-

Automatic XSS detection and Snort signatures/ ACLs generation by the means of a cloud-based honeypot system

Benoit Jacob

08009764

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the degree of
Msc Advanced Networking

Supervisor Professor William J. Buchanan

Second marker Mr Alistair Lawson

School of computing

December 2011

Authorship Declaration

I, Benoit Jacob, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: 12 December 2011

Matriculation no: 08009764

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

It is no secret that nowadays cloud computing is becoming popular with large companies, mainly because they can share valuable resources in a cost effective way. This is just the beginning of the cloud computing, an independent research firm “Forrester Research” expects the global cloud computing market to grow from $40.7 billion in 2011 to more than $241 billion in 2020 (Ried, et al., April 2011). This increasing migration towards cloud computing is resulting in an escalation of security threats, which is becoming a major issue. The cloud computing is even considered to be a “security nightmare», according to John Chambers, Cisco CEO (McMillan, 2009).

The migration of networks and servers over the cloud means that hacking techniques are now aimed at cloud-based servers. According to the Web Hacking Incident Database (WHID, 2011) the Cross Site Scripting (XSS) attacks are currently the second most implemented attacks and they are associated with a 12.58% of the overall attacks on the web. Moreover, XSS attacks can easily pass through Intrusion Detection Systems (IDS), such as Snort, without raising an alert as these systems lack detection for attacks which use hex-encoded values (Mookhey, et al., 2010).

This thesis aimed to detect XSS attacks sent to a cloud-based Webserver by using a honeypot that simulates a fake Webserver onto the cloud computing of Edinburgh Napier University. The honeypot is able to log any interaction with the webserver simulated. By developing specifics Bash scripts, this thesis showed the possibility to detect XSS attacks, through analysis of the honeypot’s log file, leading to the generation of ACLs/Snort signatures. ACLs will block the IP addresses source of the attacks and the Snort signatures will block similar packets to enter the network again.

All experiments were done through the cloud of Edinburgh Napier University to ensure that the results are the more realistic possible. The result shows that out of a random set of 50 XSS attacks, the Bash scripts implemented in the honeypot generated 26 Snort signatures. These signatures were implemented into Snort which was then able to detect 64% of the same set of XSS attacks. This is 4% above the acceptable level of True Positive alerts, which should be at least 60% of the total alerts raised (Timm, 2010). Finally, background traffic and XSS attacks were injected into the honeypot at increasing speed, to measure the efficiency of the honeypot in detecting attacks within high loads of traffic. Despite an increasing latency in correlation with the network load speed, HoneyD was able to log/detect the same XSS attacks, as seen previously. However, at 2mbps, the honeypot generated a “segmentation fault” error due to insufficient memory that the CPU could not physically address. The 2mbps load speed was identified to be the breaking point of the honeypot and an unstable interval was established between 1.5-2mbps.

The conclusion drawn in this thesis is that HoneyD coupled with Bash scripts, are successfully able to automatically detect XSS attacks and trigger the generation of ACLs/Snort signatures. Further work could be realised by improving the detecting engine.

Table of Content

List of Figures

List of Tables

Acknowledgements

This dissertation would not have been possible unless my supervisor Professor William J. Buchanan. Thank you for giving me access to Edinburgh Napier cloud computing, for helping me refine my ideas, giving me valuable advices and answering to the numerous amounts of emails sent during this project.

Moreover, I would like to thank Mr Alistair Lawson, my second marker, for the time spent reviewing this thesis and taking part to the viva voce.

Finally and not least, I would like to thank Ms Selina Kubo for the proof reading and the moral support throughout the whole project.

Chapter 1. Introduction

1.1 Context

With the increasing migration towards cloud computing, it was obvious that working on a subject related to the cloud would be an interesting topic. After reading about this technology, the main issues that were repeatedly highlighted were the security issues. The security issues concerning data privacy and protection are major factors that deter some companies from moving to the cloud. Based on these findings, the choice was taken to work in the area of Cloud Computing Security. Luckily, Edinburgh Napier University started to migrate to the Cloud Computing at the beginning of the year 2011 and was proposing to its students to devise their network laboratory work through instances emulated by the cloud. Therefore, after agreement with Pr. Bill Buchanan, it was possible for this project to be implemented within the University cloud. After further research, a way to detect attacks and intruders was found by setting up a fake server, known as honeypot, able to lure them. The implementation of a honeypot within the cloud computing seemed to be interesting project and a deeper investigation was carried out through the three technologies involved in this project, Honeypot systems, Intrusion Detection System (IDS) and Cloud Computing. These technologies are analysed and compared in the next chapter, in order to provide a good understanding to the reader.

1.2 Background

It is no secret that cloud computing is becoming popular with large companies nowadays, mainly because they share valuable resources in a cost effective way. This is just the beginning of the cloud computing, an independent research firm “Forrester Research” expects the global cloud computing market to grow from $40.7 billion in 2011 to more than $241 billion in 2020 (Ried, et al., April 2011). This increasing migration towards cloud computing results in an escalating security threat, which is becoming a major issue. Many studies (McDonald, 2011) (Wolf, 2010) show that there are major security issues to take in consideration before moving onto the cloud. The cloud has even been called “security nightmare” by John Chambers, Cisco CEO (McMillan, 2009).

On the other hand, there is a wide repertoire of hacking techniques used by pirates to gather sensitive data and stolen credentials from web servers. According to the Web Hacking Incident Database (WHID, 2011), most popular attacks concern SQL injections, Cross Site Scripting (XSS) and Denial of Service (DoS). The SQL injections and DoS attacks are often making the headlines of the news but the poorly-known XSS attacks are kept in the shadows. However, these XSS attacks, according to the WHID, are currently the second most

used attacks and associated with 12.58% of the overall attacks on the web. Moreover, XSS attacks can easily pass through Intrusion Detection System (IDS), such as Snort, without being detected as it lacks the ability to detect XSS attacks using hex-encoded values (Mookhey, et al., 2010).

The migration of networks and servers over the cloud involves that XSS attacks are, or will be soon, aimed at cloud-based Web servers. This report aims to investigate this issue by implementing a cloud-based honeypot that simulates a fake Web server able to detect XSS attacks. The XSS attacks detected are used to generate Access Control Lists (ACLs) and IDS Signatures that will be implemented in the router and IDS to block future attacks.

1.3 Aims and Objectives

The aim of this project is to automatically detect XSS attacks and generate Snort signatures to block similar future attacks. To do so, a honeypot, that simulates a Webserver and handle the XSS attacks, will be implemented on the cloud computing of Edinburgh Napier University. This project has a scientific interest as it should complete the literature in the area of Intrusion Detection Signatures by providing an automatic way to detect attacks and generate Snort signatures. In order to achieve this project, the following objectives must be achieved:

1. Review and investigate the existing literature about security issues related to the cloud computing. In addition, critically evaluate the previous work in generating IDS signatures and compare different honeypot systems. Finally, analyse a specific attacking method and ways to detect it.
2. Design some scripts to generate IDS signatures every time that an attack is detected. Design some cloud-based experiments that will show the effectiveness of the scripts created in detecting attacks and generating IDS signatures.
3. Conduct the final evaluation based on the results collected from the experiments. This evaluation should present the effectiveness of the scripts created.

1.4 Thesis layout

The remainder of this report is organised as follows:

- **Chapter 2 – Technologies review:** The first chapter provides a basic understanding for the reader about the four technologies used throughout this project. This understanding is required in the next chapter, Literature Review, which investigates further into contrasting these technologies and their issues.
- **Chapter 3 – Literature review:** This chapter critically reviews the literature in the area of cloud computing security issues, signature generation using honeypots, legal issues concerning honeypot's implementation and cross site scripting attacks. In

addition, a comparison of multiple honeypot system is carried out to determine the honeypot the most suited for this study.

- **Chapter 4 – Design:** This chapter presents a design of the scripts required to generate ACLs and Snort signatures. This script design provides a good understanding of the functions required and help in making a decision towards an appropriate programming language. The methodology used in the experiments is introduced and expected results are projected.
- **Chapter 5 – Implementation:** This chapter deploys and configure the instances required for the experiments. The experiments designed in the previous chapter are carried out and the resulting data is collected.
- **Chapter 6 – Evaluation:** This chapter evaluates the effectiveness of the prototype created, by analysing the data collected during the experiments. In addition, the results of the evaluation are compared to the results expected in the Design.
- **Chapter 7 – Conclusion:** This chapter provides a summary of the main findings and compares them with the initial objectives. A critical evaluation of the project is carried out and advice for future work is suggested.

Chapter 2. Technologies Review

2.1 Introduction

This project aims to detect attacks targeting a cloud-based server. In order to detect these attacks, a honeypot is used to simulate a fake server and log any interaction with a malicious user. This chapter provides a good background and understanding about the three main technologies used all along this paper: Honeypot, Cloud Computing, IDS and an up to date overview of the most popular attacks used by pirates to gather sensitive data and stolen credentials.

2.2 Honeypot systems

Honeypots are not defensive security systems as Intrusion Detection Systems (IDS) and Firewalls as they do not tend to protect the network but lean towards attracting the intruder. According to L. Spitzner, a honeypot is *"an information system resource whose value lies in unauthorized of illicit use of that resource"* (Spitzner, 2003). Honeypot systems are fake information (server or client) able to run multiple services, such as FTP, SQL, Web, SSH, etc. These services are configured using weak security mechanisms, making them highly interesting for an intruder in search for an easy target. Logging mechanisms are loaded with monitoring and tracking tools, which make them able to capture data resulting from an illicit access such as attacks techniques, events and intrusions. When more than a single honeypot system is used in a network, they form a network of honeypot called *Honeynet*. The following Figure 1 represents an overview of the taxonomy of a honeypot system.

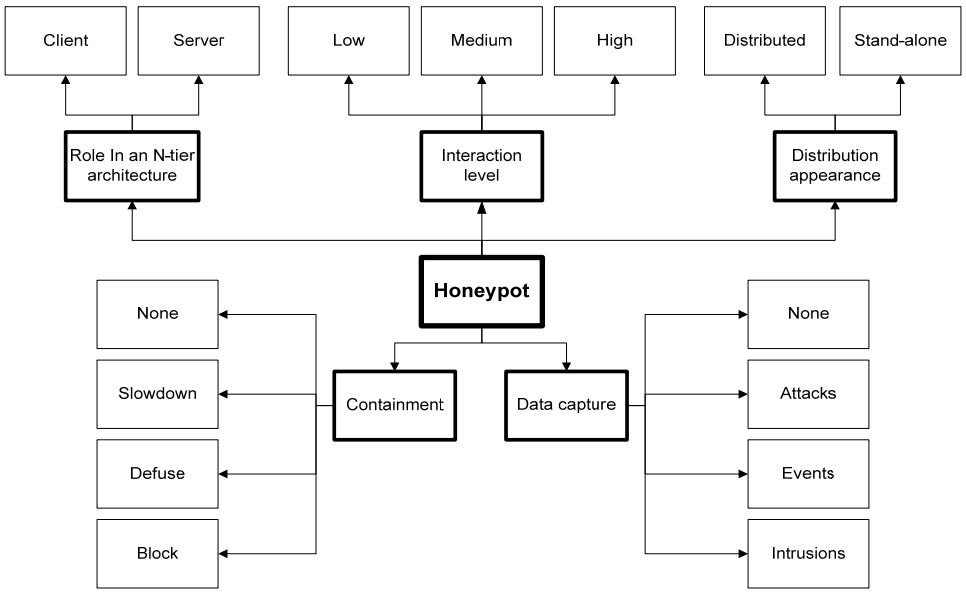


Figure 1 - Honeypot taxonomy based on (Christian Seifert, 2006)

2.2.1 Level of interactivity

A honeypot can offer three levels of interaction from low to the high or in between. The higher the interaction is, the more liberty of movement will be provided to an attacker and the more data will be collected about that attack. However, a high level of interaction induces a high risk of potential damages (Spitzner, 2002). The following description provides an evaluation of the three types of honeypots with their advantages and inconvenient.

2.2.1.1 Low interaction honeypot

A low interaction honeypot provides a limited communication between the honeypot and the attacker, seen Figure 2. Low interaction honeypots offer the possibility to emulate network services on preconfigured port, such as FTP, SQL, Web, SSH, etc. However, these services are restricted to only reply to basic queries (such as a ping or a connection attempt).

Advantages: the ease of installation and configuration, as well as a low risk of potential damage by an attacker.

Inconvenient: the collections of information are limited to the date/time of the connection, the IP source and the source/destination ports. Therefore, they are mostly used to detect unauthorized connections in the network.

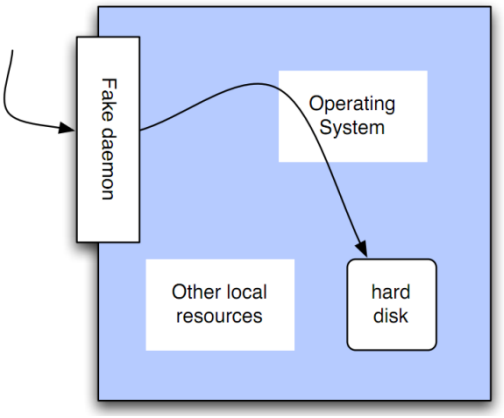


Figure 2 - Low interactivity (Koot, 2007)

2.2.1.2 Medium interaction

A medium interaction honeypot collects more information about attacks than a low interaction, by giving the ability to the attacker to interact a bit more with the honeypot, seen Figure 3. This medium interaction takes the attacker a step further and the honeypot is now able to reply to specific commands, by using preconfigured messages. The services emulated are programmed in some templates, which provide the list of the authorized commands with their specific replies. In the case of a command being received by the honeypot that does not match any authorized commands, a message “command unknown” is generated.

Advantages: the data collected is more advantageous than a low interaction honeypot, due to a higher interaction. In terms of security, a low risk of potential intrusion is expected as the honeypot only answer to preconfigured commands.

Inconvenient: the attacker generally, quickly discovers that the system does not behave as it should.

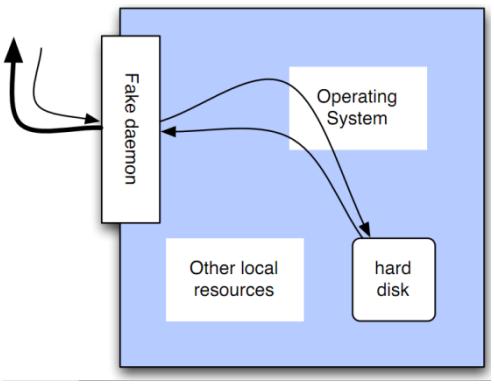


Figure 3 - Medium interactivity (Koot, 2007)

2.2.1.3 High interaction

High interaction honeypots operate on real systems and provide the attacker the possibility to break into the operating system and take control over it, represented Figure 4. There are no more limited commands like in the medium interaction honeypot, the attacker is now able to fully interact with a the whole operating system and its applications. Multiple sensors are installed on the honeypot and collect all the data used by the attacker, such as toolkits uploaded, keystrokes typed in and network data flow (Awad Johny, 2009).

Advantages: all the interaction of the hacker and all the files uploaded are captured by the honeypot. It provides a vast amount of information for the researcher about unknown attack and previously known attack.

Inconvenient: the setup of the honeypot is time consuming: Firstly, it must be customized and configured to the only applications needed for the experiment. Secondly, the vast amount of data gathered must be scrutinized by researchers in order to determine the aim of the attacker. Moreover, this type of honeypot must be always behind a firewall and constantly monitored in order to reduce the risk of a nasty attacker, which might compromise the honeypot and use it to spread attacks to other systems (Awad Johny, 2009).

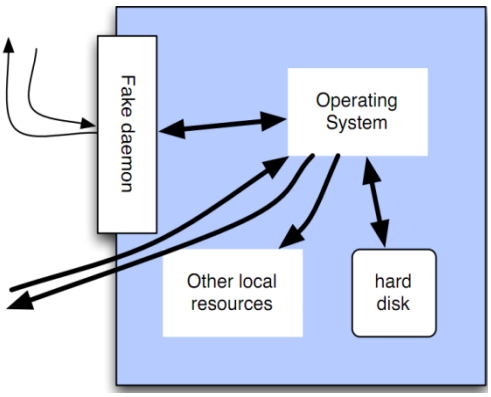


Figure 4 - High interactivity (Koot, 2007)

2.2.2 Purpose of Deployment

The deployment of a honeypots can be categorised into two categories, research honeypots and production honeypots, represented Figure 5:

Research honeypots are deployed at in a non-secure zone and are used primarily for a medium or high interaction. This kind of honeypot can be deployed in the outside of the network or in the Demilitarize Zone (DMZ). They are used to collect the maximum of information about new attacks, such as new toolkits used by attackers and commands used to break into the system. All this information is analysed by researchers in order to create new antivirus or IDS signatures that aim to improve the system defence against future attacks. However, research honeypots need to be under constant supervision in reason of their dangerous location on the outside of the network and their high interactivity, which make them an easy target for attackers.

Production honeypots are deployed inside secure environments with the aim to attract intruders away from critical systems. Usually, a low interaction honeypot is used in the production network, as we do not want an attacker to take over the system. These kinds of honeypots are also used to discover if an attacker made his way into the production network and fooled the security equipment. In addition, this honeypot gives the possibility

to detect insiders trying to hack into the production network or Worms/Trojans spreading automatically.

