section we elaborate on the tools used for carrying out these attacks. While some attacks were carried out manually, others were carried out using the following tools including sqlmap, havij, JbroFuzz and HTTPFuzzer. Our reasons for choosing these particular tools and not others are mentioned under each tool heading.

sqlmap:

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and takes over the database servers. It comes with a detection engine and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections. We have used it through the BackTrack operating system [24].This tool is chosen in our thesis because it's been widely used and recommended in a few references including [47] and [48]. According to [49], sqlmap was downloaded over seven million times.

havij:

Havij is an automated SQL Injection tool which was utilised for penetration testing in order to uncover and exploit the SQL injection vulnerabilities of web applications. We chose this tool because of the high success rate of injectiong vulnerable targets. This success rate was more than 95% according [50]. It can take advantage of a vulnerable web application and enable users to perform back-end database fingerprint, retrieve DBMS users and password hashes, dump tables and columns, fetch data from the database, run SQL statements and even access the underlying file system while executing commands on the operating system [50].

JBroFuzz:

JBroFuzz is a web application testing tool (fuzzer) that utilises both HTTP and HTTPS requests. JBroFuzz aims to provide a single, portable application which enables web protocol testing capabilities. We used JBroFuzz to initiate requests and transfer them to a web applications as we watched the corresponding responses of these requests. However, this required further analysis to determine whether we complete the attack or not.

The first important thing to analyse was the response of the numeric status code to HTTP requests. For more details about HTTP and HTTPS numeric status code (See Appendix A). See Figures 5.5, 5.6 and 5.7.

Certain attack payloads included in fuzzers were used to generate requests such as XSS and SQL injection and are crafted to successfully exploit flaws. Such flaws represented previously known vulnerabilities for web applications. JBroFuzz grouped fuzzers, with their corresponding payloads into a number of categories, depend on previously known vulnerabilities.Therefore, the human analyst would need to select the fuzzers to use in order to test a

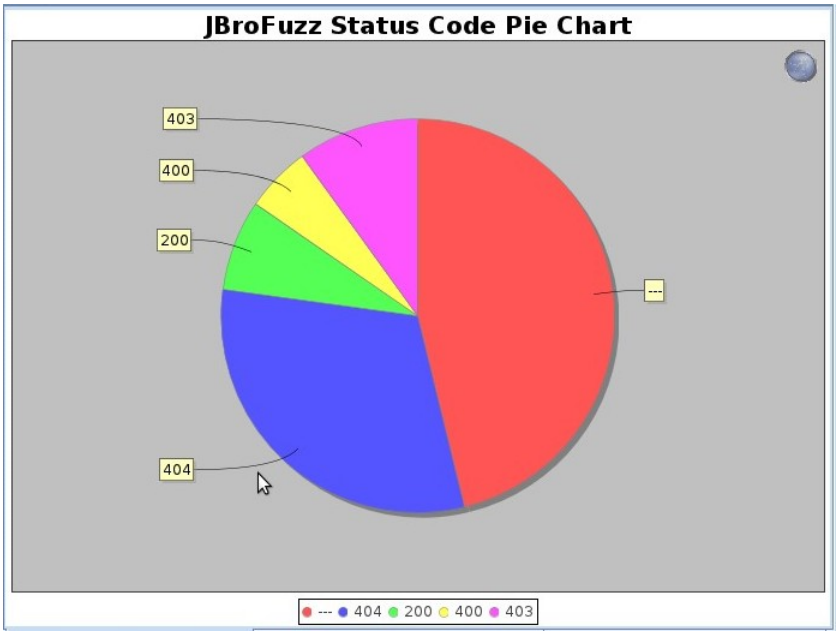


Figure 5.5: Numeric Status Code Responses of Multiple Attacks (SQL Injection, XSS ...etc) carried out on HTTP Requests

particular set of vulnerabilities and review the results in order to recognize if exploitation had succeeded or not [51].

HTTPFuzzer:

HTTPFuzzer is a web application fuzzer that we have created using C++ programing language. This fuzzer enables us to experiment on a more dynamic way with parameters inside a web page. It generates requests and puts them in the wire and waits for corresponding responses for further analysis. SQL Injection and XSS payloads are included in the designed software. (Figure X)shows a snapshot of the HTTPFuzzer in action.

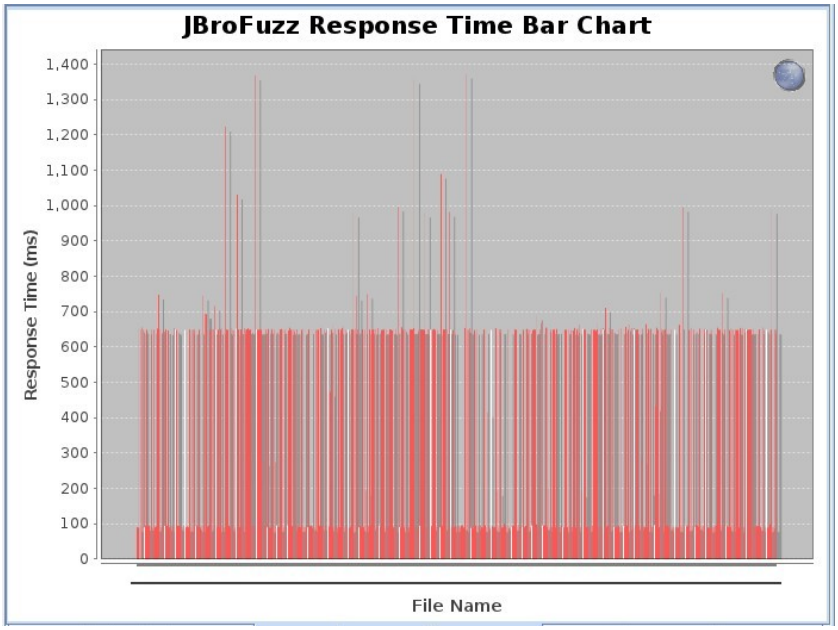


Figure 5.6: Response Time of the previous Multiple Attacks(SQL Injection, XSS ...etc) carried out on HTTP Requests

5.5 IDSs and Event Correlation Engine

5.5.1 IDS (Snort)

Intrusion Detection System (Snort): Snort is an open-source network intrusion detection and prevention system (IDS/IPS) developed by Sourcefire. We have applied Snort to our research as it combines the benefits of signature, protocol, and anomaly-based inspection and it is one of the most widely deployed IDSs/IPSs technologies worldwide. See Figure **??** for a simple snort rule which was used in this testbed.(http://www.snort.org/)

Snort's main purpose was to monitor the network traffic on the interfaces for potential attacks or any security related issues. For instance, it monitored the network interfaces for potential exploits or just network probes (e.g. nmap), which is used for acquiring information about the targeted system. All this was accomplished by Snort utilising existing signature based

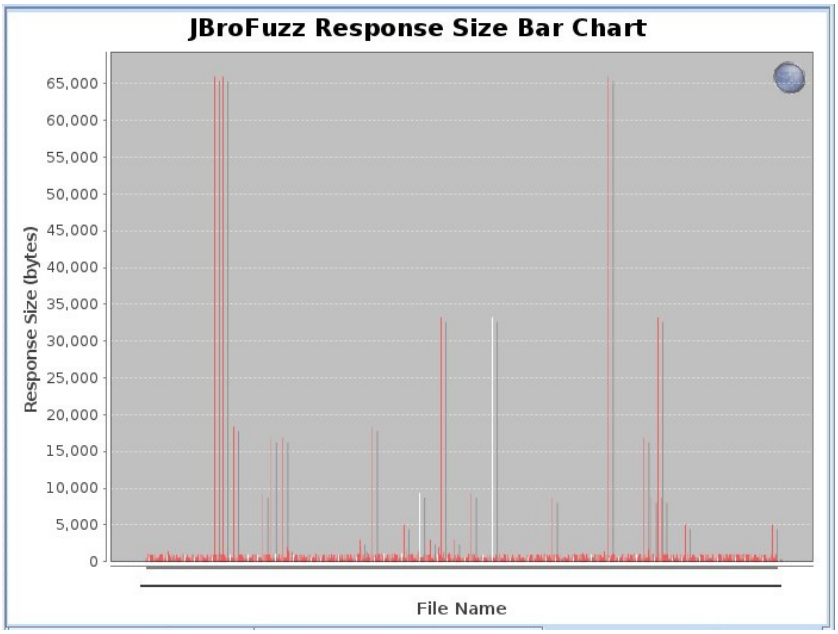


Figure 5.7: Response Size of the previous Multiple Attacks(SQL Injection, XSS ...etc) carried out on HTTP Requests

```
alert tcp \$SQL_SERVERS 1433 -> \$EXTERNAL_NET any (msg:"SQL sa brute
force failed login unicode attempt"; flow:from_server,established;
content:"L|00|o|00|g|00|i|00|n|00| |00|f|00|a|00|i|00|l|00|e|00|d|00|
|00|f|00|o|00|r|00| |00|u|00|s|00|e|00|r|00| |00|'|00|s|00|a|00|'|00|";
detection_filter:track by_src, count 5, seconds 2; reference:bugtraq,
4797; reference:cve,2000-1209; reference:nessus,10673; classtype:
unsuccessful-user;sid:3273; rev:5;)
```

Figure 5.8: A Snort Rule that was Implemented on this Testbed to Catch Potential Attacks on the Database Server

rules. When these rules match any suspected behavior, Snort send alerts in a unified2 format file. This unified2 file is immediately read by a database logging tool Barnyard2 (See Section5.5.2), and logged to a database where we can view it and see whether real attacks are under way.

```
alert tcp \$EXTERNAL_NET any -> \$HTTP_SERVERS \$HTTP_PORTS (msg:"SQL
oversizedcast statement - possible sql injection obfuscation"; flow:
established,to_server; content:"CAST|28|"; nocase; isdataat:250,relative;
content:!"|29|"; within:250; metadata:policy security-ips drop, service
http; reference:url,isc.sans.org/diary.html?storyid=3823; classtype:
web-application-attack; sid:13791; rev:1;)
```

Figure 5.9: Another Snort Rule that was Implemented on this Testbed to Catch Potential SQL Injection on the Database Server

As Snort reported any suspected activities which matched the established rules, firewalls also were applied to the interfaces in order to slow some attacks from being successful. However, highly secure configured firewalls were not highly considered in this research as we were focusing more on the forensic part of collecting evidence from the IDSs

In order to avoid potential errors Snort was built from the source code in this research. However, Snort can be downloaded as a pre-built package: this is likely to be much easier for users. The Sourcefire team, the makers of Snort, also release pre-built packages of Snort. At the time of writing this thesis, the used (latest) version is "snort-2.9.0.5.tar.gz".