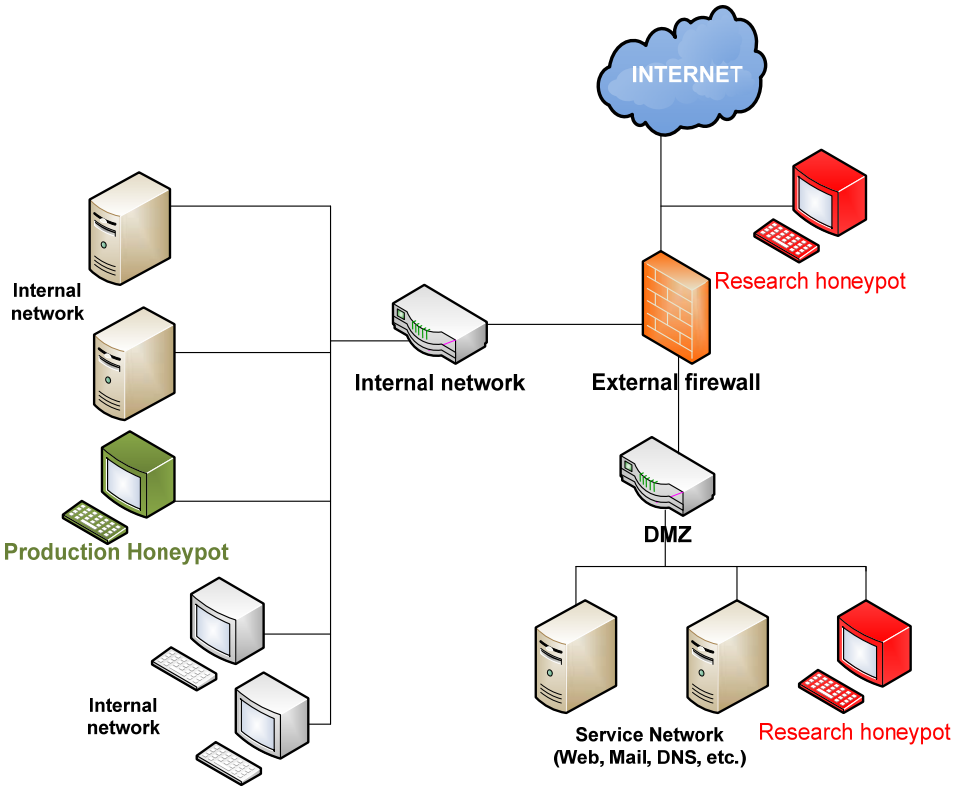


Figure 5 - Honeypot deployment: Research and Production

2.3 Intrusion Detection Systems

Intrusion Detection System is a process or device that analyses system and network activity for abnormal behaviours. The way an Intrusion Detection System (IDS) detects malicious packets widely vary according to the type of implementation (host-based or network-based) and detection techniques (signature-based or anomaly-based) used by the IDS. This section gives an overview of the advantages and inconvenient of these systems.

2.3.1 Types of IDS

2.3.1.1 Network-based

A Network-based IDS identifies intrusion by examining network traffic, as seen Figure 6. The network IDS often contains a list of known attack signatures that are compared to the live network traffic. Whenever a packet matches a signature, an attack attempt alert is raised. There are two different ways to implement a network-based IDS: Inline or Outline.

An **inline implementation** acts like a gateway and ensures that all the traffic goes directly into the IDS before reaching/leaving the production network. The benefit of this implementation is the effectiveness to quickly block potential attacks. However, it comes at a price, as the IDS sits in the middle of the network and filters the whole network traffic. The

risk is that an overload of the IDS could happen during peak times and this one could start dropping genuine data packets.

An **outline implementation** means that the IDS does not receive directly the data packets. The IDS is plugged into a switch configured to mirror the network traffic to the IDS. Therefore, all the network traffic is copied and sent to the IDS for analysis. The benefit of an outline implementation is that the IDS cannot slow down the network traffic because the traffic is copied and sent to it. The inconvenient is a slow reaction from the IDS when malicious packets are discovered. The reason is that the packets generally reach the IDS as the same time as the destination host, which make them hard to block.

2.3.1.2 Host-based IDS

A Host-based IDS is generally used to analyse all the network traffic reaching or leaving a computer that hosts a network service and are sensible to attacks, such as Webserver, Database or FTP server, see Figure 7. This host-based analysis acts like a firewall that inspects every data packet interacting the host with a unique advantage of being able to inspect encoded traffic. This advantage is possible because the data packets are decoded when they reach the destination host, therefore the host-IDS is able to analyse them. The main drawback of a Host-based implementation is the consumption of physical resources needed by the IDS to inspect the traffic, which could result in using most of the host resources at peak times.

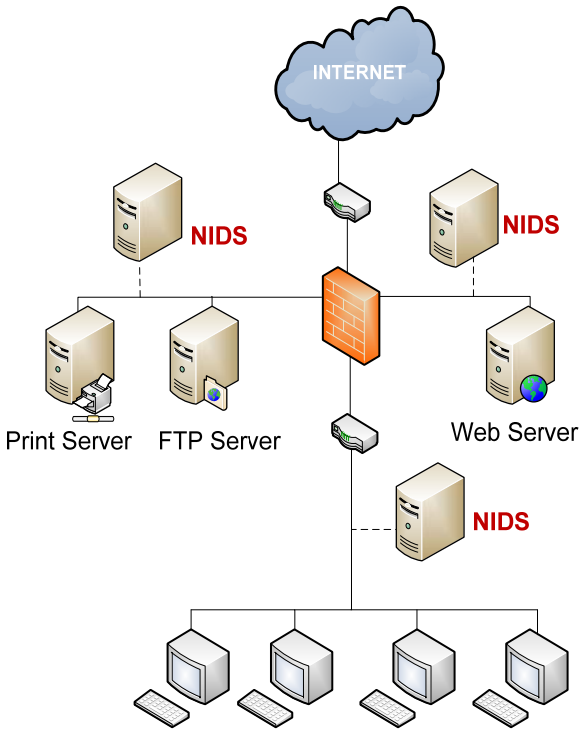


Figure 6 - Network-based IDS

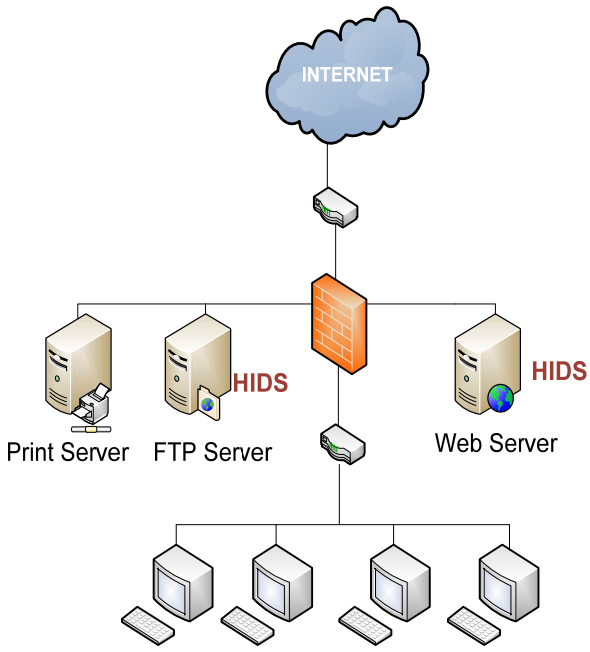


Figure 7 - Host-based IDS

2.3.2 Detection Techniques

There are two detection techniques that can be used by IDS to discover malicious data packets: the first one based on data packets signature and the second one based on anomaly detection. Each detection technique has its own benefits and weaknesses, compared in the following section.

2.3.2.1 Signature-based Intrusion Detection

Signature-based detection is the most widely type of detection used by IDS nowadays. The IDS uses a local database with multiple packet signatures known as being malicious. Each data packet going through the IDS is compared to a list of known malicious patterns. Whenever a positive match is found, it means that a malicious packet has been detected. This type of detection is similar to an antivirus, which compare files signatures to a database of malicious known virus signatures.

The advantages of a signature-based IDS are the ease of configuration and the low rate of false positive (Allen, 2004). However, the main drawback is similar to antivirus software: they cannot detect unknown malicious patterns. This means that firstly the IDS needs to be updated regularly and secondly that the IDS will always be one-step behind the attacker (Joho, 2004). Another type of IDS, anomaly-based Intrusion Detection can be used to counteract these weaknesses.

2.3.2.2 Anomaly-based Intrusion Detection

Anomaly-based Intrusion Detection inspects the data packets and compares them with the self-learned network patterns specific to the network. To generate these network patterns, the IDS go through a training phase where its only objective is to listen to the network data flow and translate it into Metadata. The patterns generated give an enormous quantity of information such as network addresses, flags, used ports, timeouts, etc. These results are considered by the IDS as a normal network activity. After this training period, the anomaly-based IDS is ready to be implemented and can compare the network traffic to the pattern generated during the training phase. Alerts are raised when the IDS detect unusual network activity, such as detecting new services which should not be active, or detecting users' access to new services that they never accessed before.

The main strength of anomaly-based IDS is the possibility to detect unknown malicious network traffic, which is not possible using a signature-based IDS. However, there are important drawbacks: Firstly, if any attacks occur during the training period, they will be considered as normal traffic and remain undetected in the future. Secondly, there are concerned towards the scalability of this system because if the network administrator decides to give new service accessibility to specifics users, the IDS will raise False Positive alerts. This leads to the main drawback of this system, which is the high number of False Positive alerts that are generated if the anomaly-based IDS is not perfectly tuned up.

2.3.3 Acceptable levels of false alarms

The two previous paragraphs describe how IDSs work: they analyse network traffic and raise alarms whenever malicious data packets are discovered. However, these alarms are not always right and unjustified alarms can be triggered. There are four kinds of alarms, they are based on the success to detect attacks and raise alarms. The Table 1 represented below illustrates the meanings of these alarms.

Table 1 - Confusion Matrix

		Alarm raised	
		Yes	No
Attack attempt	Yes	True Positive	False Negative
	No	False Positive	True Negative

In theory, a perfectly tuned IDS is able to rise only True Positive alerts, which means that a security related event has happened and an alert has been raised accordingly. However, in real conditions with a high amount of network traffic, many False Positive alerts are generally raised up by IDS. False Positive alerts mean that an alert was raised without a true security related event. Therefore, the main question is: What is an acceptable level of True Positive alerts? This question is debatable and every network engineers have their own point of views about it. However, to give a vague idea, an answer has been found on the Symantec Website (Timm, 2010): “an acceptable level of True Positive alerts should be at least 60% of the total alerts raised”. This True Positive Rate is calculated using the following formula:

$$\mathrm{TPR} = \frac{TP}{TP + FN} * 100$$

With: TP = An attack has occurred and an alarm has been raised,

FN = An attack has occurred but no alarm was raised.

In addition, the False Positive ratio can be calculated with a similar formula. It shows the proportion of instances, which were not malicious but raised an alarm (false alert):

$$\mathrm{FPR} = \frac{FP}{FP + TN} * 100$$

With: FP = An alarm has been raised but no attack occurred (False alert)

TN = The number of correct decisions on benign traffic.

A perfectly tuned IDS should have a rate of TPR = 100% and FPR = 0%, which means that every attacks have raised alerts and that benign traffic has never raised any alert. However, this case is extremely rare in live environment.

2.4 Cloud computing

The emergence of the Cloud Computing has revolution the IT industry by reducing the hardware cost while improving its scalability. Edinburgh Napier University has been using the cloud since the beginning of the year 2011. The cloud computing gives the possibility to any students to access their Virtual Machines used in the Networking labs, from any devices connected to internet, anywhere and at any time. An article written by Petter Mell for The National Institute of Standards and Technology (Peter Mell, 2011) gives a good definition of the cloud computing through five essential characteristics, three service models and four deployment models, such as following.

2.4.1 Essential Characteristics

- **On-demand self-service:** individuals can manage computing resources without needing anyone's help;
- **Ubiquitous network access:** The platform can be accessed through standard Internet-enabled devices;
- **Rapid elasticity:** resources are scalable and can be quickly adjusted when it suits (raised or lowered);
- **Resource pooling:** processing and storage are shared across a common infrastructure;
- **Pay per use:** consumers usage are tracked in order to determine charged fees based on a combination of computing power, bandwidth use and storage space;

2.4.2 Deployment Models

- **Private cloud** (internal clouds): The cloud infrastructure is owned and used by a single organisation. However, a third party service provider on or off-premise can manage the hosting and outsourced operation.
- **Community cloud**: The cloud infrastructure is shared by several related organisations. For example, all the universities of a country can use a community cloud in order to share their resources between each other's.
- **Public cloud** (external clouds): The cloud infrastructure is owned by a third party service provider that is selling cloud services to multiple organisations.
- **Hybrid cloud**: The cloud infrastructure is a composition of two or more clouds infrastructure (private, community, public). The advantage of this technology is to provide dedicated resources (private cloud) and increased scalability (public cloud) when it suits.

2.4.3 Service Models

- **Software as a Service: SaaS** delivers to the costumers business applications hosted on the cloud by a third party organisation. Customers can access these applications through an internet enable device using a thin client such as a web browser. This

model eliminates the need to install and run the applications because the applications are not installed directly on the customer's own computer, which induce an easier maintenance and support. An example of SaaS provider is *Oracle CRM On Demand* that provides both multi-tenant (all resources are shared by customers) and single-tenant (customers have owned resources) options.

- **Platform as a Service: PaaS** provides all the software needed by developer to build, deploy and manage SaaS applications. However, the service provider defines the supported tools and programming languages. Moreover, the costumer does not control the underlying cloud infrastructure such as network, servers or operating systems. An example of SaaS provider is *Oracle PaaS Platform* that provides to the costumers the ability to build their own public clouds.
- **Infrastructure as a Service: IaaS** provides a standardized virtual server with elastic resources. The consumer is able to deploy operating systems, control and manage computing resources. However, the customer is not able to manage the underlying cloud infrastructure. An example of IaaS is *Amazon Elastic Compute Cloud (EC2)* and *Simple Storage Service (S3)*.

2.4.4 Virtualisation

Today's virtualisation technology is use by the cloud computing in order to deliver on-demand cloud services, which is the key technology of the cloud computing success (Perilli, 2009). However, the concept of virtualization is not as recent as the cloud computing. Actually, it is in the mid-1960s that IBM created for the first time some virtual machines implemented within an *IBM 7044 (M44).* At the time, IBM's virtual machines were identical "copies" of the underlying hardware and each instance could run its own operating system (Singh, 2004). It was used to reduce the highly expensive hardware acquisition, which was at the time enormous pieces, and improve the ability for the users to work simultaneously. In recent days, virtualization technology has been able to separate the operating system and application from the hardware. Moreover, the hardware got cheaper and the trend for testing applications in a sand box has increased. The main reasons to use virtualisation are described by a consultant for Citrix (Bogobowicz, 2011), such as following:

- **Hardware independence:** The same image file of the computer/server can be used across multiples types of hardware.
- **More efficient resource use:** A single physical server can be split into multiple virtual machines that are sharing the resources of the physical server. Moreover, the virtual machines can be upgraded with more resources (CPU, memory, hard drive…) without any downtime.
- **Fast deployment and snapshotting:** The deployment of new virtual machines takes a couple of seconds and snapshots can be taken at any given time in order to allow rollbacks if troubles happen. It is a great benefice for software testing and evaluation.

- **Backup and transfer:** Entire operating systems can be backed up on-site or be transferred off-site, allowing a quick recovering in case of disasters.

The management and execution of multiple operating systems is done, most of the time, through a hypervisor also called Virtual Machine Manager (VMM). Hypervisors are directly installed onto the server hardware whose provide multiple operating systems. Hypervisors are classified into two types named Type 1 and Type 2, see Figure 8 . *Virtualization Review*, an independent web guide to virtualisation, states that there is no formal standards-based definition for both types but the distinction has to do with whether an underlying operating system is present (Vanover, 2009). Therefore, hypervisor type 1 level runs directly onto the host physical hardware to control and manage guest operating systems, while type 2 runs in an operating system installed onto the physical hardware (2nd level) to control and manage third level operating system. Hypervisor type 1 refers to VMware ESXi and Citrix XenServer, while Hypervisor type 2 refers to Sun VirtualBox, VMware Server and Microsoft Virtual PC.

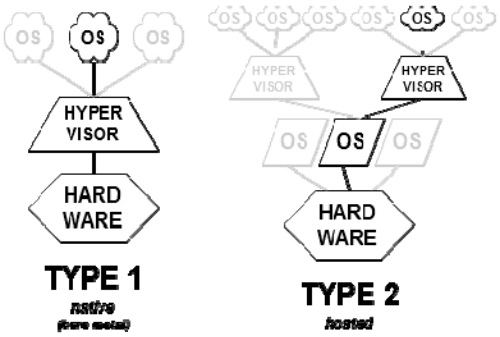


Figure 8 - Hypervisor Types (Scsami, 2011)

2.4.5 Cloud Hypervisors

2.4.5.1 General Overview

A cloud hypervisor is a mainframe operating system sitting at the lowest of the hardware environment, known as bare metal. The hypervisor is able to support many different operating environments and gives a very practical way of getting virtualization started quickly and efficiently. The main hypervisors available on the market nowadays are:

- VMware ESX (License proprietary)
- VMware ESXi (License proprietary)
- Microsoft Hyper-V with Windows Server 2008 (License proprietary)
- Microsoft Hyper-V with Windows Server Core (License proprietary)
- Citrix XenServer v4.1 (Open source)
- OpenNebula (Open source)
- Eucalyptus (Open source)

The major difference between all these hypervisors is that some are free while others are cost-effective. The battle between open source and license proprietary hypervisors involves the following advantages and disadvantages: The license proprietary hypervisor is expensive and un-flexible but benefit of a good support and an easy implementation, while the open source hypervisor is free and more flexible but harder to implement and to get the documentation.

This paper presents a project for Edinburgh Napier University. The University recently implemented VMware ESXi in the laboratory rooms, which will be used in this project. Therefore, the next section aims to only describe the features of the cloud hypervisor VMware ESXi.

2.4.5.2 VMware ESXi

VMware ESXi hypervisor is a production-proven virtualization layer that abstracts the physical server resources (processor, memory, storage…) into multiple virtual machines, see Figure 9. VMware ESXi is thinner than his big brother VMware ESX (less than 100Mb instead of 2 GB footprint) and offers all the same functionality. VMware ESXi is used in Napier University in order to reduce hardware and operation costs by sharing physical resources across virtual environments. VMware ESXi is similar to an operating system, it is installs directly on the server hardware, called "bare metal", in order to give a complete control of the server resources. Once installed, VMware ESXi allows the creation of multiple virtual environments with their own specific resources. Virtual environments are completely isolated from each other by the virtualization layer. In consequence when a misconfiguration occurs in a Virtual Machine it does not affect other virtual environments.

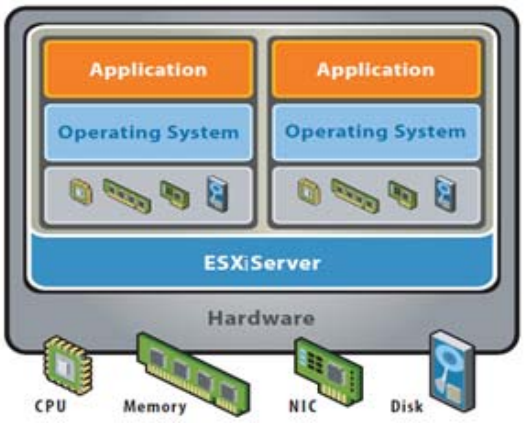


Figure 9 - VMware ESXi Server (Venezia, 2008)

The Vendor (VMware, 2011) claims the following advantages when migrating to an ESXi server:

- Run multiple applications on a single server in order to increase the hardware utilization through virtualization while reducing operating costs.
- Run a greener datacentre, ESXi can reduce the energy costs by 80%.
- Back up and recover applications more easily by taking snapshot of live machines and restoring them within seconds in case of errors.
- Virtual machines can be easily accesses through a thin clients or PCs.
- Virtualize even business-critical applications with a cost effective virtualization-based solution.

These advantages represent for Edinburgh Napier University, the possibility for professors to create practical labs within multiple virtual environments. These virtual environments are assigned to specific students in order to allow them to work with it from anywhere, at any

time and using any device internet enable. Additionally, students are able to have a risk free environment to work on, because if a misconfiguration occurs they can rollback or re-create a new VM within seconds.

2.5 Hacking techniques

Today, there is a wide repertoire of hacking techniques used by pirates to gather sensitive data and stolen credentials from servers. According to the Web Hacking Incident Database (WHID, 2011), most popular attacks concern SQL injections, Cross Site Scripting (XSS) and Denial of Service. This section aims into giving on overview of each one of these attacks.

2.5.1 SQL Injection

SQL injection the most common application layer attack technique used to steal data from organizations. This type of attack aims websites with incorrect coding that allow hackers to inject SQL commands in order to retrieve data held in the website's database. To achieve this attack, a hacker first needs a web page with features, such as login fields, forms, shopping carts or dynamic content. These features are widely used on web sites nowadays and too often not coded appropriately. Once a weak web site has been found, a hacker proceeds to a SQL injection, which aims to send SQL commands through a web application for execution by the backend database. A successful attack allows a hacker to communicate with the database in order to retrieve and/or modify sensible credentials saved on the database.

A famous example of a SQL injection is explained in a study realised by (Adam Kiezun, 2011) realised through a login form by submitting "' OR 1=1 –-" within the input login field and keeping the password field empty, which result in:

```
login='' OR 1=1 -- AND pass=''
```

The injection of this input transforms the entire WHERE clause into a tautology. The tautology means a universal unconditioned truth, always valid (William G.J. Halfond, 2006). This causes the application to conclude that the user authentication is successful and therefore, the user is granted access without any real credentials.

2.5.2 Cross Site Scripting (XSS)

Cross Site Scripting also known as XSS is one of the most common application layer hacking techniques. According to the WHID, 12.58% of the overall attacks on the web are associated with XSS (WHID, 2011). Additionally, XSS is also the origin of many other attacks such as Information Disclosures, Content Spoofing and Stolen Credentials.

Nowadays, websites are growing exponentially fast and include more complex dynamic features that deliver new services to their customers. The drawback of these dynamic websites is that they do not have a complete control over the output's interpretation by the

customer's browser. In that case, malicious content can be introduced into a dynamic web page without the client or website knowing about it.

XSS attacks work by injecting script tags in URLs and enticing unsuspecting users to click on them, ensuring that the malicious code, such as Flash, HTML, VBScript or get executed onto the user's machine. To do so, the malicious script is inserted through common inputs, such as URL parameter, Form elements, cookies and databases queries. A random user browsing on the web page will then execute the malicious script without his knowledge or consent. Often, XSS is used to compromise private information, steal cookies or execute malicious code onto the end user machine. If not filtered appropriately, an example of an XSS attack could be a user navigating on Facebook will automatically execute a script that a friend posted on his wall. An example provided by (Jim, et al., 2007) reviews the XSS script injection attack. This XSS involves injecting a malicious script, typically written in JavaScript into the content of a trusted website using the following command:

```
<script src="malicious.js"></script>
```

This injection results in loading and executing the script on the web browser of each visitor that are viewing the targeted page.

2.5.3 Denial of Service

A Denial-of-service attack (DoS) is characterized by "an explicit attempt by attackers to prevent legitimate users of a service from using that service" (CERT, 2007). Diverse attacks are able to do so:

a. Network flooding that prevent any legitimate traffic going into the network.
b. Connections disruption between two machines.
c. Restriction or Disruption of a service.

In the recent years, attackers have found a way to exploit large pool of computers to launch DoS attacks. In that case, the name Distributed Denial of Service (DDoS) is used. These attacks cause disruption of large organization's network, which generally induce a large loss of profits. DDoS attack used a large pool of computers to attack servers by sending huge amount of network traffic at the same time. This flood attack results in an overflow of the server that cannot reply to genuine packet and end up paralyzing the whole network. There are some ways to protect the network against DDoS attacks. For example DDoS SYN flood attack (Sun, et al., 2009) uses the three way TCP handshakes to massively start connection on the remote server. These connections are never acknowledged (ACK) by the attacking host and stay open on the remote server, which become overflown. A simple example of protection against this kind of attack would be to implement a timer that terminates the connection if no ACK is received in a short period of time. However, it is still hard to mitigate DDoS attacks when hundreds of thousands of computers are used in the process.

2.6 Conclusion

This chapter has compared and contrasted the different interactions and deployment purposes of honeypot systems. This comparison has provided an understanding of the advantages and limitation of different honeypot systems classification. Moreover, Intrusion Detection Systems have been reviewed in order to learn their different behaviours according to their types and detection techniques. Cloud computing has also been introduced and defined through five essential characteristics, three service models and four deployment models. This definition concludes with an overview of different cloud hypervisors. Finally, the last section gives an overview of the three most popular attacks SQL injections, XSS injections and DoS attacks.

The basic understanding of the three technologies described in this chapter is required in the next chapter, Literature Review, which goes further into contrasting these technologies and the research undertaken in their area.

Chapter 3. Literature review

3.1 Introduction

The aim of this chapter is to critically compare the current literature in the research area. The research area has been described in the previous chapter throughout three technologies: Honeypot systems, Intrusion Detection Systems and Cloud Computing.

The choice for an appropriate honeypot system is based on first three sections of this chapter. Firstly, the security issues related to the Cloud computing are evaluated in order to determine the safety of the Cloud Computing environment. Secondly, the literature is investigated to compare previous works done within the area of Signature Generation using honeypot systems. Thirdly, multiple honeypot systems are compared according to their degree of involvements and their embedded features. The end of the third section concludes on the best suited honeypot system for this project, based on the investigation carried out in the previous sections.

In addition, Cross-Site Scripting (XSS) are chosen to be the hacking technique used in this project. This decision is explained and XSS are reviewed to understand their injection mechanisms and to find out methods to detect them. Finally, a research about legal issues concerning honeypot systems is carried out, to discuss whether it is legal to implement one of them.

3.2 Cloud computing Security issues

The cloud computing is a growing interest for organisations considering cutting their IT costs by offloading infrastructure and software costs onto a 3rd party service provider (SaaS, PaaS and IaaS). However, many studies (McDonald, 2011) (Wolf, 2010) show that there are major security issues to take in consideration before moving onto the cloud. Cisco CEO, John Chambers, said during a RSA security conference that the Cloud computing was a “security nightmare” (McMillan, 2009). C. Jarabek (Jarabek, 2009) relates these security issues into three areas: virtualisation’s drawbacks, side-channels information leaks and security managements.

3.2.1 Virtualisation issues

Virtualisation is one of the main components of IaaS cloud computing. Virtualisation enables the implementation of multiples Virtual Machines onto a single piece of hardware. The main security concern stated by (Jarabek, 2009), is that VMs can regularly appear, disappear or become dormant (stopped or suspended) which make them difficult to manage in matters of security patch. The first problem is that service providers have an important volume of

VMs, which make it difficult to be patched at the same time. Secondly, long-dormant machines might take a long time before being patched and could represent a risk when being turned on, cloned or moved to other clouds without the new security patch, see Figure 10. Furthermore, the ability to rollback a VM to a previous state runs the risk of removing the patch of a previously patched security hole.

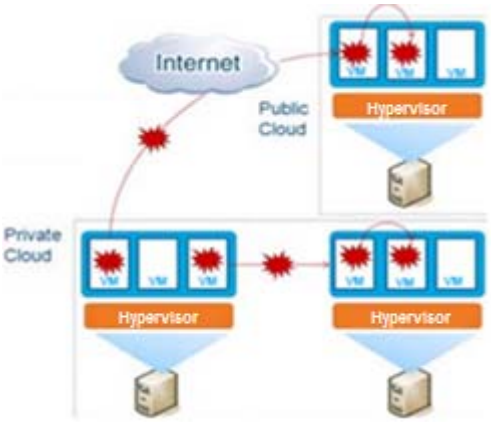


Figure 10 - Vulnerable VMs represent a risk (Jarabek, 2009)

Christodorescu et al. propose a system to counteract the patching issues (Christodorescu, et al., 2009). They describe that the cloud provider has no knowledge about the software (OS, applications, etc.) running on each VM. To counteract this effect, they propose a novel technique using the Interrupt Descriptor Table (IDT) to identify the OS being used on each VM. Then they compare a white-list of known secure code against VM' linked code (system calls, kernel modules, etc.) in order to determine if the VM runs the last security patch for each software.

However, Trend Micro (Agastya, 2011) shows that when using virtual environments, the security updates and antivirus scans should be scheduled carefully. The problem is if every VMs were updating themselves and running antivirus scan simultaneously, it will affect the performance of the server and result into a drain of its resources. This could result in an "antivirus storm" and crash the physical server. To solve this problem, Trend Micro proposes the utilisation of a dedicated security VM on each server that automatically coordinates VMs security updates and scans when they are switched on or cloned, seen Figure 11.

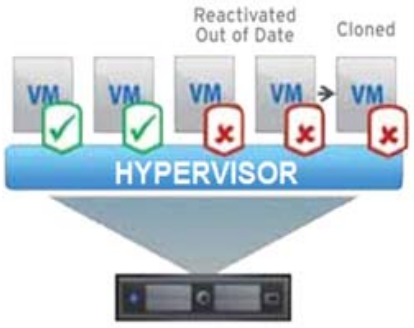


Figure 11 - Dedicated security VMs update vulnerable VMs (Agastya, 2011)

Additionally, another security issue about virtualisation stated by (Agastya, 2011) concerns the Virtual Machine Manager (VMM), which is used to manage each VM. The VMM is a single point of failure that could theoretically be compromised. In that case, it could engender a network of compromised VMs. It is a potential risk but to this day, there are no proofs in the security researches relating of a security whole within the VMM.

3.2.2 Side-channels information leaks

Side-channels information leaks concern the potential communication between VMs co-located on a single piece of hardware. The risk is that a malicious VM could access the sensitive information from a target VM sharing the same hardware resource. In their paper, Ristenpart et al. use the Amazon's EC2 cloud service to investigate this security issue (Ristenpart, et al., 2009). They show how to determine if two instances are running on the same hardware, using some network mapping tools. Then, using a disk-cache side channel, they successfully establish a means of communication between the two VMs in spite of their supposed VM isolation, as seen Figure 12. However, it is nearly impossible that this technique could target a single specified VM because this one needs to be on the same hardware that the attacking VM. Authors claimed an 8.4% success over 1500 targets VM. To counteract the issue of side-channels information leaks, it might be preferable for organisations to pay the cloud provider in order to ensure that customer's VM reside in an independent hardware.

Figure 12 - Side-channels attack (Agastya, 2011)

3.2.3 Security Management

Security Management concerns the tendency among customers to believe that the cloud provider is responsible to manage the security issues relatives to the VMs. These customers are wrong, in general, the cloud provider is not responsible for security issues on each VM. Therefore, each costumer needs to manage his VMs just as he will do in a non-virtualized environment. In their paper, Bleikertz et al. (Bleikertz, et al., 2011) propose a method to perform security audits in the Amazon's EC2 cloud. They assign VMs in to security groups and apply firewall rules specific to each groups. Then, they use the EC2 API to group the results into graphs, which let the administrator now the open ports of each VMs. To determinate which VM has a high-risk of intrusion, they use a vulnerability scanner "Nessus" to do a penetration test on to their VMs. However, the author's security auditing is based on only a single security tool, which provides a limited view of the security vulnerabilities.

To summarize, the security related to the cloud computing is in some parts similar to the one related to physical hosts security. Firstly, the administrator needs to take care of each single instance by applying security patch. Secondly, the hardware where the VMs are hosted should preferably be specific to a single organisation. Finally, customers should

always contact their cloud providers in order to know the division of responsibilities and service expectations if a security breach happened.

3.3 Signatures Generation by the means of Honeypot

Many studies have been realised on attack detection and signature generation based on honeypots. The earliest found is a paper by Levin et al. (J. Levine, 2003): They used a honeypot to collect details of worm exploits in order to generate detection signature. This first attempt did not provide an automated way of generating signature, which results in a lot of hard manual work to analyse worms' details and create signatures.

C. Kreibich and J. Crowcroft were the firsts able to provide an automated way of generating signatures using Honeycomb (Christian Kreibich, 2004). Their method is to analyse the traffic from HoneyD, a honeypot proposed by Neil Provost, in extension with Honeycomb in order to generate intrusion detection signature. All the traffic going through HoneyD is considered as suspicious; therefore, Honeycomb uses the longest common substring (LCS) algorithm, which looks for the longest shared byte sequences across pairs of connections to spot similarities in packet payloads. Then, Honeycomb generates signatures consisting in a continuous substring of a worm's payload and provides the choice to convert the signature into Snort or Bro format. The efficiency of Honeycomb is famous to generate signatures for the worms Slammer and Code-Red-II.

H.-A. Kim and B. Karp designed the Autograph system for larger-scale deployments (Hyang-Ah Kim, 2004). The difference with Honeycomb is the implementation of Autograph within the DMZ. Therefore, Autograph does not only collects traffic considered as malicious but also collects benign traffic. The Autograph system operation mode follows two main stages. In the first stage, Autograph use a port-scanner detection technique on the inbound TCP flows to classify flows as either suspicious or non-suspicious. In the second stage, Autograph generate signature based on the content of suspicious flows' payloads. The generated signature is similar as the Honeycomb's ones: a single, contiguous substring of worm's payload. In their paper, Kim and Karp demonstrate that the Autograph system can detect a newly released Code-RedI-v2 worm's signature before 2% of the vulnerable host population becomes infected.

S. Singh et al. describes the Earlybird system (Sumeet Singh, 2004) as an improvement to the Autograph system. This system eliminates the pre-filtering step and focus on scalable, high-performance implementation. In addition, Earlybird introduces two new metrics such as, content prevalence and address dispersion. The content prevalence inspects all the packets' payloads while the address dispersion tracks source and destination network addresses. Using these 2 metrics, the Earlybird system flags as suspicious invariant packet' payloads sent to a large number of network addresses. In S. Singh et al.'s paper, Earlybird detects precise signatures for the worms Code-Red, MyDoom, Sasser and Kibvu.

A first attempt to generate signature for polymorphic worms is described by J. Newsome et al. using the Polygraph system (James Newsome, 2005). Polymorphic worms are the new generation of worms that mutate as they spread across the network through self-encryption mechanisms. To understand the anatomy of polymorphic worms, they study a sample of a network flow containing an instance of a polymorphic worm. Their analyse shows that polymorphic worms after multiple re-encoding keep some invariant due to the exploit of the same vulnerability. However, they demonstrate that a single, contiguous byte string signature, like in Honeycomb, Autograph and Earlybird, is not efficient at matching polymorphic worms. To solve this problem, they identify a family of signature types more complex based on conjunction, token subsequence and bays signatures. Using all three algorithms, Polygraph generates good signatures even in presence of noise flows and multiple worms. Nevertheless, R. Predisci et al have done controversy research about Polygraph (R. Perdisci, 2006) and they propose attacks using deliberate noise injections that successfully mislead Polygraph's automatic signature generation process.

A recent study by M. Mohssen et al. (Mohammed, et al., 2010) shows that automated signature generation approach for Polymorphic Worms are now able to generate high quality signatures even in presence of noise flows. They propose a new detection method with double-honeynet to capture all worm instances. Then they generate a high-quality signature using a Principal Component Analysis (PCA) that determines the most significant data that are shared between polymorphic worms.

All the researches cited above aims to generate signature for polymorphic/non-polymorphic worms. However, it will be interesting to see the possibility to detect a wider spectrum of attack. A first attempt in that field was propose by V. Yegneswaran et al. who describe Nemean as a system able to generate signature for a wider class of attacks (Vinod Yegneswaran, 2005). The specificity of Nemean is the ability to combine protocol semantics and signature generation algorithm. In V. Yegneswaran paper, Nemean is tested out using NetBIOS exploit, which is detected with a 0% false-positive rate.