For Snort rules capturing attacks, we have applied both the Sourcefire Snort rules[52] and Emerging threat rules [2][53]. They were downloaded and updated on a frequent basis and placed in the /etc/snort directory(Linux) where rule files are normally placed. Nevertheless, a rule file can be put in any directory while bearing in mind changing the "RULE_PATH" of the configuration files.

The network model has three different sub networks. Each sub network is presented with one physical machine which has two interfaces (eth0:eth1). Each physical machine may include virtual machines when needed. The first

[2]Emerging Threats is an open source community project that produces the diverse Suricata and Snort Rulesets and firewall rules available.

interface of the first sub network is the Internet facing one (eth1). This is the Interface that connects the network/IDS system to the Internet. See the Appendix for the base configuration file used for all interfaces.

IDSs interfaces are set with different rules ranging from highly secured to weak. This can decrease and increase the threat levels and hence enable us to carry out easy and highly sophisticated attacks. These interfaces are being monitored for everything in the proposed LAN network including attacks and malware. The interfaces configuration of the sub networks is very similar, but with different network definitions.

The following command is used to start Snort on eth0:

```
sudo /usr/local/snort/bin/snort -u snort -g snort -c
/usr/local/snort/etc/snort.conf -i eth0
```

Figure 5.10: Running Snort on eth0

The following figure shows Snort in action as it's logging events in a unified2 format file. (See Figure 5.11).

5.5.2 Database Logging (Barnyard2)

Barnyard2 that is an output system for Snort. Barnyard reads this unified2 file, and then resends the data to a database backend. Unlike the database output plug-in, Barnyard is aware of a failure to send the alert to the database, and it stops sending alerts. It is also aware when the database can accept connections again and will start sending the alerts again. This Proof of Concept design had to have a single place where all security related events could be viewed[54].

A critical design concept was important for all the data (alerts) to be dumped in a unified2, single console for viewing. Unified2 is a special binary output format that is created by Snort. A system that used several pieces of

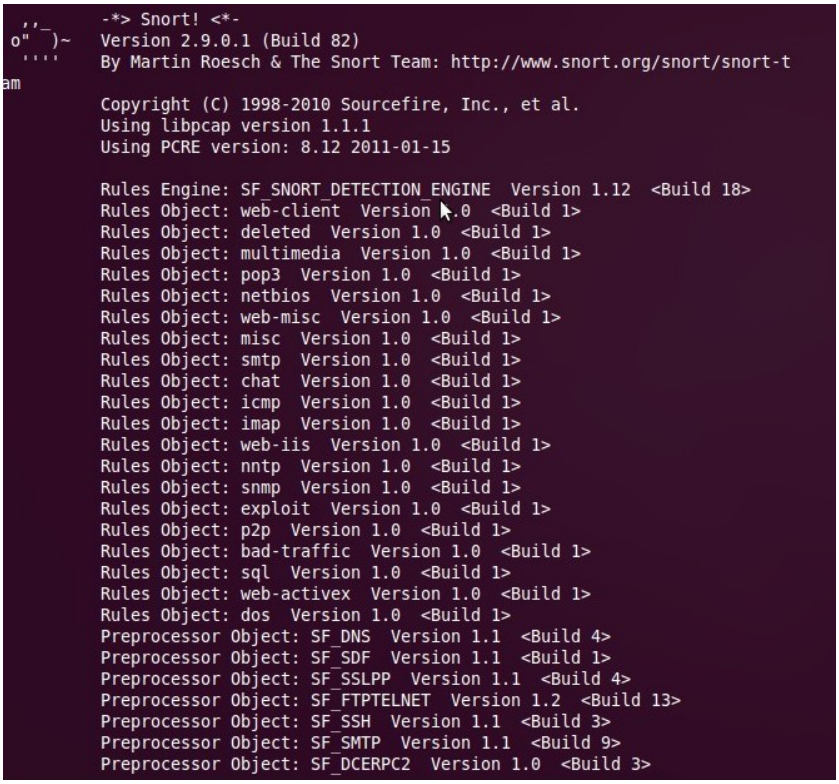


Figure 5.11: Snort (First Sensor) in Action for Logging Snort's Alerts into a unified2 format file

software which dumped data to various places was not an option; therefore, we implemented Barnyard2

As it is shown in figure 5.12 that Snort did not send alerts to the database (MySQL). The reason for this is because Barnyard2 was utilised for data correlation and collation. Barnyard2 was set in order to obtain data from Snort and send it to the back end database server.

All IDSs were configured to use the "output unified2:" option. As Snort detected potentially hostile traffic, it logged it to a file using unitied2 format. This enabled Snort to record the alert and rule that was triggered, as well as the packet information (packet dump). Barnyard2 therefore "read" this file, in real time, and send it to our SQL back end servers. The underlying idea

behind this approach is to take the burden of dealing with logging processes from Snort.

In other words, Snort monitored the subnet interfaces in real time. This is because If an untoward interruption occurs Snort would temporarily stop processing network traffic to log this information to the SQL database. This is not satisfactory, as we would potentially miss other hostile traffic while Snort is logging this information to the SQL server. In order to deal with this, a separate process was started by Barnyard2 which obtained the information from Snort, via the unified2 output format, and inserted it into the SQL database for Snort; thus off loading that process from Snort to Barnyard2. Since Snort now does not need to concentrate its efforts on SQL logging, it can continue monitoring the network interface.

```
output unified2: filename snort.eth0.log, limit 128 #Internal interface
output unified2: filename snort.eth1.log, limit 128 #Internal interface
```

Figure 5.12: Each IDS has been configured to log alerts to its own individual "unified2"

textbfnote The 'limit' option means the unifed2 file size limit. In our cases, we are setting this to 128 megabytes

Since have a total of three instances of Snort running, we ran three instances of Barnyard2. Each one read a unique Snort unifed2 formatted file. Each Barnyard2 instance logged to the database MySQL back end as a separate "sensor id". In the Snort MySQL database in the 'sensor' table, the 'sensor id' is known as the 'sid'.

Data from each specific instance of Barnyard2 was loaded to a unique sid. In the end, this allows us to specify traffic by interface from the database. Ultimately we were able to view data from the sid # 1 (Exteranl subnetwork) while ignoring sid's #2 (DMZ) and #3 (Trusted subnetwork). We

were also able to combine all the information in order to have a complete view of all the data. After creating the database and tables of the back end server, we utilised the open-source tool (OpenVPN) to create a tunnel between IDSs and the back end server. This was to ensure the integrity of the data when collecting information from Snort via Barnyard2 and sending it over the OpenVPN to the back-end (MySQL) server.

The following figure demonstrates the database and tables establishment.As stated above, in this research as mentioned before, we are using the open-source database (MySQL schema).

```
#/usr/local/mysql/bin/mysql -u root -p < ./contrib/create_mysql snort
```

Figure 5.13: The database and its tables establishment

Once the database is created, we assigned the appropriate grants/rights to the database for the remote IDS systems to be able to store information.

```
# mysql -u root -p snort
mysql> grant INSERT,SELECT on snort.* to snort@IP-Address identified by
'password';
mysql> grant INSERT,UPDATE,SELECT on snort_sensor to snort@IP-Address identified
by 'password'
mysql> quit
```

Figure 5.14: The database and its tables establishment

note In too many cases users grant "ALL" to remote IDS/IPS systems. We only want to give our IDS/IPS system the rights that it needs! If an attacker gained access to the system and stole the database credentials, he/she

would have access to ALL the back end database. We do not want that to occur.

With the grants/rights in place, we configured Barnyard2 to read the unifed2 output from Snort and inserts data into the database server. As with Snort where we separated configuration files via interface (ie - snort.eth0.conf, snort.eth1.conf), we have done the same with Barnyard2 (barnyard.eth0.conf, barnyard.eth1.conf). For example, the eth1 interface configuration file of Barnyard2.eth1.conf may look like the following

```
22. ##############################################################
23. # /etc/barnyard2/barnyard2.eth1.conf - Internal interface.
24. ##############################################################

26. config reference_file: /etc/snort/reference.config
27. config classification_file: /etc/snort/classification.config
28. config gen_file: /etc/snort/gen-msg.map
29. config sid_file: /etc/snort/sid-msg.map
30. config hostname: host2-internal
31. config interface: eth1
32. config set_gid: snort
33. config set_uid: snort
34. config waldo_file: /var/log/barnyard2/barnyard2.eth1.waldo
35. input unified2

37. output database: log, mysql, user=snort password=mypassword dbname=snort
host=Remote-IP-Address, detail full
```

Figure 5.15: Routine to Enable Barnyard2 for Logging Events on eth1 to The Database

The following command is used to start Barnyard: The following figure shows Barnyard2 in action as it's reading the event file of Snort in a unified2 format and logging its data to the database. See Figure 5.17

```
sudo /usr/local/bin/barnyard2 -c /usr/local/snort/etc/barnyard2.conf -G
/usr/local/snort/etc/gen-msg.map -S /usr/local/snort/etc/sid-msg.map -d
/var/log/snort -f snort.u2 / -w /var/log/snort/barnyard2.waldo
```

Figure 5.16: Barnyard2 starting up. the 'database' information is shown

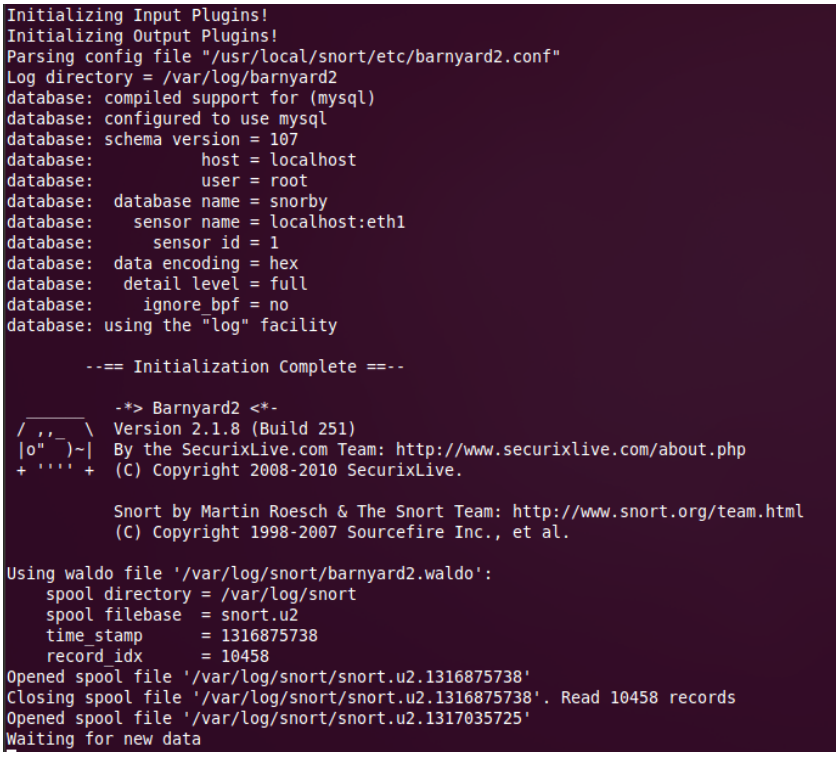


Figure 5.17: Barnyard2 (First Sensor) in Action for Logging Snort's Alerts to a Database

5.5.3 Web-Interface (Snorby)

Snorby is an open-source tool that integrates with intrusion detection systems such as Snort. It is a very useful tool for combining data, showing actionable metrics and retrieving data from different IDS sensors. Snorby was utilised in this thesis for data correlation using MySQL queries to store all information in a combined database. This is the database from which our designed DEB tool extracts digital forensic evidence. (See Chapter 6).

The configuration part was very much the same as Barnyad2. However, instead of having Barnyard2 reading snort unified2 file and logging it to the Barnyard2 database; we had Barnyard2 configured to log snort unified2 file to a Snorby database. Snorby database has more functions than Barnyard2 database. Snorby works very well with Snort and Barnyard2. See the Tests and Results in (Chapter 6), for data extraction and the results achieved combining these open-source tools.

Chapter 6

Tests and Results

6.1 Overview

The previous chapters described the novel security and forensics architecture. In this chapter, the results from applying multiple IDSs, launching and detecting attacks and correlating their resulted alerts are all presented with potential security mechanisms to both mitigate attacks and with the ability to trace them back to their sources. A novel architecture of Digital Evidence Bags is also demonstrated in this chapter.

This chapter starts with an overview of the evaluation and follows this up with the results of the attacks; the qualitative results from the PoC testbed, and the quantitative results from the network simulation which are further analysed. Known and yet to be solved problems encountered during the experiments are described at the end of this chapter.

6.2 Evaluation Process and Input Data

In order to conduct an evaluation on our designed testbed, certain security measurements have to be considered. The qualitative and quantitative methods used in the evaluation part of this thesis are described in the following

sections.

Input data used for the security and performance experiments can come from two different sources. These are network traffic from the real world and/or constructed data. The constructed data can be either intentionally generated to target known vulnerabilities or it can be randomly generated through fuzzing techniques (See Fuzzing, Section 4.2, Chapter 4).

Input validation attacks (SQL Injection and XSS) from the network traffic of the real world were captured and analysed using open-source web applications which were enabled site to public. These web applications were selected for two reasons: Moodle web application is largely used among education organisations and is reasonably secure; whereas, WebGoat is used for penetration testing training and thus is completely vulnerable. The aim was to have a web application with potential vulnerabilities for XSS and SQL Injection. This shows that not to have a completely secure web application but highly secure intrusion detection systems architecture for monitoring bad behaviours might occur in the network peripheries. See table2 for open ports which have been left on the web server (See Table2- nmap-sS output)

Network traffic related to input validation attacks in particular (XSS & SQL Injection) and other web application attacks, that have resulted from the real world, were captured and analysed using open-source tools including Snorby and SnortReport and the power of MySQL database (See Design of correlation engine - Chapter 4). The scenarios of the aforementioned attacks are already discussed in the previous chapter (See Attacks Scenarios-Chapter 4)

Based on the results from the multiple sensors of the IDSs and from other reports on malicious http-traffic, specific attacks (XSS and SQL Injection), reports were generated and sent to the main IDS in the DMZ section of the network. This has been discussed in the previous chapter. (See Network Architecture, Chapter 4)

6.3 Multiple IDSs Results

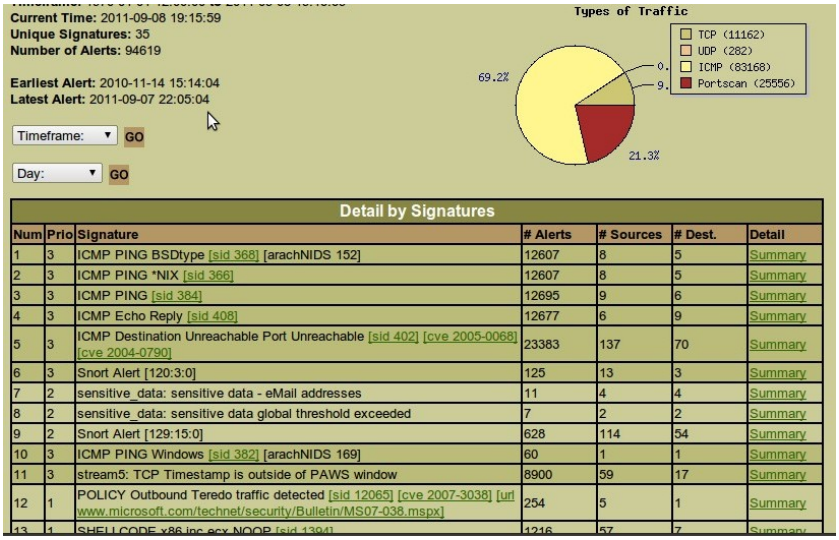
The multiple IDSs collected data during two time-frames. The first was designed to generally monitor any potential attacks on the network. This monitoring was for a period of about 9 months. (See Figure 6.1). The second time-frame was designed for more specific attacks and scenarios. It was monitored over a period of 30 days.

6.3.1 First Time-Frame

The first time-frame of monitoring potential threats to the network was set to catch all attacks with no specifications. This concentrated more on monitoring attacks without the application of the aforementioned scenarios explained in the previous chapter. Figure 6.1 shows the distribution of attack signatures which occurred in the period from 14/11/2010 (at 03:14 PM) to 07/09/2011 (at 11:05 PM). It also shows the types of traffic that mostly came through ICMP protocol; whereas, the least came through UDP protocol. This also has happened with the second time-frame.

During this time-frame 94,619 alerts was triggered mainly on ICMP protocol with low severity. Most of the alerts triggered were due to internal bad behaviours such as attempts to gain a higher privilege or Brute force attacks that were mainly carried out internally by internal users.

However, many attacks did not succeed due to the robust security and the fact that we did not make a public invitation for attackers to break into the network. SnortReport was the main tool used to analyse the collected data over this period. See Figure...



Detail by Signatures

Num	Prio	Signature	# Alerts	# Sources	# Dest.	Detail
1	3	ICMP PING BSDtype [sid 368] [arachNIDS 152]	12607	8	5	Summary
2	3	ICMP PING *NIX [sid 366]	12607	8	5	Summary
3	3	ICMP PING [sid 384]	12695	9	6	Summary
4	3	ICMP Echo Reply [sid 408]	12677	6	9	Summary
5	3	ICMP Destination Unreachable Port Unreachable [sid 402] [cve 2005-0068] [cve 2004-0790]	23383	137	70	Summary
6	3	Snort Alert [120:3:0]	125	13	3	Summary
7	2	sensitive_data: sensitive data - eMail addresses	11	4	4	Summary
8	2	sensitive_data: sensitive data global threshold exceeded	7	2	2	Summary
9	2	Snort Alert [129:15:0]	628	114	54	Summary
10	3	ICMP PING Windows [sid 382] [arachNIDS 169]	60	1	1	Summary
11	3	stream5: TCP Timestamp is outside of PAWS window	8900	59	17	Summary
12	1	POLICY Outbound Teredo traffic detected [sid 12065] [cve 2007-3038] [url www.microsoft.com/technet/security/Bulletin/MS07-038.mspx]	254	5	1	Summary
13	1	SHELLCODE x86 inc ecx NOOP [sid 1394]	1216	57	7	Summary

Figure 6.1: SnortReport Showing Details on Some Alerts Sorted by Signature

6.3.2 Second Time-Frame

The second time-frame was far more interesting. This was because the proposed scenarios were applied to the experiment. Web applications were also installed and an invitation to the public and to hacking forums and social networks was then made. The data was collected over a period of 30 days. Figure shows the distribution of hacking attempts that hit the network on a daily basis for the period of observation from Thursday 01/09/2011 (at 12:00AM) to Friday 30/09/2011 (at 11-59-PM)

Severities:

During the period of the second time-frame, 66,719 alerts were triggered; (438) with high severity, (60552) with medium severity and (15704) with low severity. These were mainly related to web application vulnerabilities. During this time-frame a different tool (Snorby) was utilised to analyse the collected data.

Figure 6.3 shows the traffic on a day when the testbed was attacked

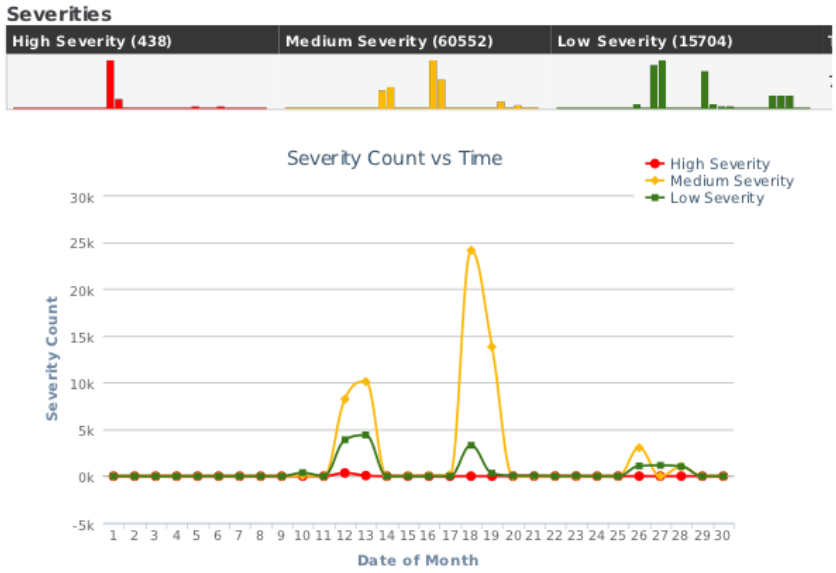


Figure 6.2: Severities of Attack During the Second-Time Frame

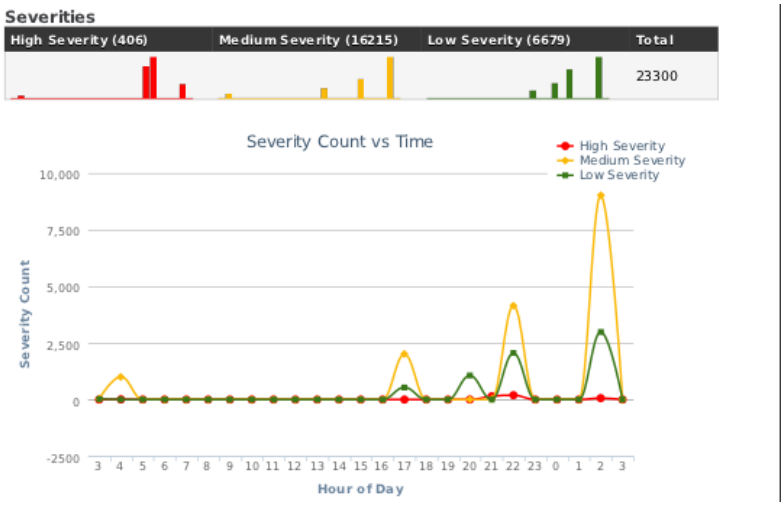


Figure 6.3: Severities of Attack During the Second-Time Frame

with a severity which ranged from a low number to a large number of attacks. During the day, the Second Scenario of this testbed (Section 5.3.2) was also implemented and it is shown in the Figure 6.3. Our aim was to show how a large number of attacks could be deployed to hide other intended at-

tacks. When we ran multiple attacks the sensor triggered so many alerts that the monitoring process was diverted. An attacker is therefore able to hide his/her actual intended attack by generating a high number of alerts so that the alerts related to the intended attack go unnoticed. This is shown in Figure 6.3 where the Red indicates the alerts of the intended severe attack, and the Green and Yellow indicate the other unimportant attacks that were triggered to make the Red go unnoticed.