In 2006, a study by (Grønland, 2006) focuses on seeing the effectiveness of a honeypot to generate IDS rules based on FTP attacks. During the study, Gronland simulate a fake FTP server using a honeypot and send FTP attacks using a vulnerability scanner. Using the results collected in the log file of the honeypot, Gronland is able to create appropriate Snort signatures to counteract future attacks. This study shows the possibility to collect log data from a honeypot and to create IDS rules based on them. However, the attacks simulated are FTP attacks, which are very rare attacks. In addition, the process of IDS signature generation is not fully automatized and an efficiency test of the honeypot prototype with background traffic is not carried out.

Recently, X. Tang proposed another technique to generate attacks signatures based on Virtual Honeypots (Tang, 2011). He uses the honeypot *HoneyD* with a specific Long Common String (LCS). The signatures are generated as a Snort format. By penetration testing using an exploit named *THCIISSLAME,* the honeypot is able to generate the same rule as the one

originally implemented in Snort. However, this paper does not provide an automated way to generate signatures. Moreover, the paper contains only one experiment and further study would be interesting to see if his system can be adapted to other exploits.

3.4 Honeypots evaluation

One of the leading research organisations in honeypot technology is *The Honeynet Project* (Honeynet, 2011). Founded in 1999, the Honeynet Project is a non-profit research organisation dedicated into investigating the latest attacks and developing open source security tools. The website provides a good list of every available honeypot systems associated with their descriptions. In addition, the investigation carried out in a book by Neil Provos "Virtual Honeypots - From Botnet Tracking to Intrusion Detection" (Niels Provos, 2007) provides a good understanding of some low- and high-interaction honeypots. The following evaluation compares honeypot systems on the server side:

3.4.1 Low/mid interaction honeypots

Labrea created by (Liston, 2003), it is the first low-interaction honeypot introducing the concept of *tarpit.* A tarpit is a service that intentionally responds slowly to incoming request. It is mainly used to slow down spammers and worms by maintaining their TCP connection open while forbidding any traffic over it.

Nepenthes are a versatile low-interaction honeypot able to collect malware. To do so, Nepenthes simulate open ports and known vulnerabilities in order to capture live malware samples trying to exploit these vulnerabilities. An interesting paper by Beacher et Al (Paul Baecher, 2006) states that over a 4 month period, Nephentes has been able collected 15 500 unique binaries. Moreover, four different antivirus engines have scanned those binaries and only 73% were detected as malicious binaries, which mean that the remaining 27% of malicious binaries went undetected through the antivirus engines.

Amun is a low-interaction honeypot developed by (Gobel, 2007). The main objective of Amun is to collect binaries of malware that spread automatically. For this purpose, Amun emulates network vulnerabilities known to attract malwares. Once a malware exploit one of these vulnerabilities, the payload transmitted is analysed and any URL found is extracted. Additionally, the honeypot try to download the malicious file and store it for later analyse.

Honeyd is a mid-interaction honeypot created by Neil Provos in 2003 and last updated in 2007 (Provos, 2007). It is a small daemon able to simulate up to 65536 virtual hosts on a network. Each host can be customized to run specific operating systems with a unique IP address (that does not match to a valid system). Additionally, advanced scripts can be implemented to simulate different services (telnet, SSH, SMTP, FTP…). These scripts go further than a simple banner and can simulate a Login/Password security and answer to few commands typed by an attacker. HoneyD is a mid-interaction honeypot because an attacker can interact with it by sending request and receiving replies.

Dionaea is a mid-interaction honeypot developed in 2009 by the Honeynet Project is the successors of Nepenthes (Honeynet, 2011). The purpose of this honeypot is to gain a copy of a malware exploiting vulnerabilities of network services. To gain a copy of the malware, the honeypot accepts to download files triggered by the malware. These files are confined in a sandbox for later analysis. The main protocol offered by dionaea is SMB, which is a very popular target for worms. Additionally, other network services are provided as HTTP, FTP, TFTP, SIP and MySQL.

3.4.2 High interaction honeypot

A good list of high interaction honeypots is provided by the (Honeynet, 2011). One of the most well-known is **Sebek** created in 2003 by the honeynet project. Sebek is a kernel module installed high interaction honeypot that allows administrators to collect malicious activities such as keystrokes on the system, even in encrypted environments.

The Honeynet Project's 3rd Generation Honeywall ('***Roo v1.4***') framework (honeywall, 2008) is one of the last high interaction honeypot created. This type of honeypot allows the attacker to interact with the system on all levels. Attacker can probe, attack and compromise the system by his own initiative. Many monitor tools installed on the honeypot allow the capture of all the activity done by the attacker, from SSH encrypted sessions to emails and toolkits uploaded. Additionally, a Honeywall gateway, which sits between the honeypot and the outside world, is set up to monitor the network traffic from/to the honeypot system. The Honeywall gateway allows all the inbound traffic to the victim systems but control the outbound traffic with an IDS. This gives the attacker the possibility to interact with the honeypot and by the meantime prevents the honeypot of harming other systems. The Honeywall Roo is a bootable CDROM that installs onto the hard drive the Cent-OS-based distribution and many other tools, which aim to capture and analyse cyber threats.

Table 2 - Differences between each level of involvement based on (Mishra, 2004)

	Low-Interaction	Mid-Interaction	High-Interaction
Degree of involvement	Low	Mid	High
Real operating system	-	-	Yes
Risk	Low	Low-Mid	High
Information gathering	Connections	Requests	All
Compromised wished	-	-	Yes
Knowledge to run	Low	Low-Mid	High
Knowledge to develop	Low	Mid	Mid-High
Maintenance time	Low	Low	Very High

3.4.3 Decision towards a honeypot system

The Table 2 reflects well the result found in the literature in matter of differences between low/mid-interaction and high-interaction honeypot. The literature (Honeynet, 2011) shows

that a lot of researches have been realised in the development of low/mid interaction honeypots in the past couple of years, while in general, far less works have been realised on high-interaction honeypots. The main causes stated by (Spitzner, 2010), are that high-interaction honeypots are very complex to deploy, maintain and require an "extensive tender loving care". Moreover, a badly configured high-interaction honeypot could potentially threaten the whole network.

The honeypot system will be installed within the cloud computing of Napier University and the security issues of this implementation needs to be taken in consideration. The review of the cloud computing security issues demonstrate that cloud, at the time of writing this project, has undoubted security weaknesses.

The very complex configuration of the high interaction honeypot and the security issues related to the cloud computing shows that it will not be appropriate to install a high-interactivity honeypot in the cloud of the University, because there will be a higher risk that an attacker compromises the honeypot and the entire network.

Therefore, the honeypot involvement leans toward a low/mid interaction honeypot. The following honeypot systems are reviewed:

- Labrea is out-dated and its functions are very limited.
- Nepenthes, Amun and Dionaea are used to trick automated malwares into the honeypot in order to get the payload of malicious worms. To do so, they require multiple attempts by the same worm in order to generate a signature. They are able to emulate a single host with multiple network vulnerabilities.
- HoneyD allows the creation of small- to large-scale virtual networks (up to 65000 hosts) using only one machine. HoneyD is open source and written in Python, which make it easily configurable. Moreover, the vulnerability scripts used to simulate different services (Telnet, Web, SSH...) are easily accessible and very flexible.

Nephentes, Amun, Dioneae and HoneyD can all emulate services with specific vulnerability, as required by this study. In comparison and after a lot of research online, HoneyD seems to be more accessible in reason of the multiple sample configurations, service scripts, tools associated and large open forum available on its website (Provos, 2007). Consequently, HoneyD will be used in this project.

3.5 Cross Site Scripting (XSS)

3.5.1 Introduction and Decision

There is a wide repertoire of hacking techniques used by pirates to gather sensitive data and stolen credentials from web servers. According to the Web Hacking Incident Database (WHID, 2011), most popular attacks concern SQL injections, Cross Site Scripting (XSS) and Denial of Service. Statistics concerning the top attack methods used in 2011 have been realised by the WHID. Their results are showed in the following Figure 13:

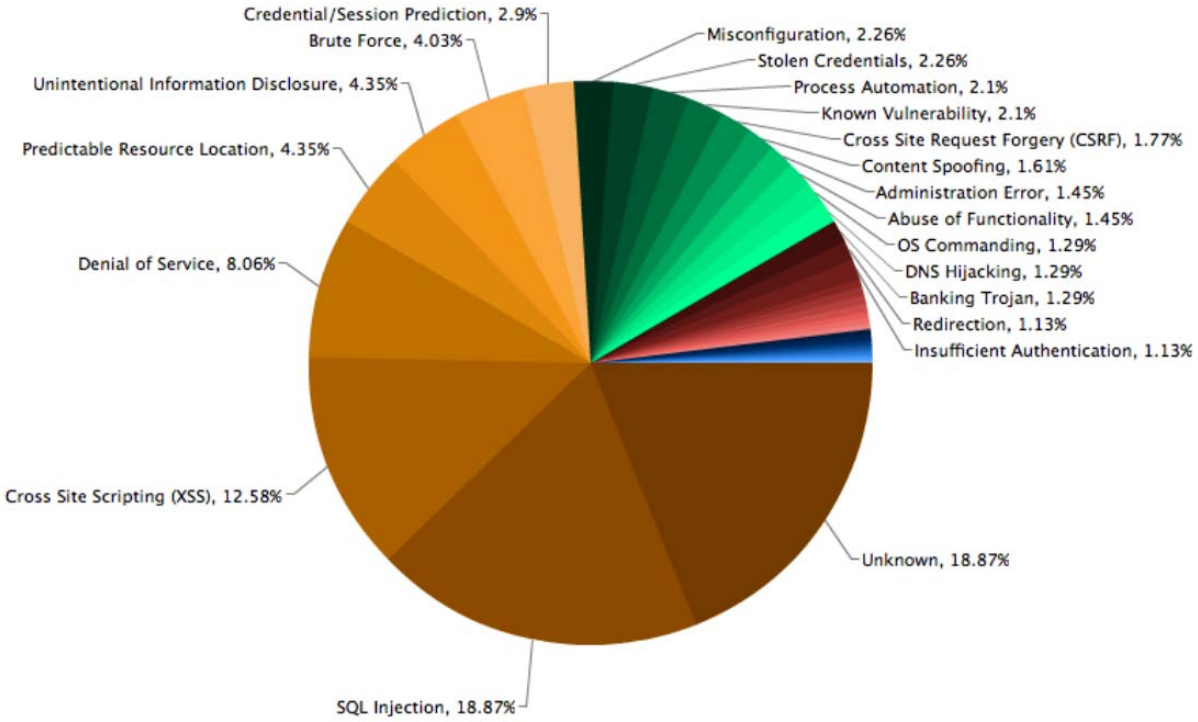


Figure 13 - Statistics of top attacks 2011 (WHID, 2011)

The decision to study the Cross Site Scripting attacks has been made, after introducing the top three attack methods in the chapter Technologies Review 2.5. This decision is based on the unpopularity of this type of attack. Indeed, SQL Injections and DDoS attacks are often at the headlines of the news; however, it is rare to hear about Cross Site Scripting even thought this year it has been the second most used attack on the internet. This section is going to analyse the literature about the injection theory of XSS attacks, the prevention rules used to prevent a XSS injection and examples showing up to date XSS attacks.

3.5.2 Injection theory

Injection is an attack that allows a malicious user to break out of a data context and switch into a code context, through the use of special characters. These special characters injected in a text context will be interpreted by the server side as a code context and could cause serious issues. XSS injections use this technique to insert code into HTML documents. HTML is the predominant mark-up language used in web pages. However, this language is one of the worst of all the coding languages, because it supports so many different types of valid encodings and becomes a total mishmash. Moreover, HTML supports the addition of other languages as XML, JavaScript, URL, CSS, VBScript etc., which make it even more confusing. XSS are injected in the hierarchical structure of the HTML Document Object Model (DOM) through fields that accept untrusted data. This untrusted data corresponds to the data entered by the users of a website. This data can be injected in two different ways (OWASP, 2011):

- Injecting UP is the most common way in XSS injection. This technique uses a special character to close the current context, which results in going up a level in the HTML

hierarchy and to start a new context with a malicious code. An HTML example consists in inserting a closed angle-bracket followed by a new script tag, which result in the execution of the script inserted between the scripts tag, as seen below:

```
<a href=http://website/Link_to_your_website/>      --Normal user
<a href="/> <script>alert("XSS")</script> <"/>     --Malicious user
```

- Injecting DOWN is the less common way in XSS injection. This technique aims to start a code sub context without closing the existing context. An HTML example consist in inserting JavaScript within the current context, in order to introduce a sub context within the src attribute, as seen below:

```
<img src="http://website/Link_to_your_image"/>    --Normal user
<img src="javascript:alert('document.cookie')"/> --Malicious user
```

3.5.3 Detection Rules

To prevent the injection of malicious script through URLs or forms that accept untrusted data, a simple attempt consist in blocking the input of the characters "<" and ">", which are generally used to inject script tags (CERT, 2011). Therefore, an attacker will not be able to inject the previous tag "<SCRIPT>" into a web page because the angled brackets will be detected and blocked. This solution works well but blocks only a small part of the attacks.

According to K. Mookhey, the famous IDS Snort, which has a set of signatures for detecting XSS attack, can be easily evaded by XSS attacks (Mookhey, et al., 2010). Indeed, Snort XSS signatures do not filter hex-encoded values. Thus, Snort would not detect the characters "**<**" and "**>**" that appear as "**%3C**" and "**%3E**" while the tag **<script>** would appear as **%3C%73%63%72%69%70%74%3E.** These hex-encoded values are used to inject embedded scripts tags without being detected by the NIDS.

To avoid this problem, hex encoded values corresponding to the blocked characters, should be also blocked. K. Mookhey gives a list that can avoid most of the XSS attacks by using specifics characters that should be restricted from the user's input, represented in Table 3:

Table 3 - Special characters specification based on (Mookhey, et al., 2010)

Special Characters	Signification
<	Introduce a tag.
&	Introduce a character entity when used in conjunction with some attributes in HTML encoding.
>	Some browsers treat this character as special because they assume that the author really meant an opening angled bracket ("<"), but omitted it in error.
%	Introduce Hexa-encoded values that are used to avoid detection by security mechanisms.
/	End an HTML entity.

In addition to the filtering of special characters, the OWASP foundation provides a XSS prevention sheet composed of a set of eight prevention rules that should stop the vast majority of XSS (OWASP, 2011). These rules aim in reducing the freedom of the user's input of untrusted data into an HTML document.

3.5.4 Recent Cross Site Scripting Attacks

3.5.4.1 A Wide spectrum of websites targeted

XSS attacks are really common, as seen in the Figure 13, 12.5% of the attacks are attributed to XSS (WHID, 2011). However, it is hard to realise that the bigger the web sites are, the more chances they had or have issues with XSS. A good source of recent XSS vulnerabilities can be found on the XXSed project (XSSed, 2011). This website aims to provide up to date information about the last XSS vulnerabilities found by their community. Before being published, the XSS are validated by the moderators, an email is sent to the company targeted and an archive of the vulnerability is saved on their website. By sorting the vulnerabilities by websites, main stream webservers are easily found: Google, Yahoo, Lastminute, Facebook, Twitter, Youtube, Myspace, Ebay, Microsoft… and the list goes on and on. Each one of these websites were, and are still probably, vulnerable to XSS attacks.

3.5.4.2 Up to date example: Facebook XSS

Facebook is a widely used social networking website that has more than 800 million users and 50% of them log on the website on any given day. Last April, two separate cross-site scripting vulnerabilities were uncovered (Harmonyguy, 2011). These holes were used to spread viral links or attacks.

On April 3rd, the first vulnerability came from a page on the mobile version of Facebook. The interface was a prompt used for posting stories to a user's wall. The parameters used for the text input in that prompt were not accurately defined so that the text input was able to escape the filtering process. The vulnerability was a code that opens a connection to the Facebook home page and posts into the wall of the user the message "Wow..cool! Nice page…" followed by a link to a viral page. Anyone attempting to click on this link would have the same code executed and the same message posted on his wall. Luckily, the viral page was not used for malwares spreading but simply for advertisements and spam. This vulnerability was using the code seen Figure 14:

```
<iframe id="CrazyDaVinci" style="display:none;"
src="http://m.facebook.com/connect/prompt_feed.php?display=wap&user_message_prompt='
<script>window.onload=function(){document.forms[0].message.value='Just visited
http://y.ahoo.it/gajeBA Wow.. cool! nice page dude!!!';document.forms[0].submit();}</script>
</iframe>
```

Figure 14 - Facebook XSS text prompt (Harmonyguy, 2011)

Later, on the 7th of April, a second vulnerability was detected and came from the Facebook app page (Harmonyguy, 2011). Despite the filters installed by Facebook to prevent scripts

from apps to modify the page's Document Object Model (DOM), the XSS vulnerability was still able to execute code. Indeed, this code was hidden within a video hyperlink, which is interpreted on Facebook as an image place holder ("href=") followed by the link. Users clicking on this video (that actually works) would execute without their consents the hidden script. Again, XSS made its way around the Facebook filters by using a simple space character to execute a JavaScript code, see Figure 15. Normally, Facebook's filters block JavaScript input but the simple addition of a space preceding the command in the place holder field, was able to pass undetected by the filters and to be executed as JavaScript code by the browser.