Attack Protocols

Similar to the first time-frame, most attacks' traffic came through ICMP protocol; whereas, reasonable amount of attacks' traffic came through TCP protocol. The interesting factor happened during this time-frame was that no traffic came through UDP protocol at all. (See Figure 6.4).

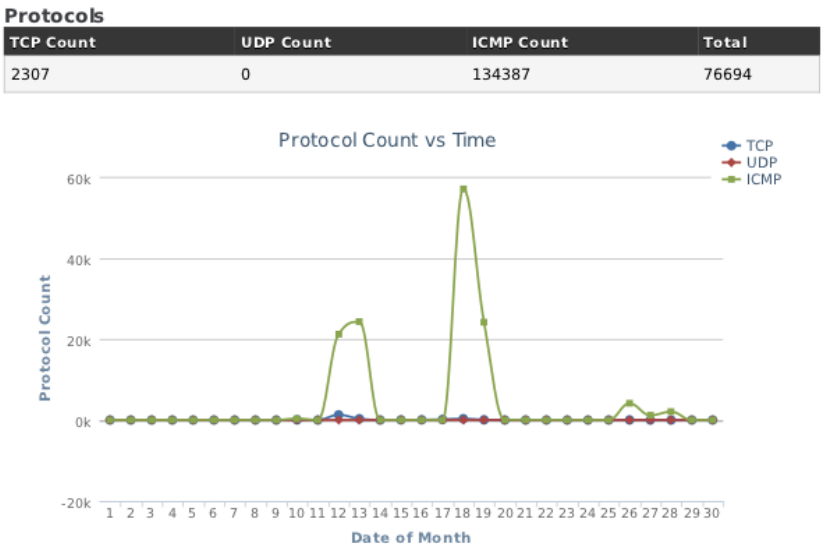


Figure 6.4: Protocols Were Used to Attack the Web Application

Attack Signatures

The number of hacking attempts per day ranged from a low of Zero to a high of 24087 alerts. The average number of triggered signatures per day, or mean, was $\mu = \frac{76708}{30} \approx 2557$ attempts per day. The Median is 23.5. All type of alerts were included in the triggered signatures including the low severity attacks that were carried out on the testbed. The standard deviation for the number of alerts per day is

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \approx 6273.97$$

Top 15 Signatures

Signature Name	Percentage	Event Count
frag3: Fragments smaller than configured min_fragment_length	74.55%	35940
ICMP PING	7.28%	3511
ICMP Echo Reply	7.28%	3510
ICMP PING *NIX	7.27%	3504
ICMP Destination Unreachable Port Unreachable	1.19%	574
SHELLCODE x86 inc ecx NOOP	0.88%	425
http_inspect: LONG HEADER	0.85%	408
ICMP Time-To-Live Exceeded in Transit	0.16%	75
Snort Alert [129:15:0]	0.14%	66
SNMP AgentX/tcp request	0.12%	59
SNMP request tcp	0.12%	59
http_inspect: U ENCODING	0.06%	29
sensitive_data: sensitive data - eMail addresses	0.02%	12
ICMP PING Windows	0.02%	8
stream5: TCP Timestamp is outside of PAWS window	0.01%	7
ICMP Destination Unreachable Communication with Destination Ho...	0.01%	7
http_inspect: NON-RFC DEFINED CHAR	0.01%	6
sensitive_data: sensitive data global threshold exceeded	0.01%	4
ICMP PING BSDtype	0.01%	3
SHELLCODE x86 inc ebx NOOP	0.01%	3

Figure 6.5: Top 15 Signatures Occurred During the Month

The mode of the most signatures occurred was: “frag3: Fragments smaller than configured min_fragment_length” with a percentage of 74.55%. This occurred 35940 times in the whole month. This was when we were attempting to flood the sensor with high load of traffic in order to blind the monitoring process while running other separate attacks including SQL Injection and XXS. See Figure 6.5) for the Top 15 Signatures was triggered in the whole month.

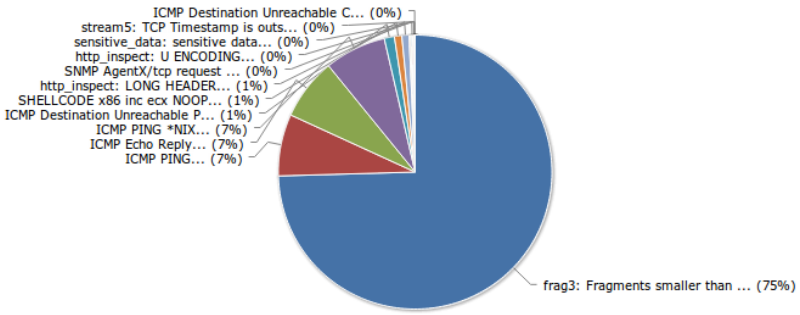


Figure 6.6: Another Diagram Showing Largest 12 Occurred Signatures During the Month

Attack Source Addresses

The source addresses for most occurred attacks were mainly coming through internal machines with IP Addresses 192.168.1.6, 192.168.1.5 and 192.168.1.2. Some attacks came from New Zealand while others came from other parts of the world including USA, Australia, China and Japan. There were a few other attackers from other countries but the attacks they performed were not major. (See Figure 6.7).

Top 10 Source Addresses

Source IP Address	Percentage	Event Count
192.168.1.6	52.27%	39878
192.168.1.5	28.34%	21623
192.168.1.4	15.73%	12001
192.168.1.2	1.48%	1128
192.168.1.1	0.71%	544
202.36.179.68	0.35%	266
60.234.52.128	0.21%	164
203.97.30.147	0.17%	132
66.94.240.25	0.1%	76
98.137.51.1	0.1%	74

Figure 6.7: Top 10 Source Addresses Produced Attacks During the Month

Attack Destination Addresses

The top 10 destination addresses were definitely our internal servers. As it is shown in Figure 6.8, most destinations were within the internal network of the testbed. These internal destinations IPs start with 192.168.1.x. Whereas, 121.74.226.149, was the public IP of the web application we enabled on the web server.

Top 10 Destination Addresses

Destination IP Address	Percentage	Event Count
192.168.1.4	83.75%	63899
192.168.1.2	8.64%	6595
192.168.1.5	3.63%	2773
121.74.226.149	2.68%	2046
192.168.1.6	1.33%	1013
192.168.1.1	0.39%	297
203.97.30.147	0.04%	30
74.125.127.139	0.04%	27
69.171.229.11	0.01%	5
74.125.127.138	0.0%	2

Figure 6.8: Top 10 Destination Addresses Received Attacks During the Month

6.4 A Proposed Digital Evidence Bag

Since we have now monitored, collected and traced these attacks, it is time to produce the evidence of these attacks that can be incorporated into a trusted container (DEB). All this hard work is to support our proposition that Security and Forensic is one unit that cannot be separated. As with a thief, in the end a clear legal valid case must prove that a crime has been committed.

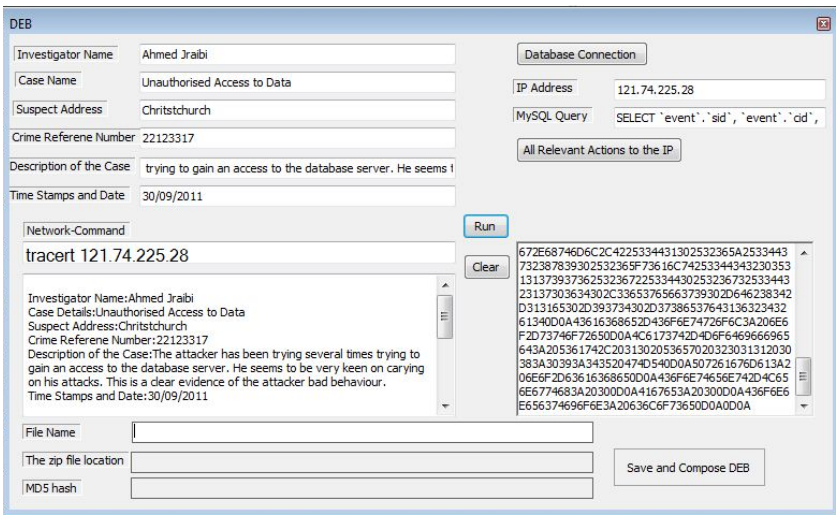


Figure 6.9: DEB Interface in Process for Acquiring Evidence and Wrapping it with MD5 Hash

The PoC of this DEB is to apply the ideas of Turner's approach [9] to having a global DEB that can be used to collect evidence from any application. In our DEB we prove that a DEB can incorporate evidence from intrusion detection systems and multiple sources such as command line tools (See Table). It enables investigators to establish a connection to the main database of the IDS and command line tools. From this point the power of open-source MySQL queries can be used to retrieve data from the database in a secure way. This connection must have the same, or with save, connection (e.g. encrypted tunnelling). After all the necessary data has been

retrieved into a file, this file can be both composed and encrypted with Md5 hash. (See Figure 6.11).

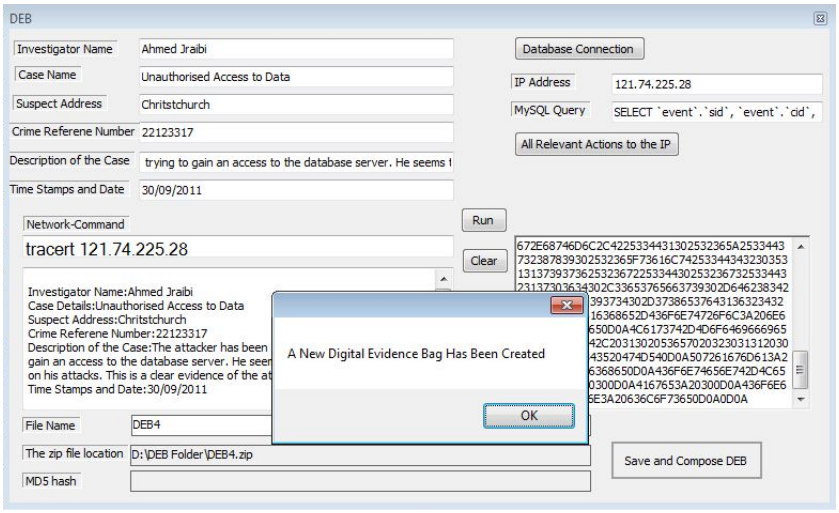


Figure 6.10: DEB Has been Created and Wrapped

This will save time for investigators when acquiring evidence especially in a live-forensic technique. This also applies to those investigators who use open-source command-line tools and write down their outputs or manually copy them to a file. Using this method prevents them from tracing back an ongoing attack. The following figure is a demonstration of the designed PoC DEB which describes a case of an attacker attempted to insert an SQL Injection into the database server. We will use that information to show how evidence can be wrapped into a container (DEB). (See Figure 6.11).

6.5 Conclusion

In conclusion, most attacks came from the external network targeting web applications' vulnerabilities in general. Most attackers seem to use script kiddies tools to perform their attacks. Some attackers use fuzzers and hacking tools such as Wa3f. Different operating systems vary among attackers

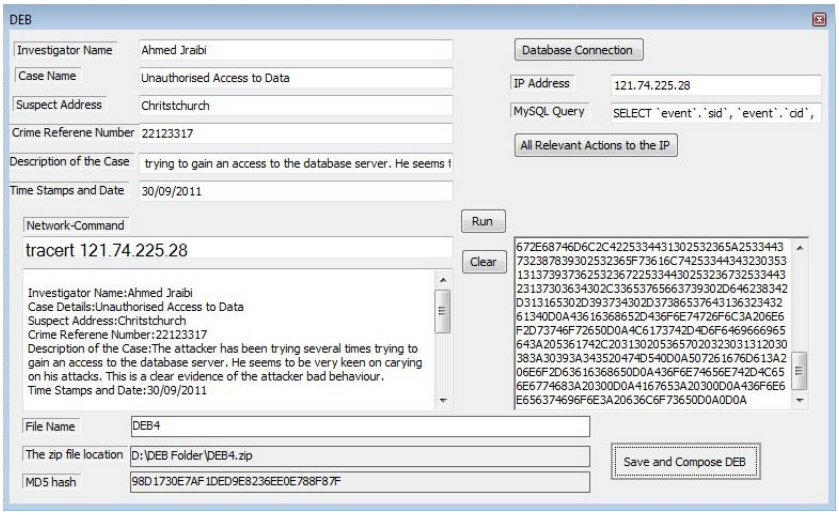


Figure 6.11: DEB Composed and Saved with MD5 Hash

with the minority using Linux while the majority use Windows. Different IP addresses from different countries have been captured by the IDS including New Zealand, USA, China. etc.

The use of a rich diversity of sensor information may achieve the development of more reliable IDS. The rationale behind this is that sensor variety is needed because each sensor perceives different information depending on its capabilities, its function and where it is deployed in the network. The amount of information required to infer malicious activity using distributed heterogeneous sensor architectures would overwhelm any human network manager and automatic processing becomes necessary.

6.5.1 Thesis Limitations

Although multiple intrusion detection systems offer extensive detection capabilities, they do have some limitations. These include the analysis of encrypted network traffic, the handling of high traffic loads, and their apparent limited ability to withstand attacks against the intrusion detection systems

themselves. These limitations are discussed below.

Encrypted network traffic including virtual private network (VPN) connections, HTTP over SSL (HTTPS), and SSH sessions cannot be detected by network-based IDSs. However, some IDSs can perform a limited analysis of the setup of encrypted connections. They can identify the client or server software, whether it has known vulnerabilities or is misconfigured. Therefore, for example, in order to ensure an analysis is performed on the payloads within an encrypted network traffic, IDSs should be placed in such a position as to enable an analysis of the payloads before they are encrypted or after they are decrypted.

Intrusion detection system sensors themselves were susceptible to various types of attacks. Certain type of tools (e.g. Nmap, DoS attacking tools) were used from different virtual machines to generate large volumes of traffic in a short period of time. Two of the three attacking scenarios, that were deployed in our experiment for this thesis, caused a major problem for the IDSs (sensors). The goal of those two scenarios was to confuse the monitoring process by flooding the sub network with a large volume of network traffic, or by generating network traffic that was likely to trigger many IDS alerts in a short period of time. This was intentionally carried out in order to divert the administrator's attention from another, separate, attack which was under way. (See Section 5.3).

Large volumes of traffic was generated as the sensor was reporting anomalous activity (Attempted Denial of Service) which apparently exhausted the sensor's resources. It cased the sensor not only to generate too many alerts and thus enabling the intended attack alerts to go unnoticed but to slow it down to such an extent that it ultimately failed.

Chapter 7

Conclusion and Future Work

7.1 Summary

This thesis has described the situation of multiple threats and vulnerabilities on multiple servers. It has also discussed the need to have multiple intrusion detection systems based on different networks. The foundation of this thesis was the importance of multiple entities. These entities included, attacks, vulnerabilities, servers: firewalls, IDSs, databases and web server. All these entities were combined in a novel testbed based on an active network which enabled us to join individual strengths together and to overcome their specifc weaknesses. Another significant factor of this research was the creation of an unique DEB along with IDSs output and open-source networking tools (e.g. whois). Existing solutions and ongoing research in the area of security (IDSs), and forensics (DEB) were described and analysed for their effectiveness on the thesis testbed. The results from the Proof-of-Concept testbed showed that a security architecture of multiple IDSs for large networks can be implemented using Unix-based systems and free open-source tools.

The results of the multiple attacks showed, under some circumstances, that it was possible to prevent attacks. When a server is attacked, it is likely that other servers in the same sub network or nearby will also be attacked. Our aim was to protect multiple servers on a network by preventing attacks

from spreading to other servers on the network. If there were a web server in a network then it is possible that a database server exists on the same sub network or nearby. Therefore, if we received an XXS attack on the web server, then it was likely that we would also receive attacks on the database server (e.g. SQL Injection). We also noted that when an attacker wanted to attack a particular target, he/she would test the vulnerable aspects of the target before starting the real attack. This gave us an indication of when a potential attack was coming and; therefore, when other servers on the network should be protected.

The results from these multiple IDSs showed the enormous benefits of having a centralised database of multiple IDSs (sensors). By using multiple sensors and combining their outputs we were able to enhance the security of the implemented network. This should apply to any other network regardless how big or small it was. This also showed that internet security as a whole can be improved by having multiple sensors and centralised databases all over the internet through which suspicious behaviour can be monitored. These databases could be analysed and utilised for security purposes in order to prevent a potential threat from spreading over the internet. However there were shortcomings to this approach. There was the possibility of a sensor crash due to a large volumes of traffic such as distributed denial of service (DDoS) attacks and anomalous activity such as fragmented packets with the aim to exhaust a sensor's resources or cause it to crash. Suggestions to overcome these shortcomings are further discussed in a future work section. (See Section 7.2).

The last importnat part of the thesis was the acquired digital evidence from both networking command line tools as well as the centralised database of the multiple IDSs through a novel DEB tool. The results from the Proof-of-Concept (PoC) of our testbed indicated that it is possible and crucial to have an open-source DEB that can be connected to the IDSs. This thesis indicates that it is possible to include a DEB within the IDSs utilities. Many third party tools to handle IDSs alerts were used in this thesis. These in-

cluded Snorby and SnortReport which in themselves, DEB functions should be incorporated. This is for future work so that potential evidence can be encapsulated and exported as a wrapped DEB in a forensically sound manner.

7.2 Future Work

7.2.1 Performance

Since the thesis' testbed architecture was a Proof-of-Concept implementation utilising tools that enabled the thesis' author to demonstrate the use of existing open-source tools, performance was not considered a major factor in the thesis. However, if custom built software is used with high featured hardware tools, performance enhancement will be gained. This needs further research beyond the Proof-of-Concept and some indications are included in this thesis.

7.2.2 Realistic Attacks Data

To capture real-world malicious attacks network data, two sub networks (external and DMZ) were exposed to the Internet with low to medium security. A few ports were open and web applications were available for the public. Because the testbed was confined to two small public subnets, the results were not extensive enough to be representative of the whole Internet. However, real attacks were performed from within the network and from the Internet. While the internal attacks were very satisfying and produced the significant part of the results, the outsider attacks we received from the internet were not that satisfactory. The reason for this was probably more elements were required to attract hackers and crackers from the internet such as a honepot which could be considered for future work.

7.2.3 Extensions and Generalisation

The scope of this thesis was established to research, analyse, and improve the security and forensics of a large network. The main concentration was the implementation of multiple intrusion detection systems with the aim of obtaining evidence extracted through a digital evidence bag tool. The Proof-of-Concept can be applied to any type of network, whether it is a small or a large network, and can be extended even further. This is due to the large load of network traffic that will need to be handled. This extension requires more research and the implementation of a live network with advanced hardware tools. Although, some virtual machines were used for this thesis, generally speaking, they should not be used for major elements in the testbed but can for minor elements. This is in order to have more realistic testbed and thus a more real-world measurement for the performance For example for a router and servers in the sub network there must be physical machines but these sub networks can be extended from the inside. This extension can be through implementing virtual machines simulating attacks from within the network.

Further research and more experiments can extend the security architecture we have developed to a general security architecture for all network traffic. The testbed and network can also be extended to provide a more accurate model of the real world. The centralised IDSs' databases can be analysed and bad behaviours or patterns can be further analysed for improving Internet Security. The analysis of collaborated IDSs patterns can track down Distributed Denial of Service attack (DDOS) and possibly prevent or at least mitigate Zero-Day attacks from spreading over the Internet in a short period of time.

When the suspicious activity is being recorded, the source of these suspicious activities can be stopped at an early stage of before they can proceed further. For example, when the attacker starts checking the database server to see whether it is vulnerable to SQL Injection or not, forensic recording will have started and so any traffic from the same source can be stopped at that

point from further accessing the network. This can be done automatically by setting a software running between the firewall and the IDS together with the automatic creation of new rules.

Further research on the DEB architecture and the functions of the IDSs could lead to a significant contribution in the area of security and forensics. An open-source DEB tool must be established sooner or later. The proposed DEB in this thesis is able to deal with databases and command-line networking tools and extracts digital evidence from them. However, this DEB can be extended to extract digital evidence from more tools and software. The bottom line is having a tool that can deal with various other tools and extract digital evidence from them in a forensically sound manner. The DEB of this thesis was a PoC and didn't have enough time and resources to complete it. It is to be carried out in thr future for further development and reseach.