Figure 15 - Facebook XSS Javascript (Harmonyguy, 2011)

3.6 Legal issues

The legal issues concerning honeypot systems remain a controversial topic. Most of the articles found on the web are out-dated and only provide the authors' self-opinions. There are no law concerning honeypots deployment in the UK. Nevertheless, on the other side of the Atlantic Ocean, the CyberLaw101 (Radcliffe, 2007) has been adopted in 2007 in the United-States of America: "a primer to US law the deployment of honeypot systems". The major legal issues concern the privacy and entrapment. However, a recent article published on the Symantec website (Spitzner, 2010), stated: "there are no legal cases recorded on the issues concerning honeypot deployment in the US". Therefore, until a judge gives a court order, the law stay blurry.

The legal issues of a honeypot deployment on the cloud are even more unclear because of the multiple countries that could be involved. For example, a French person could implement a honeypot on the cloud. The cloud is based in California and the honeypot receives attacks coming from China. In that case, there would be three countries involved, each one of them with different laws enforcement. Internet can become exponentially confusing and incomprehensible for law authorities.

To have a reasonable idea about the legal issues of a cloud-based honeypot, I contacted the Amazon Web Services (AWS) security group (Amazon, 2011) and asked if it was allowed to implement a honeypot within EC2. Their reply is firm:

« *It is **against AWS Terms of Service** and Acceptable Use Policy to set up honeypots on EC2. Those restrictions are specifically called out in the following:*

> *You may not use, or encourage, promote, facilitate or instruct others to use, the Services or AWS Site for any illegal [or] harmful use...*
>
> *Prohibited activities...include...activities that may be harmful to other, our operations or reputation...*
>
> *Prohibited content include[s]...content or other computer technology that may damage, interfere with, surreptitiously intercept, or expropriate any system, program, or data, including viruses, Trojan horses, worms, time bombs, or cancelbots.*
>
> *Prohibited activities...include...attempting to probe, scan, or test the vulnerability of a System or to breach any security or authentication measures used by a system. System includes any network, computer or communications system, software application, or network or computing device.* »

3.7 Conclusion

Throughout this literature review, many security issues concerning the cloud computing have been discussed. The cloud computing is far from being a safe heaven and was even described to be a “security nightmare” by Cisco CEO (McMillan, 2009). The main concerns related to the security patch of VMs associated with the roll back function, which make them hard to be kept up to date (Christodorescu, et al., 2009). In addition, researchers’ (Ristenpart, et al., 2009) show that VMs from different owners located on the same piece of hardware may be able to communicate together. Therefore, this investigation shows that, to this day, there are many security issues related to the Cloud Computing.

Based on a study by the Web Hacking Incident Database (WHID, 2011), one of the most common application layer hacking techniques aiming webservers are the Cross-site scripting attacks. The world is moving towards the cloud computing because it is cost effective and more convenient. This means that webservers are being moved from being physical hosts into being a virtualized host on the cloud. This virtualization of webservers on the cloud induces that Cross-Site Scripting attacks are, nowadays, aiming cloud based webserver. By following the trend, the first aim of this project is to implement a honeypot simulating a Web server onto the cloud computing. This honeypot aims to detect XSS attacks. A first gap in the literature has been identified during the literature review as no studies have set up a honeypot onto the cloud computing. Initially it was thought the gap was due to the recent emergence of the cloud computing. However, after contacting the AWS (Amazon, 2011), a second reason was found: regulations of cloud providers do not allow the implementation of honeypots on their cloud infrastructure. Luckily, this project is going be implemented into Edinburgh Napier University privately owned cloud infrastructure, which will avoid cloud provider regulations.

The second aim of this project is to generate ACLs and IDS signatures when XSS attacks are detected. Similar works have been done before and have used the honeypot's log file in order to generate IDS signatures to counteract FTP attacks (Grønland, 2006) or worms attacks (Tang, 2011). However, a second gap was identified in the literature because the signature generation always required user's intervention during the process. To fulfil this gap, the second aim is redefined into fully automate the whole signature generation process.

Additionally, a third gap was found in the honeypot literature: studies reviewed show that honeypots can detect attacks even if background traffic is injected. However, there are no studies relating the background traffic speed breaking point to the honeypot's stability.

The decision towards a honeypot system is based on a comparison of multiple honeypot systems according to their level of interactivity. Despite of keeping the risk low in Edinburgh Napier University Cloud, the decision was taken not to use a high interaction honeypot. Therefore, the choice was based on a panel of low/middle interaction honeypots. After a comparison of the features provided by each honeypot, the choice was taken to use HoneyD (Provos, 2007) because it seemed to be more accessible than other honeypots, according to the multiple sample configurations, service scripts, tools associated and large open forum available on its website.

Finally, by carrying out the literature review, three gaps have been highlighted within the research area. The aim of the next chapter is to propose a prototype design that will fulfil these gaps.

Chapter 4. Design

4.1 Introduction

The literature review has outlined that previous works in the area of honeypot systems do not permit fully automated generation of signatures for NIDS. In addition, no studies discuss the installation of a honeypot within the Cloud Computing and no studies refer to measuring the background traffic breaking point to assess whether the honeypot becomes unstable. This chapter is going to fulfil these gaps by designing some scripts to fully automate the whole process of ACLs and Snort signature generation. In addition, a design of the honeypot within the cloud infrastructure will be proposed. Finally, this chapter will design some experiments to find out the effectiveness of the prototype created and the breaking point of the honeypot systems. The set up for this implementation will be operated within Edinburgh Napier University's private cloud computing infrastructure and kept for future use by the students.

4.2 Network Architecture Overview

An ideal network architecture design is to filter the network traffic before it enters inside the cloud infrastructure. To do so, the design of the prototype network architecture is split into two sections: The first section is composed of two security layers that analyse the IP packet and content before entering the cloud infrastructure. The second section is composed of a security sensor (honeypot) placed within the cloud infrastructure, which is able to detect undiscovered threats.

On the front end of the network, a cisco router is placed and followed by Snort NIDS. The router is used to filter the traffic by using IP access control lists known as ACLs. The configurations of ACLs allow to filter network packets based on their IP source/destination, protocol and port. ACLs are implemented on the interface of the router desired and filter inbound and/or outbound traffic. Each ACL have specific actions (log/drop) to do when IP packet match is discovered. ACLs are effective to filter IP packets, to block suspicious individuals or restrict access to specific services. However, they cannot examine the content carried out within the packet, which can be harmful to the network. To examine the content carried out by a packet, the NIDS Snort is placed below the router and provides a second layer of security. Snort examines the packet's content against its set of signatures, to determine the packet maliciousness. Here again, if a content match is discovered, specific actions will be taken to log or drop the packet. The choice of placing the router on the front end and the NIDS above is well thought: the router can process packets quicker than the NIDS. Therefore, less traffic will be sent to the NIDS, which will ensure a low congestion in

the network. Once a packet successfully passed these two security layer, it will be forwarded to the cloud hypervisor.

The cloud hypervisor acts like a router and determine which packet goes to which instances. Instances can be any types of machines or servers allowed by the hypervisor. The honeypot system is one of these instances. The honeypot is able to emulate a template that simulates a fake web server. Packets received by the honeypot are analysed to determine their maliciousness, which will be explained in the next section. The following network architecture draft and flow chart show the different decisions taken when a packet goes through the network architecture. Both Figure 16

Figure 16 and Figure 17 have been place next to each other to ease the understanding of the device decisions within the network.

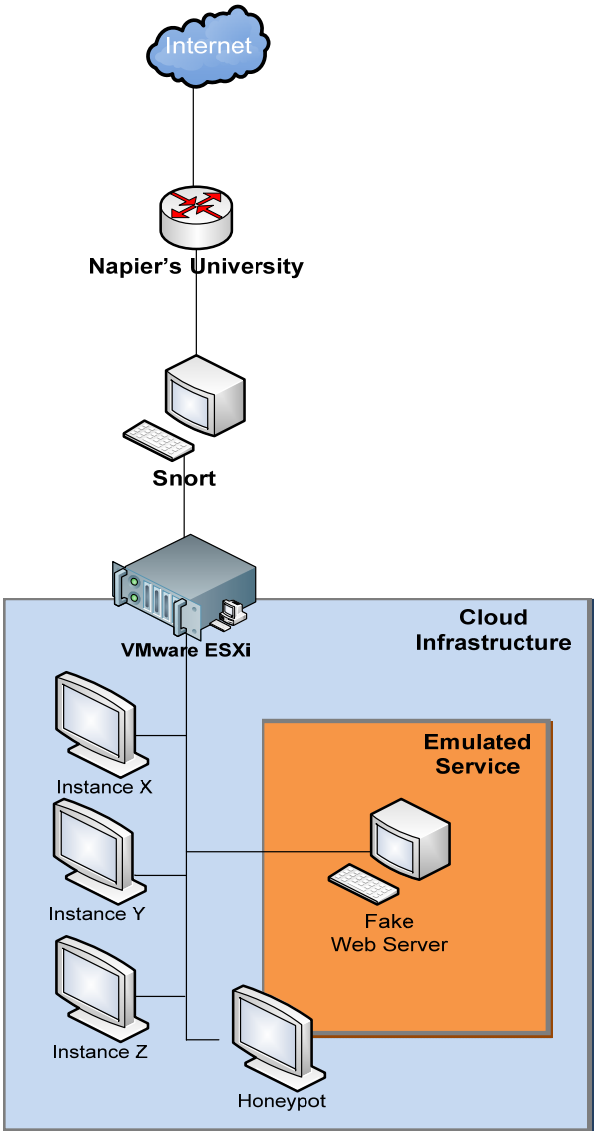


Figure 16 – Ideal Network Architecture Design

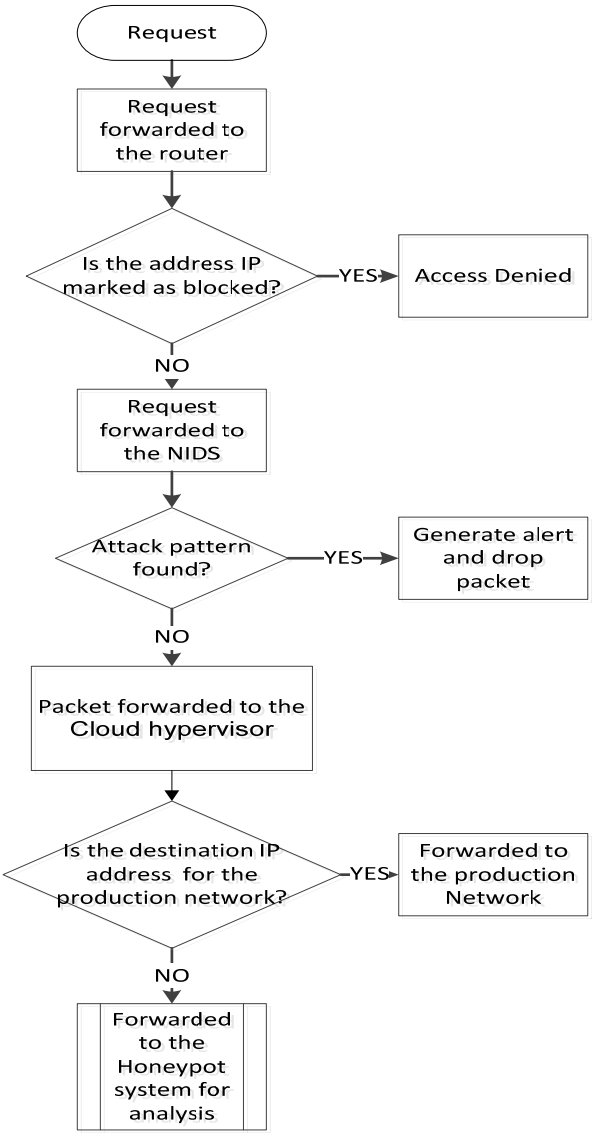


Figure 17 - Flow chart paths choices for incoming packets

This network architecture design would be the best architecture to ensure security and high efficiency of the network. However, for the rest of the project, instances for NIDS and the

attacker will be implemented within the cloud infrastructure. Implementing these instances on the cloud would be easier to manipulate and provide the same results than if the NIDS/attacker were outside the cloud.

4.3 XSS Simulation and Logging capability

To simulate and log XSS attacks, the first step consists in creating a source that will simulate the attacks. To do so, XSS vulnerabilities will be retrieved from the "XXSed.com" website, which is a database of recent XSS vulnerabilities found by the website's community, previously seen in the Literature Review 3.5.4.1. By using this database, a list containing 50 XSS will be established, which will give a large panel of realistic attacks that could be used against a webserver.

The second step consists in creating a target able to handle and log these XSS attacks. This is done by building a HoneyD's template that simulates a web server. This web server will host a web page that will be closely monitored and any interaction with it will be logged.

Finally, once these attacks are simulated and every interaction with the web page detected, the last step consists in choosing a logging method, which are detailed in Appendix A.4. The requirement for this logging method is that it should log any interaction with the web page. Therefore, the service level logging capability is chosen because it can log any interaction with the service emulated (web server).

4.4 Detection of XSS attacks and Generation of ACLs/Snort Signatures

4.4.1 Extracting content and IP address

The generation of ACLs and Snort signatures involve the understanding on how these security measures works. The section reviewed in Design 4.2 explains that ACLs are mainly used to filter packets based on their IP addresses, ports and services, while the Snort signatures are used to analyse and filter suspicious content carried in the packet.

Concerning the generation of ACLs, this study only aims in discarding suspicious IP/individuals that have interacted with the honeypot, because any interaction with the honeypot is considered as malicious activity. Therefore, IP addresses that have interacted with the honeypot will be extracted from HoneyD's log and inserted in an ACL template. These ACLs, once implemented in the router, will block future packets coming from these IP addresses.

The detection of XSS attack, in this study, is based in detecting the injection of the "<script> </script>" tags or its equivalent in hex-encoded value "%3Cscript%3E %3C%2Fscript%3E" into the webpage, as seen in the Literature Review 3.5.2. These tags are widely used to inject malicious scripts into webpages and a common way of evading security filters is to

replace their angle brackets by their equivalent in hex-encoded value. Every interaction with the webpage is monitored and logged into HoneyD's log file. The aim of the prototype script is to automatically extract this information and parse it into a Snort signature template that will be implemented in order to filter future malicious traffic. An example can be seen in the following example:

- HoneyD's log file:

```
--MARK--,"Thu Oct 13 11:14:03 EDT
2011","webmin/HTTP","192.168.230.130","192.168.230.140",61176
,80, "GET /gp/seller/product-ds/registration.html?ld=<script>
alert(document.cookie%)</script> HTTP/1.1
                    ...
",
--ENDMARK--
```

- Snort signature template:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"XSS
attempt: script injection detected"; flow:to_server,
established;content:"<script>alert(document.cookie%)</script>
"; classtype:attempted-admin;sid:1000001;)
```

- ACL template:

```
170 deny tcp host 192.168.230.130 any
```

4.4.2 Programming language decision

Nowadays, many programming language are available such as C/C++, Perl, Python, Bash, JavaScript, etc. The generation of ACLs and Snort signatures involves a lot of interaction with the UNIX shell through specific commands. Therefore, the programming language required should provide an easy way to do so. This specificity narrows the research down to two programming languages: Perl and Bash. Perl is a very powerful language, widely used and can be combined with many plugins. On the other hand, Bash is more limited but offers a great interaction with the shell and is easier to maintain. In comparison, Bash is generally used for short scripts (under 100 lines) while Perl is used for bigger ones. The scripts that will be created in this chapter will be short and will require a lot of system command interaction. Based on these facts the choice is made to use the Bash language to code the required scripts.

4.5 Experimentations

The two previous sections show the creation of XSS attacks, the detection of these attacks and the generation of ACLs/Snort signatures. This section implements four experiments that will be used to determine if the prototype is working properly and to calculate its effectiveness in Lab environment and in a simulated real world environment. A summary of these experiments has been realised in the Figure 18.

In these experimentations ACLs are not implemented, because they will block all the network traffic coming from the attacker host.

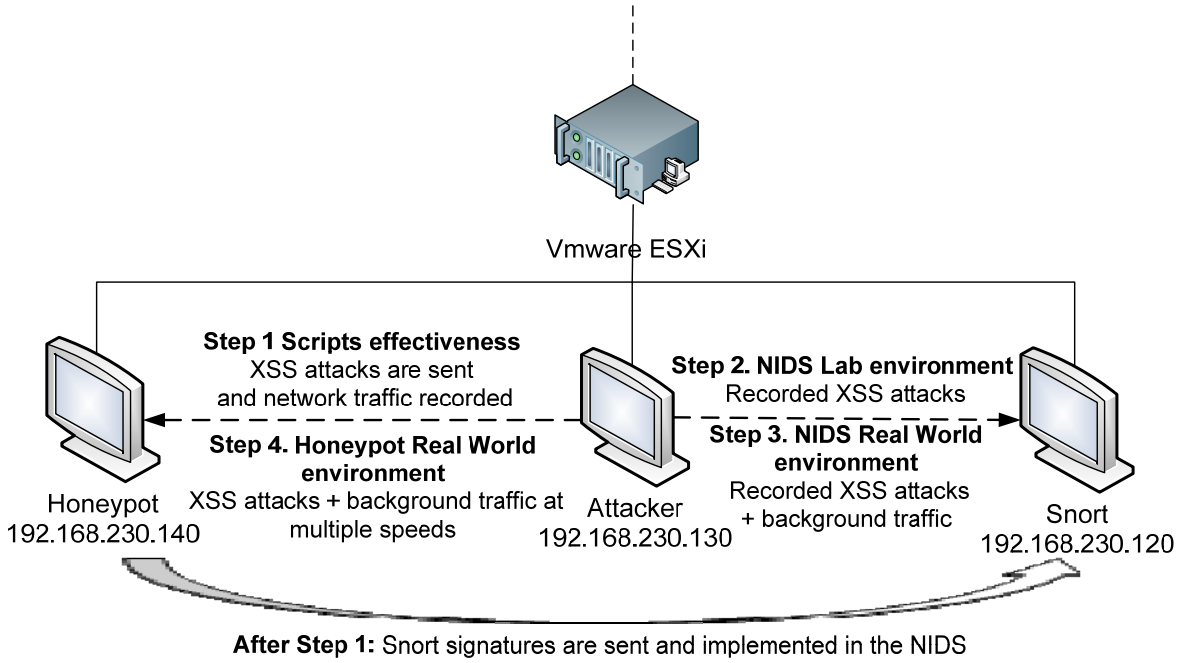


Figure 18 – Experiments steps summary

4.5.1 Scripts effectiveness

The scripts experiment aims in testing the ability of the scripts to generate ACLs and Snort signatures. To do so, the list of XSS attacks is going to be injected on the honeypot and should result in generating a number of ACLs/Snort signatures. During the injection phase, the network flow is going to be recorded using Wireshark and saved as "honeyd_attack.pcap" file. The expected results should show that the Snort signatures generated should correspond to the number of attacks containing a "script" tag.

At the end of this test, the Snort signatures generated by the honeypot will be automatically sent and implemented in Snort

4.5.2 NIDS within Lab environment

The NIDS lab environment experiment aims in testing the Snort signature effectiveness in a controlled environment. Indeed, the snort signatures generated have been implemented in the NIDS and the network flow of the attack has been recorded in the "honeyd_attack.pcap" file. This experiment needs to modify the end points of the network flow "honeyd_attack.pcap" by using the Tcpreplay suite. The modification will change the IP addresses from Attacker-Honeypot, to Attacker-Snort and save the modified file in "snort_attack.pcap". This file will be sent on the network using Tcpreplay. The expected results should show that the number of alerts raised by Snort is identical to the number of signatures implemented.

4.5.3 NIDS in Real World simulation

The NIDS real world simulation experiment aims in testing the prototype effectiveness with background traffic. Indeed, a real world implementation would involve a wide mix of genuine network traffic going through the NIDS. This background traffic will be generated by using the UNIX tool TCPreplay, which will send a file “snort_background.pcap” containing network packets onto the network. This file packet is free of attacks and comes from the Massachusetts Institute of Technology (MIT, 2011). The IP addresses of this file packet will be rewritten during the implementation to suit the needs of this experiment.

The background traffic “snort_background.pcap” and the previous set of XSS attacks “snort_attack.pcap” will be mixed together and sent to the NIDS (at low speed 10mbps). Hopefully, the expected result should show that the background traffic did not raise any false positive alerts.

4.5.4 Honeypot in Real World simulation

This last experiment aims in testing the effectiveness of the Prototype created. This experiment is similar to the previous one but aims in testing the honeypot effectiveness through increasing network traffic. The Tcpreplay suite is able to send the packets “honeyd_background.pcap” at a predefined speed on the network. XSS attacks will be injected in parallel with the background traffic. The background traffic speed will be increased from 0mbps by step of 0.5mbps. Two measures will be taken according to the background traffic speed: The latency to send the 50 XSS attacks and the number of attack detected (equal to the number of signature raised).

Expected results at a low speed should indicate the same number of alerts, as detected previously in the lab environment. However, at a higher speed, the expected results are that HoneyD will have a high amount of traffic to analyse and it will probably start dropping packets, which involve missing out XSS attacks. The experiment will show the breaking point of the prototype under a certain amount of network traffic.

4.6 Conclusion

The network architecture designed, proposes to implement the honeypot within the production network. In that way, the honeypot is able to detect XSS attacks that bypassed security devices placed on the front end of the network, such as the NIDS and the router. The detection of an XSS attack by the honeypot generates an ACL, to block the IP address source of the attack, and a Snort signature to block future incoming packet using the same attack content. The ideal network architecture put forwards in this chapter would be to install a boundary router followed by a NIDS to provide a double filter of incoming packet.

In order to generate ACLs and Snort signatures, the design proposes to create two Bash scripts (one for each generation) that automatically extract relevant information from the honeypot log file and parse it into its template (ACL or Snort signature). This fulfils the gap

found in the literature review, which states that there are no mechanisms that provide an automated way to generate ACLs/Snort signatures.

A set of four experiments are designed to test out the honeypot and the NIDS. However, a different network architecture design is created in order to ease the implementation of the experimentations. In addition, the experiments do not implement the ACLs created into a router. These ACLs have only been created to show the possibility to generate ACLs using a honeypot system.

The first experiment aimed to evaluating the effectiveness of the Bash scripts created in detecting XSS attacks and generating ACLS/Snort signatures. The Snort signatures generated are then implemented into the NIDS for the next two experiments. These experiments inject the same XSS attacks but into the NIDS. They evaluate firstly, if the NIDS is able to detect the XSS attacks by using the new set of signatures and secondly, by injecting background traffic to evaluate if any False Positive alarm is raised. Finally, the last experiment consists of injecting background traffic and XSS attacks into the honeypot to determine the ability and latency to detect the attacks under increasing background traffic. This experiment put forward a breaking point, due to the increasing background traffic, after which the honeypot became unstable. This last experiment has fulfilled the second gap found in the literature review, which states that there are no experiments showing the breaking point of the honeypot.

In the next chapter, the design of the scripts and experiments will be implemented within the cloud computing of Edinburgh Napier University. This implementation will fulfil the last gap found in the literature review, which is the gap about cloud-based honeypot implementation.

Chapter 5. Implementation

5.1 Introduction

This chapter aims to create scripts and implement experiments designed in the previous chapter. The previous chapter has clarified the aims of each script and experiment needed for this project.

In the previous chapter, the scripts required to generate ACLs and Snort signatures have been designed. The design showed that specific information of the honeypot log file can be extracted and injected into appropriate templates to create ACLs and Snort signatures. In addition, the previous chapter presented a set of four experiments, which tested the effectiveness of the honeypot coupled with the scripts in detecting XSS attacks. These experiments are to be implemented in this chapter.

The network architecture of the implementation is composed of three instances, in order to simulate a honeypot, a NIDS and an attacker. The configuration of these instances is described within the following chapter, however, background understanding of the features and capabilities offered by the honeypot “HoneyD” and the NIDS “Snort” are respectively proposed in Appendix A. and Appendix B. which eases the understanding for the reader.

5.2 Cloud configuration and Test bed

The network architecture seen in the Design 4.2 requires the creation of a unique instance to simulate the honeypot (Fake Web Server). However, for this implementation, the design needs to be adapted in order to simulate also the instances of the NIDS and the Attacker. The cloud hypervisor available at the University is VMware ESXi with vCenter Lab manager. The requirements to create three instances can be quickly achieved through lab manager, within a couple of clicks. The portal of lab manager is accessed through the following address: https://lm2003.napier.ac.uk/. The login credentials must be pre-activated by Bill Buchanan to enable the connection onto the platform (Buchanan 2011). Once connected on the platform, three instances are created and deployed, all under the operating system BackTrack5, as seen Figure 19. The choice for the operating system BackTrack5 instead of Widows or other UNIX platform is done because BackTrack5 provides a large panel of security tools pre-installed. This saves a lot of time, because it is generally a troublesome task to install new software in a UNIX environment.

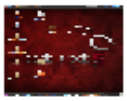
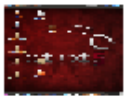
Consoles	Configuration Name	Status	Owner
	Attacker BT5	Deployed	Jacob, Benoit
	Honeypot BT5	Deployed	Jacob, Benoit
	Snort BT5	Deployed	Jacob, Benoit

Figure 19 - Deployment of instances

The instances have been deployed on the student network, which means that they are able to communicate together within the same subnet. The access to the outside (internet) is enable, although not needed in the following experiments, but could be useful in order to retrieve the last packages needed for the security tools. The test bed of the network implementation of the three instances is represented below, Figure 20.

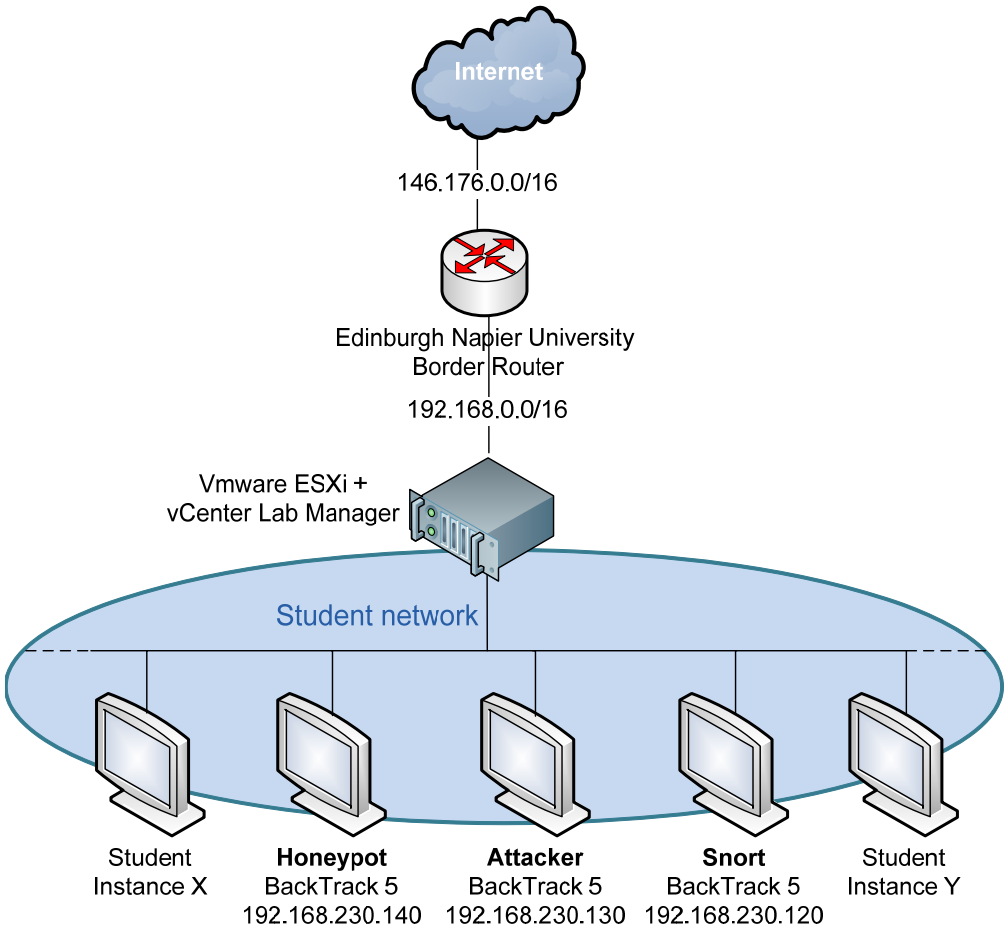


Figure 20 - Instances implementation overview

5.3 Instances configuration

The configurations of the three instances involve setting up a NIDS “Snort”, a honeypot “HoneyD” and an attacker instance. These instances are running the Operating System BackTrack 5. Even though the setup of these instances is not very hard to realise, there are a lot of command line options to choose from, and it is not always obvious which ones should be used and in which context. This section aims to make the implementation understanding easier for beginner users.

5.3.1 HoneyD

5.3.1.1 Basic installation

The honeypot system used for this study is HoneyD. A review of HoneyD’s features and capabilities is provided in reviewed in Appendix A. The installation of HoneyD under a UNIX system is done with the command “apt-get install HoneyD”. Luckily, the prompt command informs that HoneyD is already installed in the system. This is because Backtrack 5 comes with pre-installed security tools and apparently includes HoneyD. However, the installation path of HoneyD is not indicated, so the “locate” command is typed in to find out where it is:

```
honeypot@bt:~# locate honeyd
   /usr/local/share/honeyd
```

By default HoneyD does not has any service scripts implemented. These services scripts are used to simulate services as FTP server, Email server, Web server, etc. A package containing service scripts “service.tar” is retrieved from HoneyD’s website (Provos, 2007) and extracted using the following command:

```
honeypot@bt:~# tar –zxvf honeyd.tar /usr/local/share/honeyd/
```

5.3.1.2 Configuration file

HoneyD’s configuration file is used to create some virtual devices. This is done by first setting up a template named “Webserver” and the template personality named "Linux kernel 2.4.20". These options are used to deceive fingerprinting tools by making the system looks like genuine. The uptime, which defines how long the system will stay on, is configured to stay awake for a long time. The “add” command defines the information related to a service script. In this configuration, the service script webmin.sh (used to emulate the webserver) is configured on the TCP port 80. Any other traffic that use the protocols UDP or TCP will be reset. The name of the Ethernet card used in the configuration is defined by “3com”. Finally, the template Webserver is “bind” (associated) with the IP address 192.168.230.140 (which should be a non-used IP address). Therefore, the IP address of the Webserver is 192.168.230.140. The configuration file is presented below:

```
#HoneyD’s WebServer configuration file
create Webserver
set Webserver personality "Linux kernel 2.4.20"
```

```
set Webserver uptime 9999999
add Webserver tcp port 80 "sh ./scripts/suse7.0/webmin.sh”
set Webserver default tcp action reset
set Webserver default udp action reset
set Webserver ethernet "3Com"
bind 192.168.230.140 Webserver
```

5.3.1.3 Service script adaptation

The service script used in this study is Webmin.sh. It was created by Fabian Bieker to emulate a web page with an authentication form. The script is 407 lines long and provides multiple options that are not needed in our experimentations. However, two lines need to be changed in the Webmin.sh file, in order to make the service script work. The first line should indicate the full path to the file base.sh, which is located in the script directory created earlier. The second line defines where the service script writes its log messages. This line should be configured to save log message in the file honeyd/msg.log. An overview of Webmin.sh and the modified lines is shown below:

```
#!/bin/sh
# Webmin.sh
     (... text...)
. /usr/local/share/honeyd/scripts/base.sh
     (... text ...)
LOG="/usr/local/share/honeyd/msg.log"
```

5.3.1.4 Starting HoneyD

Before getting HoneyD started, two last commands need to be typed in. The first commands change the attributes of the files “msg.log”, “webmin.sh” and “base.sh” in order to enable everybody to access them. This is required otherwise HoneyD cannot access these files and fail to start up, to start a service or to log alerts.

```
honeypot@bt:/usr/local/share/honeyd# chmod 777 ./msg.log
honeypot@bt:/usr/local/share/honeyd# chmod 777 ./scripts/base.sh
honeypot@bt:/usr/local/share/honeyd# chmod 777
./scripts/suse7.0/webmin. sh
```

HoneyD is now configured and ready to start. To start up, the last command consists in indicating HoneyD the network interface chosen, the emplacement of the configuration file and the IP range attributed. The command is shown below:

```
honeypot@bt:/usr/local/share/honeyd# sudo honeyd -d -i eth1 -f
myconfig.conf 192.168.230.140
```

5.3.2 Snort

The NIDS Snort is provided in the BackTrack 5 distribution. However, before starting the program, the network interface and the configuration file need to be modified.

5.3.2.1 Network interface

First of all, an “ifconfig” command needs to be typed in, to determine the network interface that Snort will be listening on.

```
snort@bt:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:0c:29:81:2c:81
          inet addr:192.168.233.128  Bcast:192.168.230.255
Mask:255.255.255.0
                            ...
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
                            ...
```

The “ifconfig” command shows two network interfaces: eth1 and lo. Eth1 is the network interface connected to the outside (needed for this experiment) while “lo” is a loopback interface generally used for testing purposes.

5.3.2.2 Configuration file

Snort’s configuration file “snort.conf” is used at start up time. It contains six basic sections: “variable definitions”, “config parameters”, “pre-processor configuration”, “output module configuration”, “new action types” and “rules configuration”. For this implementation only the last section “rule configuration” needs some modifications. In the default configuration, Snort includes paths to multiple rules files. These rules files have specific patterns that are used by Snort to detect an attack. In this project, these files are not needed, therefore they should be ignored. However, the rules file “snort.rules” that contains the XSS attacks signatures, created during the implementation, needs to be added in the configuration file. Therefore the path to this rules file is added, while other rules files are commented by using a # character, such as the following:

```
snort@bt:/etc/snort# nano snort.conf
           ...
#############################################################
# Step #6: Customize your rule set
#          ...
#############################################################

include $RULE_PATH/snort.rules
#include $RULE_PATH/local.rules
#include $RULE_PATH/bad-traffic.rules
#include $RULE_PATH/exploit.rules
           ...
```

5.3.2.3 Starting Snort

Once the network interface is found out and the configuration file modified, the command line used to start Snort can be typed in, such as the following:

```
snort@bt:/etc/snort# snort -A console -i eth1 -l ./log -c snort.conf
```

```
        --== Initializing Snort ==--
                ...
        --== Initialization Complete ==--
                ...
```

This command line has four options: “-A console“ specifies that alerts will be shown in the console, “-i eth1” specifies the network interface used to capture traffic, “-l ./log” specifies the logging directory for the alerts and “-c snort.conf” specifies the configuration files.

Once the command is typed in, Snort starts initializing each plugin one by one and prints an “initialisation complete” message to confirm that the start-up went successfully.

5.3.3 Attacker

The configuration of the attacker’s instance consists is split into two sections: Firstly, the creation of a list of XSS attacks that will be injected in the lab and real world experiments, and secondly, the installation of the Tcpreplay suite needed to generate background traffic for the real world experiment.

5.3.3.1 XSS list

A list of 50 XSS is created based on recent XSS attacks found on the XSSed.com website. This list is composed of various types of XSS attacks that use different injection methods. Therefore, some of these XSS do not use the traditional “script” tags but use different tags or encryption methods. The sources of these attacks come from multiple websites as Google, EBay, Amazon, Yahoo and so on. However, for this project the website names (www.website.com) will be deleted in order to keep only the URL string without the website name. The full XSS list is provided in the DVD-ROM enclosed with this thesis, but the following gives three examples of what it looks like:

```
/NEWS/?act=VIEW&ano=958&pst=-1&startno=1121&pageno=9&FID=
%22%3E%3Cscript%3Ealert%28%2Fwww.r3t.n3t.nl%2F%29%3C%2Fscript%3E

/gp/change-password/-%22%3E%3Cscript%3Ealert%28document.cookie
%29%3C/script%3E-.html

/mailto?prop=movies&locale=us&url=&title=irsdl"STYLE="width:expressi
on(alert(/Internet Explorer Only-Irsdl is here again/));
```

5.3.3.2 Tcpreplay suite

The Tcpreplay suite is a set of tools that can inject previously captured traffic onto the network. Additionally, Tcpreplay can rewrite the layer 2 and 3 header and replay the traffic at different speeds throughout the network. This tool is perfectly suitable for the real world experiments. The installation procedure is straight forward by using the “apt-get install tcpreplay” command, which automatically install the Tcpreplay suite composed of Tcpreplay, Tcprewrite and Tcpprep:

```
attacker@bt:~# sudo apt-get install tcpreplay
          ...
```

```
The following NEW packages will be installed:
  tcpreplay ...
```

5.4 Prototype scripts creation

5.4.1 Scripts overview

ACLs generation and Snort signatures generation require different operations and methodology. Therefore, they have their own specific script written in Bash language, previously chosen in the Design 4.4.2. The scripts require to extract specific data from HoneyD's log and to inject them into ACL or Snort signature templates. This needs to be done in an automated way to fulfil the gap found in the literature review. An overview of the main functions required by these two scripts has been realised in the following flowcharts Figure 21 and Figure 22:

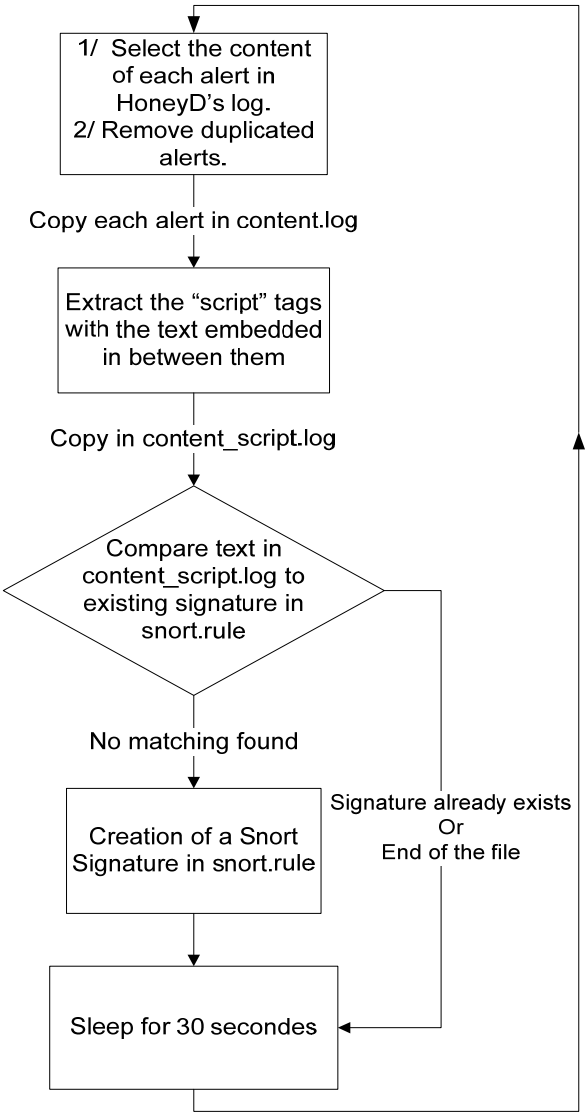


Figure 21 - Flow Chart Snort Script prototype

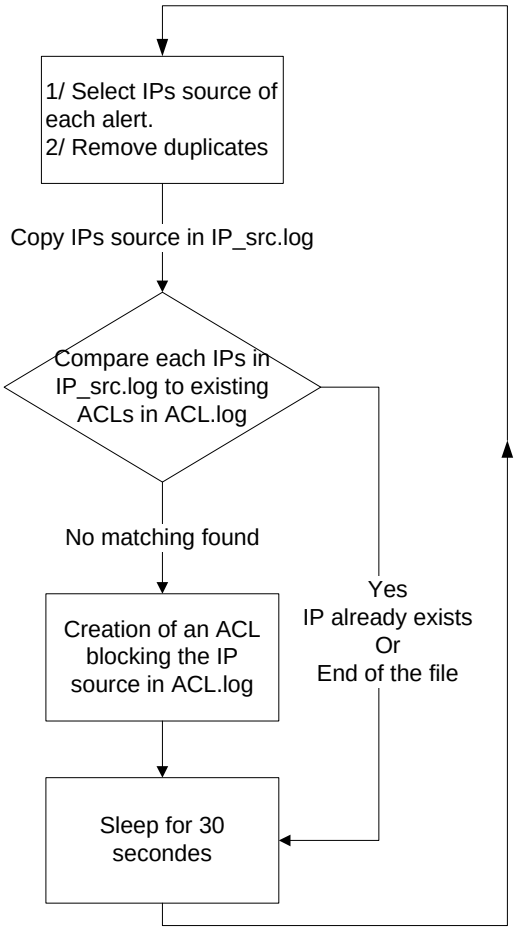


Figure 22 - Flow Chart ACL script prototype

5.4.2 Snort signature script

The following Snort script is a coded representation of the flow chart seen above. To resume, this script extracts the URL string that has been logged by HoneyD, analyse it to find "<script></script>" tags or hex-encoded equivalent, and parse the result into a Snort signature template. This Snort script has been created especially for this study and the explanation following describes its main operations. In addition, another explanation is provided in the Appendix C.1, which gives a line by line description of each single operation.

```
#Snort_script.sh
#!/bin/ksh
count=1000000
(
while true
do
nawk -F /^ '/GET/ {print $0}' msg.log | cut -d' ' -f2| sort | uniq
> content.log
grep 'script>\|script%3E \|script%3e' content.log | sed -e
"s/.*<script>//;s/<\/script>.*//" | sed -e
"s/.*%3Cscript%3E//;s/%3C\/script%3E.*//" | sed -e
"s/.*%3Cscript%3E//;s/%3C%2Fscript%3E.*//" > content_script.log
cat content_script.log |
(
while read line
do
res=`grep -c $line snort.rules`
     if [ $res == 0 ];
          then echo 'alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:" XSS attempt: script injection detected ";
flow:to_server,established; content:"'$line'"; nocase; sid:
'$count';)'\'>> snort.rules count=`expr $count + 1 `
          else echo 'Snort signature '$line' already existing'
     fi
done
)
 sleep 30
done
)
```

The Snort script begins with a "count" variable, which will be used later on to identify each signature with a unique number. A loop starts with the nawk option, which retrieves each line of HoneyD's log (msg.log) starting by /GET. These lines show the interaction from an attacker onto the webpage (commands typed, URL strings, etc.). Results found are saved in "content.log". Then, each line is examined to detect any tags "<script></script>" (or hex encoded angle bracket equivalent), which are widely used tags in XSS attacks. Positive results are saved in "content_script.log". Then, each line of this file is read and compared to the existing Snort signatures file "snort.rule" (to avoid duplicated signatures). If a match is

found, the variable “res” is incremented and an echo is displayed. If no results are found, the line is injected into a Snort signature template and sent in “snort.rule”. Finally, the prototype script waits 30second before doing a loop.

5.4.3 ACL script

The following ACL script is a coded representation of the flow chart seen in Figure 22. To resume, this script extracts the IP address source of every machine that interacts with the web page. As nobody should be interacting with the honeypot, any interaction is considered as suspicious. This IP address is then injected into an ACL template. This script has been created especially for this study. An explanation is given after the script, but a more detailed explanation, with a line by line explanation can be found in the Appendix C.2.

```
#ACL_script.sh
#!/bin/ksh
count=200
(
while true
do
nawk -F"[\",]" '/MARK/ {print $9}' msg.log | cut -d' ' -f1| sort |
uniq | grep -v '^$'> IP_src.log

cat IP_src.log |
(
while read line
do
res=`grep -c $line ACL.log`;
if [ $res == 0 ];
     then echo ''$count' deny tcp host '$line' any'\>> ACL.log
     count=`expr $count + 10 `
else echo 'IP match found'
fi
done
)
 sleep 30
done
)
```

The ACL prototype script begins with a “count” variable, used to identify each ACL with a unique number. A loop starts with the nawk option, which retrieves each lines of HoneyD’s log (msg.log) starting by /MARK/. From these lines, IP addresses source are extracted, duplicate addresses are removed and the result is sent to “IP_src.log”. Then, each line of this file is read and compared to existing ACL file “ACL.log” (to avoid creating duplicated ACLs). If a match is found, the variable “res” is incremented and an echo is displayed. If no results are found, the line is injected into an ACL template and sent in “ACL.log”. Finally, the prototype script waits 30second before doing a loop.

5.4.4 Snort signature transfer

This last script is a short example that shows the possibility to automatically transfer the Snort signatures file “snort.rule”. This is done by starting a SSH connection and transferring the file using the “SCP” option command. In order to work, the SSH connection needs to be preconfigured as a passwordless connection. The steps to configure a passwordless connection by using private/public keys pair are provided in Appendix D. The following script establishes a SSH connection, transfer the file “snort.rule” from the honeypot to the NIDS Snort and wait 60 seconds before making a loop:

```
#SCP_script.sh
#!/bin/ksh
(
while true
do
scp /(client_snort_signature_folder)/snort.rule
ben@192.168.230.120:/(NIDS_directory_signatures_folder)/snort.rules
sleep 60
done
)
```

5.5 Experimentations

The instances have been configured and the scripts have been created in the previous section of this chapter. The following section presents the implementation of the set of experiments designed in the Design 4.5. In order to ease the understanding for the reader, a UML sequence has been created in following Figure 23. This UML sequence represents each interactions involved in each experiments.

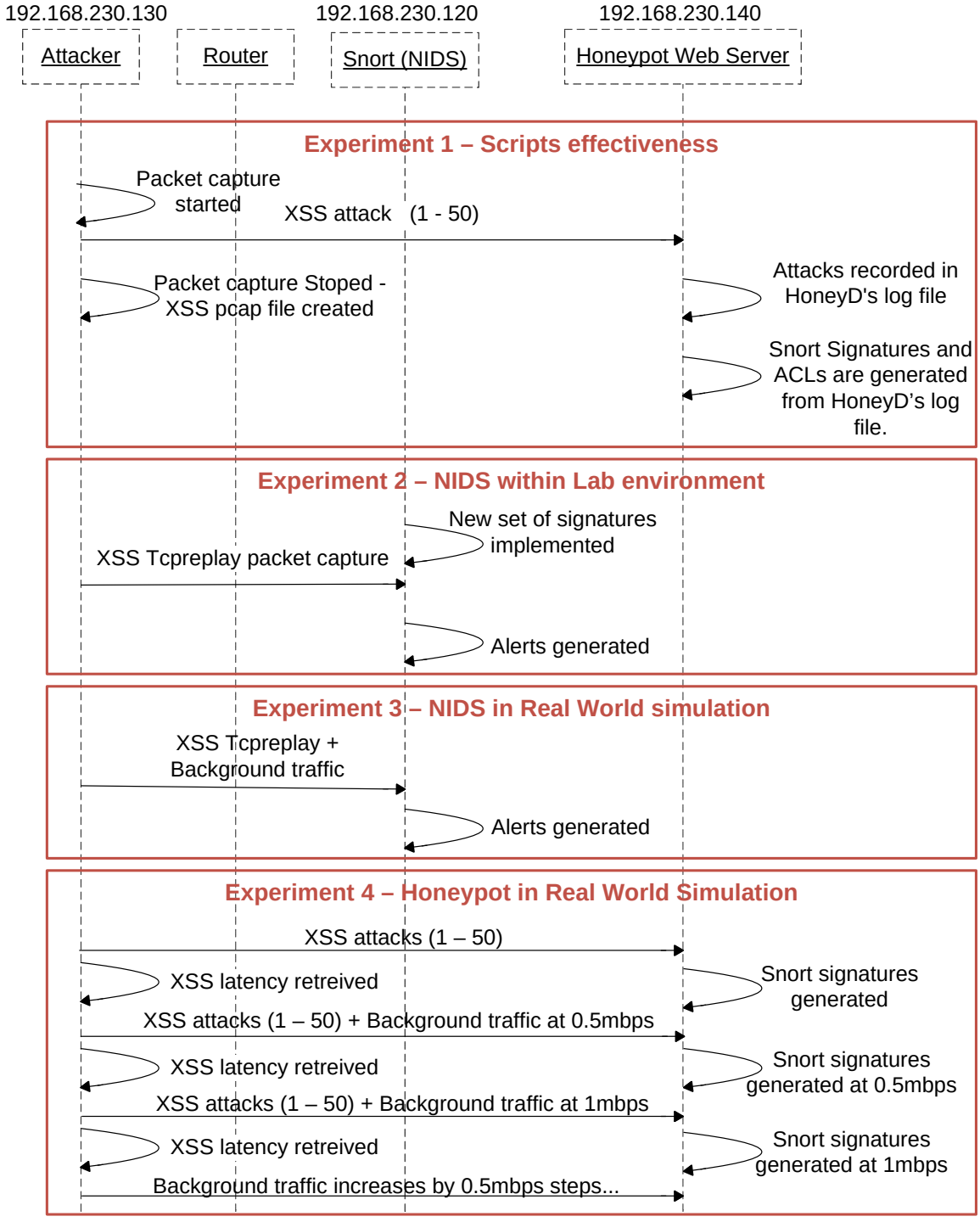


Figure 23 - UML Sequence Experiments model

5.5.1 Scripts effectiveness

The scripts effectiveness experiment involves firstly, to record the network flow while simulating XSS attacks towards the honeypot and secondly, to implement the list of Snort signatures generated by these attacks into Snort.

To record the network flow between the attacker and the honeypot, Wireshark is listening on the attacker's network interface, which is used to send these attacks. The Wireshark packet capture needs to be started before the first experiment and stopped when all the attacks have been sent. The packet capture will be saved in "honeyd_attack.pcap".

To start the honeypot, the following command explained in the section 5.3.1.4, is typed in the command prompt:

```
honeypot@bt:/usr/local/share/honeyd# sudo honeyd -d -i eth1 -f
myconfig.conf 192.168.230.140
```

The XSS attacks list injection is realised by using the "wget" command, which allows to send HTTP requests. The HTTP requests can be extracted from the file "XSS_attack.txt" using the '-I" option:

```
attacker@bt:~# wget -i ~/Desktop/XSS_attack.txt
```

The following HTTP request containing XSS attacks will be sent to the web server:

```
http://192.168.230.140/NEWS/?act=VIEW&ano=958&pst=-
1&startno=1121&pageno=9&FID=
%22%3E%3Cscript%3Ealert%28%2Fwww.r3t.n3t.nl%2F%29%3C%2Fscript%3E

http://192.168.230.140/gp/change-password/-
%22%3E%3Cscript%3Ealert%28document.cookie %29%3C/script%3E-.html

http://192.168.230.140/mailto?prop=movies&locale=us&url=&title=irsdl
"STYLE="width:expression(alert(/Internet Explorer Only-Irsdl is here
again- BugReport.ir andSoorush.SecProject.com/));
     ...
```

The honeypot will analyse these URL strings and generate Snort signatures/ACLs when a string is composed of "<script></script>" tags (or hex-encoded equivalent). The result of this generation will be displayed in the file snort.rules and ACL.log. Once all the attacks have been sent, the packet capture "honeyd_attack.pcap" is stopped and saved, as seen Figure 24.

Source	Destination	Protocol	Info
192.168.230.130	192.168.230.140	HTTP	GET /%22%3E%3C/script%3E%3Cscript%3Ealert
192.168.230.130	192.168.230.140	HTTP	GET /%22%3E%3Cscript%3Ealert%28%2Fwww.r3t
192.168.230.130	192.168.230.140	HTTP	GET /%22%3E%3Cscript%3Ealert(/This-XSS-do
192.168.230.130	192.168.230.140	HTTP	GET /%22%3E%3Cscript%3Ealert(document.coo
192.168.230.130	192.168.230.140	HTTP	GET /%3C%22%3C%3C%22%3C%3CsCrIPT%3Ealert(
192.168.230.130	192.168.230.140	HTTP	GET /%3D%3C/noscript%3E%3Cscript%3Ealert(
192.168.230.130	192.168.230.140	HTTP	GET /%3E%3Cscript%3Ealert(/xss/)%3C/scrip
192.168.230.130	192.168.230.140	HTTP	GET /2%22%3E%3Cscript%3Ealert(document.co
192.168.230.130	192.168.230.140	HTTP	GET /<IMG SRC=\"jav& HTTP/1.0

Figure 24 - Wireshark capture honeyd_attack.pcap

5.5.2 Snort within Lab environment

The lab environment Snort experiment involves replaying the previous packet capture with a different destination IP address (Snort). To do so, the Tcpreplay suite is required to modify the "honeyd_attack.pcap" file. This suite is composed of 3 main tools, Tcpprep, Tcprewrite and Tcpprep.

Firstly, the Tcpprep tool is used to create a cache file “attack.cache”. This cache file splits the packet capture of “honeyd_attack.pcap” into two sides (client/server). This is done to avoid Tcpreplay to do any calculation, which result into having a higher packet rate.

```
attacker@bt:~# tcpprep --auto=bridge --pcap=honeyd_attack.pcap --
cachefile=attack.cache
```

Secondly, the Tcprewrite tool is used to modify the end points IP addresses. Indeed, in the first experiment the traffic was sent from the Attacker to the Honeypot. In this experiment the traffic will be sent from the Attacker to the NIDS. Therefore the destination address needs to be modified by using the –endpoints option. Additionally, the cache file “attack.cache” and input file “honeyd_attack.pcap” need to be provided. The modified file is saved using the –-outfile option with the name “snort_attack.pcap”.