Bibliography

[1] J. Wiles and A. Reyes, *Computer Forensics in Today's World*. Burlington: Syngress, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B8KJR-4S69PSH-4/2/34773ec41cdbbfe874dee6fac6660c1c

[2] E. S. Pilli, R. Joshi, and R. Niyogi, "Article: A generic framework for network forensics," *International Journal of Computer Applications*, vol. 1, no. 11, pp. 1–6, February 2010, published By Foundation of Computer Science.

[3] S. R. Selamat, R. Yusof, and S. Sahib, "Mapping process of digital forensic investigation framework," *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, no. 10, pp. 163–169, 2008.

[4] A. Ganame, J. Bourgeois, R. Bidou, and F. Spies, "A global security architecture for intrusion detection on computer networks," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–8.

[5] J. Nehinbe, "Log analyzer for network forensics and incident reporting," in *Intelligent Systems, Modelling and Simulation (ISMS), 2010 International Conference on*, 2010, pp. 356–361.

[6] C. Lin, L. Zhitang, and G. Cuixia, "Automated analysis of multi-source logs for network forensics," *Education Technology and Computer Science, International Workshop on*, vol. 1, pp. 660–664, 2009.

[7] P. Turner, "Selective and intelligent imaging using digital evidence bags," *Digital Investigation*, vol. 3, no. Supplement 1, pp. 59–64, 2006, the Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06). [Online]. Available: http://www.sciencedirect.com/science/article/B7CW4-4KCPVBY-2/2/44b61e9eabd53bf3dbaeea1e9a1bfcea

[8] D. Haagman *et al.*, "Good practice guide for computer-based electronic evidence," accessed April 2011. [Online]. Available: http://computer-forensics.7safe.com/the-acpo-guide-for-electronic-evidence/

[9] P. Turner, "Applying a forensic approach to incident response, network investigation and system administration using digital evidence bags," *Digital Investigation*, vol. 4, no. 1, pp. 30–35, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B7CW4-4MT5K2R-2/2/1a08ca3fff0601ae12fee6ffa70b97d6

[10] ——, "Unification of digital evidence from disparate sources (digital evidence bags)," *Digital Investigation*, vol. 2, no. 3, pp. 223–228, September 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B7CW4-4GWBF1D-1/2/847d1447394144c7dcd47bdd270a1acc

[11] I. G. G. Richard and V. Roussev, "Next-generation digital forensics," *Commun. ACM*, vol. 49, pp. 76–80, February 2006. [Online]. Available: http://doi.acm.org/10.1145/1113034.1113074

[12] H. Zeidanloo, A. Bt Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani, "Botnet detection based on traffic monitoring," in *Networking and Information Technology (ICNIT), 2010 International Conference on*, 2010, pp. 97–101.

[13] Y. Zeng, X. Hu, and K. Shin, "Detection of botnets using combined host- and network-level information," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 2010, pp. 291–300.

[14] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, 2007, pp. 715–720.

[15] W. Hailong and G. Zhenghu, "Heterogeneous multi-sensor information fusion model for botnet detection," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, vol. 2, May 2010, pp. 428–431.

[16] M. del Mar Fernandez and I. Porres, "An evaluation of current ids," Master's thesis, Linköping University, SE-581 83 LINKÖPING, Sweden, February 2008. [Online]. Available: urn:nbn:se:liu:diva-11635

[17] M. Govindarajan and R. Chandrasekaran, "Intrusion detection using neural based hybrid classification methods," *Computer Networks*, vol. In Press, Uncorrected Proof, pp. –, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-51S25P9-2/2/b7c2acd2944f945e67112a98f37a8c02

[18] A. Servin and D. Kudenko, "Multi-agent reinforcement learning for intrusion detection," in *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, ser. Lecture Notes in Computer Science, K. Tuyls, A. Nowe, Z. Guessoum, and D. Kudenko, Eds. Springer Berlin / Heidelberg, 2008, vol. 4865, pp. 211–223. [Online]. Available: http://www.springerlink.com/content/e2471134g8531018/

[19] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "Mcpad: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128608003927

[20] *Combining Multiple Intrusion Detection and Response Technologies in an Active Networking Based Architec-*

ture, June 2003. [Online]. Available: http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/telematik/Mitarbeiter/schaefer/Publications/

[21] *HB 171-2003 Guidelines for the management of IT evidence*. Standards Australia, 2003. [Online]. Available: http://infostore.saiglobal.com/store/Details.aspx?productid=568739

[22] M. Sutton, A. Greene, and P. Amini, *Fuzzing: brute force vulnerabilty discovery*, ser. Safari Books Online. Addison-Wesley, 2007. [Online]. Available: http://books.google.co.nz/books?id=D2VGAAAAYAAJ

[23] OWASP, "Sql injection," April 2011, accessed July 2011. [Online]. Available: https://www.owasp.org/index.php/Category:Attack

[24] J. Clarke *et al.*, *SQL Injection Attacks and Defense*, 1st ed. Burlington, MA: Syngress, May 2009. [Online]. Available: http://www.syngress.com/hacking-and-penetration-testing/SQL-Injection-Attacks-and-Defense/

[25] OWASP, "Cross-site scripting (xss)," 2011, accessed July 2011. [Online]. Available: https://www.owasp.org/index.php/Category:Attack

[26] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of sql injection and cross-site scripting attacks," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 199–209. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070521

[27] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *Proceedings of the 13th international conference on World Wide Web*, ser. WWW '04. New York, NY, USA: ACM, 2004, pp. 40–52. [Online]. Available: http://doi.acm.org/10.1145/988672.988679

[28] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in *Security and Privacy (SP), 2010 IEEE Symposium on*, may 2010, pp. 332–345.

[29] W. Tian, J. Xu, K.-M. Lian, Y. Zhang, and J. feng Yang, "Research on mock attack testing for sql injection vulnerability in multi-defense level web applications," in *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, dec. 2010, pp. 1–5.

[30] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/1135777.1135817

[31] RSnake, "Xss (cross site scripting) cheat sheet esp: for filter evasion." [Online]. Available: http://ha.ckers.org/xss.html

[32] Wardg, "Web security," accessed May 2011. [Online]. Available: http://code.google.com/p/doctype/wiki/ArticleXSS

[33] G. Ollmann, "Understanding the cause and effect of css (xss) vulnerabilities," in *HTML Code Injection and Cross-site scripting*. [Online]. Available: http://www.technicalinfo.net/papers/CSS.html

[34] A. Kieyzun, P. Guo, K. Jayaraman, and M. Ernst, "Automatic creation of sql injection and cross-site scripting attacks," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, may 2009, pp. 199–209.

[35] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/1135777.1135817

[36] DP, "Critical facebook xss bugs could be used to hijack accounts," Sep. 2010. [Online]. Available: http://www.xssed.com/

[37] H. Shahriar and M. Zulkernine, "Mutec: Mutation-based testing of cross site scripting," in *Proceedings of the 2009 ICSE Workshop on Software*

Engineering for Secure Systems, ser. IWSESS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 47–53. [Online]. Available: http://dx.doi.org/10.1109/IWSESS.2009.5068458

[38] P. Godefroid, A. Kiezun, and M. Y. Levin, "Grammar-based whitebox fuzzing," *SIGPLAN Not.*, vol. 43, pp. 206–215, June 2008. [Online]. Available: http://doi.acm.org/10.1145/1379022.1375607

[39] P. Tonella and F. Ricca, "A 2-layer model for the white-box testing of web applications," in *Web Site Evolution, 2004. WSE 2004. Proceedings. Sixth IEEE International Workshop on*, sept. 2004, pp. 11–19.

[40] K. Haller, "White-box testing for database-driven applications: a requirements analysis," in *Proceedings of the Second International Workshop on Testing Database Systems*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 13:1–13:6. [Online]. Available: http://doi.acm.org/10.1145/1594156.1594172

[41] G. Janardhanudu and K. van Wyk, "White box testing," Sep. 2009. [Online]. Available: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/white-box/259-BSI.pdf

[42] B. Ornarli, "C++ buffer overflow exploit," Dec. 2008. [Online]. Available: http://www.infernodevelopment.com/c-buffer-overflow-exploit

[43] A. Lanzi, L. Martignoni, M. Monga, and R. Paleari, "A smart fuzzer for x86 executables," in *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, ser. SESS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 7–. [Online]. Available: http://dx.doi.org/10.1109/SESS.2007.1

[44] H. Dai, C. Murphy, and G. Kaiser, "Configuration fuzzing for software vulnerability detection," *Availability, Reliability and Security, International Conference on*, vol. 0, pp. 525–530, 2010.

[45] J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability #x00026; attack injection for web applications," in *Dependable Systems Networks, 2009.*

DSN '09. IEEE/IFIP International Conference on, 29 2009-july 2 2009, pp. 93–102.

[46] I. Elia, J. Fonseca, and M. Vieira, "Comparing sql injection detection tools using attack injection: An experimental study," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, nov. 2010, pp. 289–298.

[47] A. Ciampa, C. A. Visaggio, and M. Di Penta, "A heuristic-based approach for detecting sql-injection vulnerabilities in web applications," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, ser. SESS '10. New York, NY, USA: ACM, 2010, pp. 43–49. [Online]. Available: http://doi.acm.org/10.1145/1809100.1809107

[48] B. D. A. G. and M. Stampar, "automatic sql injection and database takeover tool," 2011, accessed April 2011. [Online]. Available: http://sqlmap.sourceforge.net/

[49] W. G. Halfond, S. Anand, and A. Orso, "Precise interface identification to improve testing and analysis of web applications," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, ser. ISSTA '09. New York, NY, USA: ACM, 2009, pp. 285–296. [Online]. Available: http://doi.acm.org/10.1145/1572272.1572305

[50] ITSecTeam, "Havij v1.15 advanced sql injection," Jun 2011, accessed April 2011. [Online]. Available: http://itsecteam.com/en/projects/project1.htm

[51] OWASP, "Jbrofuzz," July 2011, accessed July 2011. [Online]. Available: https://www.owasp.org/index.php/JBroFuzz

[52] "Sourcefire vrt certified rules - the official snort ruleset," accessed September 2011. [Online]. Available: http://www.snort.org/snort-rules/

[53] "Emerging threats," accessed September 2011. [Online]. Available: http://www.emergingthreats.net/

[54] "About barnyard2," accessed July 2011. [Online]. Available: http://www.securixlive.com/barnyard2/about.php

[55] "Http status codes," July 2011, accessed September 2011. [Online]. Available: http://www.google.com/support/webmasters/bin/answer.py?answer=40132

[56] R. Fielding *et al.*, "Hypertext transfer protocol – http/1.1," June 1999, accessed January 2011. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html

Chapter 8

Appendices

8.1 Appendix A

8.1.1 HTTP Status Codes

When a request is made to a web server, the server returns an HTTP status code in response to the request. This status code provides information about the status of the request. It gives information about the request and response. Some common status codes are [55]:

- 200 - the server successfully returned the page
- 404 - the requested page doesn't exist
- 503 - the server is temporarily unavailable

A complete list of HTTP status codes is below. For further information see the W3C page on HTTP status codes [56].

1xx (Provisional Response)

Status codes that indicate a provisional response and require the requestor to take action to continue. See Table 8.1

Code	Description
100 (Continue)	The requestor should continue with the request. The server returns this code to indicate that it has received the first part of a request and is waiting for the rest.
101 (Switching protocols)	The requestor has asked the server to switch protocols and the server is acknowledging that it will do so.

Table 8.1: HTTP Response 1xx (Provisional response)

2xx (Successful)

Status codes that indicate that the server successfully processed the request.(See Table8.2)

Code	Description
200 (Successful)	The server successfully processed the request. Generally, this means that the server provided the requested page.
201 (Created)	The request was successful and the server created a new resource.
202 (Accepted)	The server has accepted the request, but hasn't yet processed it.
203 (Non-authoritative information)	The server successfully processed the request, but is returning information that may be from another source.
204 (No content)	The server successfully processed the request, but isn't returning any content.
205 (Reset content)	The server successfully proccessed the request, but isn't returning any content. Unlike a 204 response, this response requires that the requestor reset the document view (for instance, clear a form for new input).
206 (Partial content)	The server successfully processed a partial GET request.

Table 8.2: HTTP Response 2xx (Successful)

3xx (Redirected)

Further action is needed to fulfill the request. Often, these status codes are used for redirection. (See Table 8.3).

Code	Description
300 (Multiple choices)	The server has several actions available based on the request. The server may choose an action based on the requestor (user agent) or the server may present a list so the requestor can choose an action.
301 (Moved permanently)	The requested page has been permanently moved to a new location. When the server returns this response (as a response to a GET or HEAD request), it automatically forwards the requestor to the new location.
302 (Moved temporarily)	The server is currently responding to the request with a page from a different location, but the requestor should continue to use the original location for future requests. This code is similar to a 301 in that for a GET or HEAD request, it automatically forwards the requestor to a different location.
303 (See other location)	The server returns this code when the requestor should make a separate GET request to a different location to retrieve the response. For all requests other than a HEAD request, the server automatically forwards to the other location.
304 (Not modified)	The requested page hasn't been modified since the last request. When the server returns this response, it doesn't return the contents of the page.

Table 8.3: HTTP Response 3xx (Redirected)

4xx (Request Error)

These status codes indicate that there was likely an error in the request which prevented the server from being able to process it. (See Table 8.4).

5xx (Server Error)

[!ht] These status codes indicate that the server had an internal error when trying to process the request. These errors tend to be with the server itself, not with the request. See Table 8.5

Code	Description
400 (Bad request)	The server didn't understand the syntax of the request.
401 (Not authorized)	The request requires authentication. The server might return this response for a page behind a login.
403 (Forbidden)	The server is refusing the request.
404 (Not found)	The server can't find the requested page. For instance, the server often returns this code if the request is for a page that doesn't exist on the server.
405 (Method not allowed)	The method specified in the request is not allowed.
406 (Not acceptable)	The requested page can't respond with the content characteristics requested.
407 (Proxy authentication required)	This status code is similar 401 (Not authorized); but specifies that the requestor has to authenticate using a proxy. When the server returns this response, it also indicates the proxy that the requestor should use.

Table 8.4: HTTP Response 4xx (Request error))

Code	Description
500 (Internal server error)	The server encountered an error and can't fulfill the request.
501 (Not implemented)	The server doesn't have the functionality to fulfill the request. For instance, the server might return this code when it doesn't recognize the request method.
502 (Bad gateway)	The server was acting as a gateway or proxy and received an invalid response from the upstream server.
503 (Service unavailable)	The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state.
504 (Gateway timeout)	The server was acting as a gateway or proxy and didn't receive a timely request from the upstream server.
505 (HTTP version not supported)	The server doesn't support the HTTP protocol version used in the request.

Table 8.5: HTTP Response 5xx (Server error)

8.2 Appendix B

8.2.1 Coding

DEB Command Line Networking Tools & DB Connection

```
                    //References used
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;
using System.IO.Compression;
```

```
using System.Security.Cryptography;

namespace WindowsFormsApplication1
{    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            ExecuteCommandAsync(textBox1.Text);
            ExecuteCommandSync(textBox1.Text);
        }
        public void ExecuteCommandSync(object command)
  try
    { System.Diagnostics.ProcessStartInfo procStartInfo = new
      System.Diagnostics.ProcessStartInfo("cmd", "/c " + command);
      procStartInfo.RedirectStandardOutput = true;
      procStartInfo.UseShellExecute = false;
      procStartInfo.CreateNoWindow = true;
      System.Diagnostics.Process proc = new System.Diagnostics.Process();
      proc.StartInfo = procStartInfo;
      proc.Start();

      string result = proc.StandardOutput.ReadToEnd();

           //Recording DEB Metadate

     textBox2.AppendText(Environment.NewLine + "Investigator Name:" +
     textBox7.Text + Environment.NewLine + "Case Details:" + textBox8.Text +
     Environment.NewLine + "Suspect Address:" + textBox9.Text +
     Environment.NewLine +     "Crime Referene Number:" + textBox10.Text +
     Environment.NewLine + "Description     of the Case:" + textBox11.Text +
     Environment.NewLine + "Time Stamps and Date:" +     textBox14.Text +
```

```
     Environment.NewLine + "Used Command:" + textBox1.Text +     Environment.NewLine
    + Environment.NewLine + result +     "----------------------------" +
    Environment.NewLine);

    }
         catch (Exception objException)
         {}
        }
        public void ExecuteCommandAsync(string command)
        { try
            {   Thread objThread = new Thread(new
ParameterizedThreadStart(ExecuteCommandSync));
                objThread.IsBackground = true;
                objThread.Priority = ThreadPriority.AboveNormal;
                objThread.Start(command);
            }
            catch (ThreadStartException objException)
            {}
            catch (ThreadAbortException objException)
            {}
            catch (Exception objException)
            {}
        }               // Creating MD5 Hash for the created DEB
        private static MD5 md5 = MD5.Create ( );
        private void button2_Click(object sender, EventArgs e)
        {
            this.folderBrowserDialog1.ShowNewFolderButton = false;
            this.folderBrowserDialog1.RootFolder =
            System.Environment.SpecialFolder.MyComputer;
            DialogResult result = this.folderBrowserDialog1.ShowDialog();

            string foldername = this.folderBrowserDialog1.SelectedPath;
            textBox3.Text = foldername + "\\" + textBox4.Text + ".txt";
            textBox6.Text = foldername + "\\" + textBox4.Text + ".zip";
                if (result == DialogResult.OK);
```

```
            string FILE_NAME = textBox3.Text  ;
            StreamWriter MyStream = null;
            string MyString = "Hello World";
            MyStream = File.CreateText(foldername + "\\" + textBox4.Text
+".txt");
            MyStream.Close();
            //MyStream.Write(MyString);
            if (System.IO.File.Exists(FILE_NAME) == true)
            {
                System.IO.StreamWriter objWriter = new
System.IO.StreamWriter(FILE_NAME);
                objWriter.Write(textBox2.Text);
                objWriter.Close();
                MessageBox.Show("A New Digital Evidence Bag Has Been Created");
            }
            else
            {
                MessageBox.Show("File Does Not Exist");
            }
 FileStream sourceFile = File.OpenRead(textBox3.Text);
 FileStream destFile = File.Create(foldername + "\\" + textBox4.Text + ".zip");
 GZipStream compStream = new GZipStream(destFile,
 CompressionMode.Compress);

            try
            {   int theByte = sourceFile.ReadByte();
                while (theByte != -1)
                {   compStream.WriteByte((byte)theByte);
                    theByte = sourceFile.ReadByte();
                }
            }

            finally              //Compressing the DEB in A Zip file
            {  compStream.Dispose();
```

```
            }
               using ( FileStream stream = File.OpenRead ( textBox6.Text ) )
   {
     byte [ ] checksum = md5.ComputeHash ( stream );
     textBox5.Text = ( BitConverter.ToString ( checksum ).Replace ( "-",
     string.Empty ));
   }
        }
        private void button3_Click(object sender, EventArgs e)
        {
        }
        private void textBox4_TextChanged(object sender, EventArgs e)
        {   if (textBox4.Text.Length > 0);
            {  button2.Enabled =true;
            }
        }
        private void Form1_Load(object sender, EventArgs e)
        {      button2.Enabled = false;
        }
        private void button3_Click_1(object sender, EventArgs e)
        {       textBox1.Text = "";
                textBox2.Text="";
        }
    }
}
```

HTTPFuzzer: Request Class Code (C++)

```
// Request.cpp: implementation of the Request class.
#include "stdafx.h"
#include "HTTPrequest.h"
#include "Request.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

// Construction/Destruction
Request::Request()
{}
Request::Request()
{}

// MemBufferCreate: Passed a MemBuffer structure, will allocate a memory buffer
// of MEM_BUFFER_SIZE. This buffer can then grow as needed.

void Request::MemBufferCreate(MemBuffer *b)
{
    b->size = MEM_BUFFER_SIZE;
    b->buffer =(unsigned        char *) malloc( b->size );
    b->position = b->buffer;
}

// MemBufferGrow:Double the size of the buffer that was passed to this function.

void Request::MemBufferGrow(MemBuffer *b)
{
```

```
    size_t sz;
    sz = b->position - b->buffer;
    b->size = b->size *2;
    b->buffer =(unsigned    char *) realloc(b->buffer,b->size);
    b->position = b->buffer + sz;                    // readjust current position
}

// MemBufferAddByte:Add a single byte to the memory buffer, grow if needed.

void Request::MemBufferAddByte(MemBuffer *b,unsigned char byt)
{
    if( (size_t)(b->position-b->buffer) >= b->size )
        MemBufferGrow(b);
    *(b->position++) = byt;
}

// MemBufferAddBuffer: Add a range of bytes to the memory buffer, grow if needed.

void Request::MemBufferAddBuffer(MemBuffer *b,
                  unsigned char *buffer, size_t size)
{
    while( ((size_t)(b->position-b->buffer)+size) >= b->size )
    MemBufferGrow(b);
    memcpy(b->position,buffer,size);
    b->position+=size;
}

// GetHostAddress: Resolve using DNS or similar(WINS,etc) the IP address for a
// domain name such as www.wdj.com.

DWORD Request::GetHostAddress(LPCSTR host)
{
    struct hostent *phe;
    char *p;
    phe = gethostbyname( host );
```

```
    if(phe==NULL)
        return 0;
    p = *phe->h_addr_list;
    return *((DWORD*)p);
}

// SendString: Send a string(null terminated) over the specified socket.

void Request::SendString(SOCKET sock,LPCSTR str)
{    send(sock,str,strlen(str),0);
}
// ValidHostChar: Return TRUE if the specified character is valid for a host name, i.e

BOOL Request::ValidHostChar(char ch)
{    return( isalpha(ch) || isdigit(ch)
     || ch=='-' || ch=='.' || ch==':' );
}
// ParseURL: Used to break apart a URL such as //
//http://www.localhost.com:80/TestPost.htm into protocol, port, host and request.

void Request::ParseURL(LPCSTR url,LPSTR protocol,int lprotocol,
        LPSTR host,int lhost,LPSTR request,int lrequest,int *port)
{    char *work,*ptr,*ptr2;
    *protocol = *host = *request = 0;
    *port=80;
    work = strdup(url);
    strupr(work);
    ptr = strchr(work,':');                        // find protocol if any
    if(ptr!=NULL)
    {   *(ptr++) = 0;
        lstrcpyn(protocol,work,lprotocol);
    }
    else
    {   lstrcpyn(protocol,"HTTP",lprotocol);
        ptr = work;
```

```
    }
    if( (*ptr=='/') && (*(ptr+1)=='/') )          // skip past opening /'s
        ptr+=2;
    ptr2 = ptr;                                   // find host
    while( ValidHostChar(*ptr2) && *ptr2 )
    ptr2++;
    *ptr2=0;
    lstrcpyn(host,ptr,lhost);
    lstrcpyn(request,url + (ptr2-work),lrequest);  // find the request
    ptr = strchr(host,':');                        // find the port number, if any
    if(ptr!=NULL)
    {   *ptr=0;
        *port = atoi(ptr+1);
    }
    free(work);
}

// SendHTTP:
// Main entry point for this code.
// url- The URL to GET/POST to/from.
// headerSend- Headers to be sent to the server.
//post- Data to be posted to the server, NULL if GET.
//postLength- Length of data to
//post.req - Contains the message and headerSend sent by the server.
// returns 1 on failure, 0 on success.

int Request::SendHTTP(LPCSTR url,LPCSTR headerReceive,BYTE *post,
        DWORD postLength,HTTPRequest *req)
{
    WSADATA             WsaData;
    SOCKADDR_IN         sin;
    SOCKET              sock;
    char                buffer[512];
    char                protocol[20],host[256],request[1024];
    int                 l,port,chars,err;
```

```
    MemBuffer           headersBuffer,messageBuffer;
    char                headerSend[1024];
    BOOL                done;

    ParseURL(url,protocol,sizeof(protocol),host,sizeof(host), // Parse the URL
    request,sizeof(request),&port);
    if(strcmp(protocol,"HTTP"))
        return 1;

    err = WSAStartup (0x0101, &WsaData);                    // Init Winsock
    if(err!=0)
        return 1;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    //if (socket == INVALID_SOCKET)
    if (sock == INVALID_SOCKET)
    return 1;

    sin.sin_family = AF_INET;                       //Connect to web sever
    sin.sin_port = htons( (unsigned short)port );
    sin.sin_addr.s_addr = GetHostAddress(host);

    if( connect (sock,(LPSOCKADDR)&sin, sizeof(SOCKADDR_IN) ) )
        {        return 1;
        }
//printf("\r\n\r\n <<SEND HTTP REQUEST:>> \r\n\r\n"); //Send request

    if( !*request )
        lstrcpyn(request,"/",sizeof(request));

    if( post == NULL )
       {
        SendString(sock,"GET ");
        strcpy(headerSend, "GET ");
       }
```

```
else
  { SendString(sock,"POST ");
    strcpy(headerSend, "POST ");
  }
SendString(sock,request);
    strcat(headerSend, request);

SendString(sock," HTTP/1.0\r\n");
    strcat(headerSend, " HTTP/1.0\r\n");

SendString(sock,"Accept: image/gif, image/x-xbitmap,"
    " image/jpeg, image/pjpeg, application/vnd.ms-excel,"
    " application/msword, application/vnd.ms-powerpoint,"
    " */*\r\n");
    strcat(headerSend, "Accept: image/gif, image/x-xbitmap,"
    " image/jpeg, image/pjpeg, application/vnd.ms-excel,"
    " application/msword, application/vnd.ms-powerpoint,"
    " */*\r\n");

SendString(sock,"Accept-Language: en-us\r\n");
    strcat(headerSend, "Accept-Language: en-us\r\n");

SendString(sock,"Accept-Encoding: gzip, deflate\r\n");
    strcat(headerSend, "Accept-Encoding: gzip, deflate\r\n");

SendString(sock,"User-Agent: Mozilla/4.0\r\n");
   strcat(headerSend, "User-Agent: Mozilla/4.0\r\n");

if(postLength)
{
    wsprintf(buffer,"Content-Length: \%ld\r\n",postLength);
    SendString(sock,buffer);
    strcat(headerSend, buffer);
}
     //SendString(sock,"Cookie: mycookie=blablabla\r\n");
```

```
        //     printf("Cookie: mycookie=blablabla\r\n");
    SendString(sock,"Host: ");
                strcat(headerSend, "Host: ");
    SendString(sock,host);
                strcat(headerSend, host);
    SendString(sock,"\r\n");
                strcat(headerSend, "\r\n");

    if( (headerReceive!=NULL) && *headerReceive )
        {
        SendString(sock,headerReceive);
        strcat(headerSend, headerReceive);
        }

   SendString(sock,"\r\n");  // Send a blank line to signal end of HTTP headerReceive
   strcat(headerSend, "\r\n");

    if( (post!=NULL) && postLength )
        {
        send(sock,(const char*)post,postLength,0);
                post[postLength] = '\0';
                strcat(headerSend, (const char*)post);
        }
// strcpy(req->headerSend, headerSend);
        req->headerSend = (char*) malloc( sizeof(char*) *
strlen(headerSend));
        strcpy(req->headerSend, (char*) headerSend );
//printf("\r\n\r\n <<RECEIVE   HTTP   REQUEST  : >> \r\n\r\n\r\n");
// First read HTTP headerReceive

    MemBufferCreate(&headersBuffer );
    chars = 0;
    done = FALSE;

    while(!done)
```

```
{
    l = recv(sock,buffer,1,0);
    if(l<0)
        done=TRUE;
    switch(*buffer)
    {  case '\r':
            break;
        case '\n':
        if(chars==0)
        done = TRUE;
        chars=0;
        break;
        default:
        chars++;
        break;
     }
    MemBufferAddByte(&headersBuffer,*buffer);
}

req->headerReceive        = (char*) headersBuffer.buffer;
*(headersBuffer.position) = 0;

MemBufferCreate(&messageBuffer);  // Now read the HTTP body
do
{
    l = recv(sock,buffer,sizeof(buffer)-1,0);
    if(l<0)
        break;
    *(buffer+l)=0;
    MemBufferAddBuffer(&messageBuffer, (unsigned char*)&buffer, l);
} while(l>0);
*messageBuffer.position = 0;
req->message = (char*) messageBuffer.buffer;
req->messageLength = (messageBuffer.position - messageBuffer.buffer);
```

```
    closesocket(sock);                  // Cleanup
    return 0;
}
// SendRequest
//void Request::SendRequest(bool IsPost, LPCSTR url, char *pszHeaderSend, char
// *pszHeaderReceive, char *pszMessage)

void Request::SendRequest(bool IsPost, LPCSTR url, CString &psHeaderSend,
CString &psHeaderReceive, CString &psMessage)
{   HTTPRequest req;
    int i,j,rtn; FILE *fp; LPSTR buffer;
    req.headerSend = NULL;
    req.headerReceive = NULL;
    req.message = NULL;

    //Read in arguments
      if(IsPost)
    {                /* POST */
    i       = psHeaderSend.GetLength();
    buffer  = (char*) malloc(i+1);
    strcpy(buffer, (LPCTSTR)psHeaderSend);
    rtn = SendHTTP(    url,"Content-Type: application/x-www-form-urlencoded\r\n",
    (unsigned char*)buffer,
    i,
    &req);
    free(buffer);
    }
    else           /* GET */
    rtn = SendHTTP(url,NULL,NULL,0,&req);
        if(!rtn)
                                        //Output message and/or headerSend
        {   psHeaderSend     = req.headerSend;
            psHeaderReceive  = req.headerReceive;
            psMessage        = req.message;
            free(req.headerSend);
```

```
            free(req.headerReceive);
            free(req.message);
        }
    else
        {
        //printf("\nFailed\n");
        MessageBox(0, "Retrieve Failed", "", 0);
        }
}
```

8.3 Appendix C

8.3.1 Multiple IDSs Configuration

When having multiple interfaces for sensors, and wanting to log in to different destinations. Multiple separate instances of both snort and barnyard2 are needed.

Snort

This is just the important part of Snort.conf file. For more details refer to Snort website.

```
# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/local/snort/lib/snort_dynamicpreprocessor/
# path to base preprocessor engine
dynamicengine /usr/local/snort/lib/snort_dynamicengine/libsf_engine.so
# path to dynamic rules libraries
dynamicdetection directory /usr/local/snort/lib/snort_dynamicrules

# unified2
vlan_event_types
output unified2: filename snort.u2, limit 128
....
# database
# output database: alert, <db_type>, user=<username> password=<password> test
dbname=<name> host=<hostname>
# output database: log, mysql, user=DATABASEUSERNAME password=YOURPASSWORD
dbname=DATABASENAME host=localhost
...
```

Barnyard2.conf

As Snort logs its alerts to a unified2 file, Barnyard read it and log into a database. This database can be Barnyard's database or Snorby or Base, or any other tool that can process the alerts in an easy way.

```
# database: log to a variety of databases

  output database: log, mysql, user=USER_ACCESS_THE_DB
password=DB_PASS dbname=DB_NAME host=localhost
#   output database: alert, postgresql, user=snort dbname=snort
#   output database: log, odbc, user=snort dbname=snort
#   output database: log, mssql, dbname=snort user=snort password=test
#   output database: log, oracle, dbname=snort user=snort password=test
```

We put Snorby in the database name field as we used Snorby for viewing alerts and analysing them.

Snorby

snorby_config.yml

Since Barnyard2 was set to read alerts from Snort unified file and log them to Snorby database, the Snorby configuration file should look similar to the following:

```
development:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf

test:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf

production:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf
```

Snorby Database.yml

Whereas Snorby database configuration file should like similar to the following:

```
 snorby: &snorby
  adapter: mysql
  username: root
  password: "PASSforRoottoAccessTheDB" # Example: password: "s3cr3tsauce"
  host: localhost

development:
  database: snorby
  <<: *snorby

test:
  database: snorby
  <<: *snorby

production:
  database: snorby
  <<: *snorby
```