```
attacker@bt:~# tcprewrite --endpoints=192.168.230.130:
192.168.230.120 --cachefile= attack.cache
--infile=honeyd_attack.pcap --outfile=snort_attack.pcap
–-skipbroadcast
```

Finally, the Tcpreplay tool is used to replay an entire communication saved into the newly created “snort_attack.pcap”. Additional options such as the network interface used --intf1 and a low output speed –mbps are also defined.

```
attacker@bt:~# tcpreplay –-mbps=10 --intf1=eth1 snort_attack.pcap
```

5.5.3 Snort in Real World simulation

The Snort real world simulation consists in adding genuine background traffic to the previous experiment. To simulate this background traffic, the packet capture “background.pcap” retrieved from the MIT needs to be modified with the Tcpreplay suite. This is similar to the previous experiment: A cache file is created for “background.pcap”, the endpoints of the network flow are modified and a new packet capture “snort_background.pcap” is created.

```
attacker@bt:~# tcpprep --auto=bridge --pcap=background.pcap --
cachefile=background.cache
attacker@bt:~# tcprewrite --endpoints=192.168.230.130:
192.168.230.120 --cachefile= background.cache --
infile=background.pcap --outfile=snort_background.pcap –-
skipbroadcast
```

The background packet file is now configured and ready to be sent on the network. To mix up both packet captures, two command prompts need to be opened on the attacker instance to type both commands at the same time.

```
#Command prompt1
attacker@bt:~# tcpreplay –-mbps=10 --intf1=eth0
snort_background.pcap

#Command prompt2
attacker@bt:~# tcpreplay –-mbps=10 --intf1=eth0 snort_attack.pcap
```

5.5.4 HoneyD in Real World simulation

This last experiment is similar to the previous one but involves the honeypot effectiveness throughout multiple network speed. The background traffic is sent at different speed from 0mbps by step of 0.5mbps to the honeypot. In parallel, the XSS attacks are injected in a different terminal. The background packet capture is modified with Tcprewrite to change the IP addresses with the one of the Attacker and the Webserver:

```
attacker@bt:~# tcprewrite --endpoints=192.168.230.130:
192.168.230.140 --cachefile= background.cache
--infile=background.pcap --outfile=honeyd_background.pcap
–-skipbroadcast
```

Finally, two command prompts are started, one for the background traffic and one for the XSS injection. Experiments are repeated with background traffic sent at 0.5mbps steps, as seen below:

```
                    0mbps background traffic
#Command prompt1
attacker@bt:~# (0mbps background traffic = no background traffic)

#Command prompt2
attacker@bt:~# wget -i ~/Desktop/XSS_attack.txt
      ... Done

                  0.5mbps background traffic
#Command prompt1
attacker@bt:~# tcpreplay –-mbps=0.5 –-loop 20 --intf1=eth0
honeyd_background.pcap

#Command prompt2
attacker@bt:~# wget -i ~/Desktop/XSS_attack.txt
      ... Done

                    1mbps background traffic
#Command prompt1
attacker@bt:~# tcpreplay –-mbps=1 –-loop 20 --intf1=eth0
honeyd_background.pcap

#Command prompt2
attacker@bt:~# wget -i ~/Desktop/XSS_attack.txt
      ... Done
...
```

The injection time of the 50 XSS attacks is recorded at each 0.5mbps steps. Moreover, the number of signatures generated by Snort signature script is counted and then deleted for each steps

5.6 Conclusion

In this chapter, two Bash scripts have been created and implemented. They gave the possibility to detect XSS attacks and generate ACLs/Snort signatures based on the log file of HoneyD. Additionally, a third Bash script was created to automatically send and implement the Snort signatures generated by the honeypot, into the NIDS "Snort". Therefore, this chapter demonstrates that the generation and implementation of Snort signatures can be fully automated with the help of Bash scripts.

Three instances have been created on the Cloud Computing of Edinburgh Napier University to implement this project. This chapter has shown, step by step, the commands typed in the command prompts of each instance, to install and configure the packages needed for the execution of the experiments. A set of 50 random XSS attacks have been collected from the XXSed.com website and used during the experiments. This gives a large panel of different XSS attacks that will hopefully be detected by the honeypot system.

The experiments are complete and the results have been collected. The next chapter will analyse the results and discuss the effectiveness of the Bash scripts in detecting XSS attacks through HoneyD's log. Furthermore, the effectiveness of Snort and HoneyD to detect these XSS attacks will be tested through injection of Background traffic.

Chapter 6. Evaluation

6.1 Introduction

The results collected during the previous experiments are going to be analysed in this chapter. The results will be compared to assess the effectiveness of the prototype created. This effectiveness will be assessed by calculating the True Positive Ratio and the False Positive Ratio of each experiment. In addition, the breaking point of the honeypot system will be determined by injecting background traffic at increasing speed. This breaking point will indicate the moment where the honeypot becomes unstable and stop detecting XSS attacks. Finally, all the results will be compared against the expected results written in the chapter Design 4.5, to determine whether the behaviours of the HoneyD were as expected or not.

6.2 Scripts effectiveness

The script effectiveness experiment consists in sending XSS attacks towards the honeypots in order to detect them and generate Snort signatures by using the Snort script. In addition, this experiment records the network flow of the attack in order to replay it in the NIDS experiments.

XSS attacks have been sent from the attacker to the honeypot Web server. To evaluate the number of XSS attacks detected by the honeypot's Bash scripts, the number of signatures that have been automatically generated in the "snort.rule" file will be counted.

The whole XSS injection was composed of 50 random XSS attacks and out of them, "snort.rule" shows that 26 signatures have been generated. This means that 26 XSS attacks out of 50 have been detected. The following gives a quick overview of these Snort signatures but the whole list can be found in Appendix E. :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"alert(document.cookie)"; nocase; sid: 1000000;)


alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"a=eval;b=alert;a(b(/XSS/.source));%3C/script37%22%3%3Cmarqu
ee%3E%3Ch1%3EXSS%20by%20Xylitol%3C/h1%3E%3C/marquee%3E"; nocase;
sid: 1000001;)


alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"alert("DaiMon")"; nocase; sid: 1000002;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"ipt>alert('XSS');</scr"; nocase; sid: 1000003;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"alert%28%2FXSS+By+RedTuninG%2F%29%3B"; nocase; sid:
1000004;)
```

In addition an ACL has been generated to block the traffic coming from the attacker IP address:

```
200 deny tcp host  any 192.168.230.130
```

To evaluate the effectiveness of the signatures prototype in the lab environment, the true positive ratio, explained in the chapter Technologies Review section 2.3.3, is calculated

Lab Prototype True Positive Ratio

$$\mathrm{TPR} = \frac{TP}{TP + FN} * 100$$

$$\mathrm{TPR} = \frac{26}{26 + 24} * 100$$

$$\mathrm{TPR} = 52\%$$

The True Positive ratio is a way of saying how good the honeypot is at alerting on real attacks. The chapter Technologies Review section 2.3.3 states that an acceptable level of True Positive alerts should be at least 60%. In this experiment, the True Positive ratio is equal to 52%, which makes it 8% under the acceptable level.

6.3 Snort within Lab environment

The Snort lab experiment implements the newly generated signatures into the NIDS Snort and replays the XSS attack packet towards the NIDS.

During Snort initialization an error is printed. The error message specifies that punctuation errors are present in the newly generated signature file “snort.rule”. After analysing “snort.rule”, errors are found to originate from additional semicolons within the “msg” and “content” fields. In Snort signatures, semicolons should only be used to separate options. However, the strings used in the XSS attacks were directly inserted into the signatures template. In some cases, these strings were composed of semicolons, which were misunderstood by Snort and were generating errors. To mitigate those errors, backslashes were typed before these specific semicolons in order to make them become some no special characters. The modifications are shown below:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected "; flow:to_server,established;
content:"alert(String.fromCharCode(88,83,83,32,98,121,32,66,117,103,
66,117,115,116,101,114,32,45,32,76,101,105,97,32,98,117,103,98,117,1
15,116,101,114,46,99,111,109,46,98,114))\;"; nocase; sid: 1000013;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg XSS attempt:
script injection detected";flow:to_server,established;
content:"a=eval\;b=alert\;a(b(/XSS/.source))\;%3C/script37%22%3%3Cma
rquee%3E%3Ch1%3EXSS%20by%20Xylitol%3C/h1%3E%3C/marquee%3E"; nocase;
sid: 1000014;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected ";flow:to_server,established;
content:"ipt>alert('XSS')\;</scr"; nocase; sid: 1000015;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:" XSS attempt:
script injection detected ";flow:to_server,established;
content:"&to=extarea%20cols=1000%20rows=1000%20style=%22position:%20
absolute\;%20top:%200px\;%20left:%200px\;%22%20onmouseover=%22alert%
28%27Hacked%20By%20B34TR1B0X%20For%20LeaderHackers.ORG%27%2";
nocase; sid: 1000016;)
```

After modification of the previous signatures, Snort initializes correctly and the XSS packet capture is sent from the Attacker to the NIDS. Once the 678 packets that compose the XSS packet capture are sent, Snort displays that 36 alerts have been generated, as seen Figure 25:

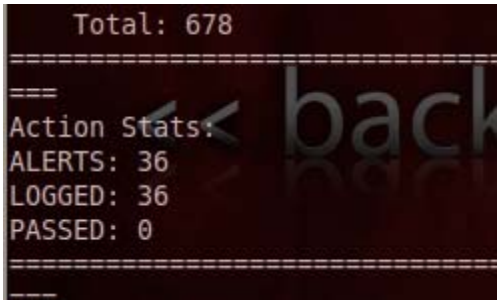


Figure 25 - Snort Lab results

Surprisingly, Snort detected 36 XSS attempt while the honeypot generated signatures for only 26 XSS attacks. This result is surprising because the same number of attack should have been found by the honeypot and by Snort. To understand what happened, the Snort's log file is analysed using Wireshark. This log shows that four TCP out-of-order messages were generated in the XSS packet capture, which resulted in duplicating four of the XSS attacks, as seen Figure 26.

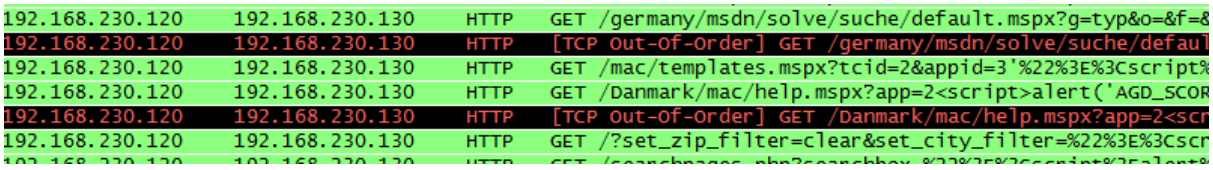


Figure 26 - Snort log file lab

These four duplicated messages were causing four additional alerts and consequently, they should be removed from the total alerts.

36 alerts – 4 duplicated alerts = 32 alerts.

However, there are still a difference between the 26 signatures generated by the honeypot and the 32 XSS attempt raised by Snort. A second analyse of Snort's log file was done, looking for other duplicated alerts, but none of them were found: each alert corresponds to a unique XSS attack string. This fact leads to only one explanation: some Snort signatures are detecting multiple attacks. This is a possibility because some XSS attacks could be using the same strings between the script tags for multiple websites. The script that generates Snort signature is built in a way to not duplicate signatures and therefore, one signature could lead in detecting multiple XSS attacks. Finally, those 32 alerts are True Positive and correspond to unique XSS attacks. The TPR ratio of this experiment is calculated below:

Lab NIDS True Positive Ratio
$$\text{TPR} = \frac{TP}{TP + FN} * 100$$ $$\text{TPR} = \frac{32}{32 + 18} * 100$$ $$\text{TPR} = 64\%$$

The chapter Technologies Review section 2.3.3 states that an acceptable level of True Positive alerts should be at least 60%. In this experiment, the True Positive ratio is equal to 64%, which makes it 4% above the acceptable level and 12% above the previous experiment. However, the TPR calculated in the previous experiment is wrong because it has been calculated on the number of signatures generated without taking in consideration that one signature could detect multiple XSS attacks. Based on that fact, the Scripts effectiveness TPR result should be of 64%, equivalent to this experiment.

6.4 Snort in Real World simulation

The Snort Real world simulation consists in simulating a real world environment by injecting background traffic mixed up with the XSS attack packet. The aim of this experiment is to determine if the background traffic generates False Positive alert results.

During this experiment, the DARPA data set packet has been sent mixed up with the 50 XSS attack packets, which results in a total of 48562 packets. Both packets were sent at a speed of 10mbps. Over this large network traffic, 36 alerts have been generated by Snort, as seen Figure 27:

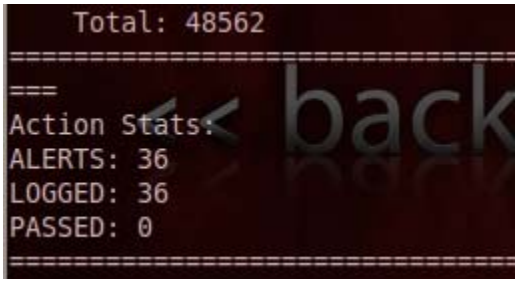


Figure 27 - Real world Snort alerts

This result is similar to the one found previously in the Lab NIDS experiment. The same number of XSS attacks has been sent, which means that results of both experiments are equivalent. This concludes that the injection of background traffic did not raise any False Positive alerts. Therefore using the False Positive Ratio with FP = 0 (no False positive alerts) done:

Lab NIDS False Positive Ratio

$$\mathrm{FPR} = \frac{FP}{FP + TN} * 100$$

$$\mathrm{FPR} = \frac{FP}{FP + (total\ packets - XSS\ packets)} * 100$$

$$\mathrm{FPR} = \frac{0}{0 + (48562 - 678)} * 100$$

$$\mathrm{FPR} = \frac{0}{0 + 47884} * 100$$

$$\mathrm{FPR} = 0\%$$

The False Positive Ratio is 0% because this experiment did not raise any False Positive results. This is a great result because the FPR should always be as close as 0% if possible.

The True Positive Ratio is equivalent to the one calculated in the previous experiment because the same number of True Positive alerts have been detected: 36alerts – 4 duplicated = 32 TP alerts. Therefore the True Positive Ratio is still equal to 64%, meaning 4% above the acceptable level.

6.5 HoneyD in Real World simulation

The HoneyD in real world simulation consists in sending background traffic at different speeds while injecting the 50 XSS attacks into the honeypot. This determines the performances in detecting the attacks and generating Snort signatures in a busy network environment with background traffic. To do so, increasing background traffic speed is sent to the honeypot and two results are collected: the latency to inject the 50 XSS, and the number of signatures generated. Indeed, the latency is printed on the attacker terminal once the 50XSS are successfully injected and the number of signatures generated can be easily found in the file “snort.rule” on the honeypot. The background traffic increases by 0.5mbps steps and latency/signatures generated are collected at each step. The following charts, Figure 28 and Figure 29, have been created with the results retrieved:

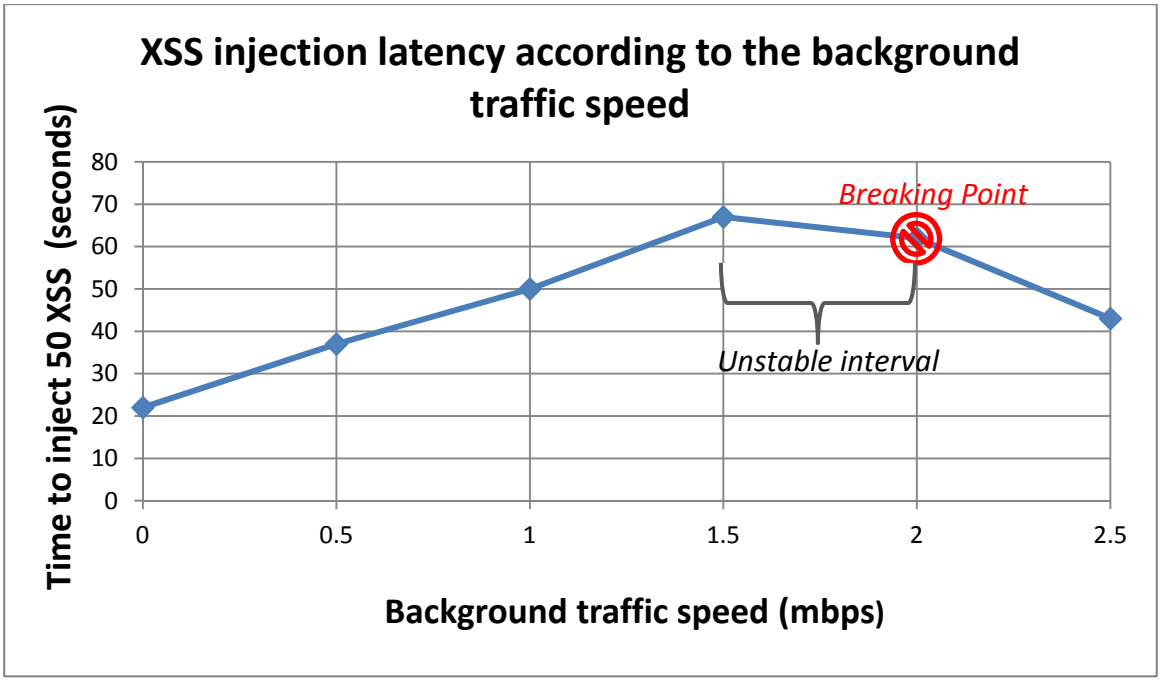


Figure 28 - Chart XSS injection latency according to the background traffic speed

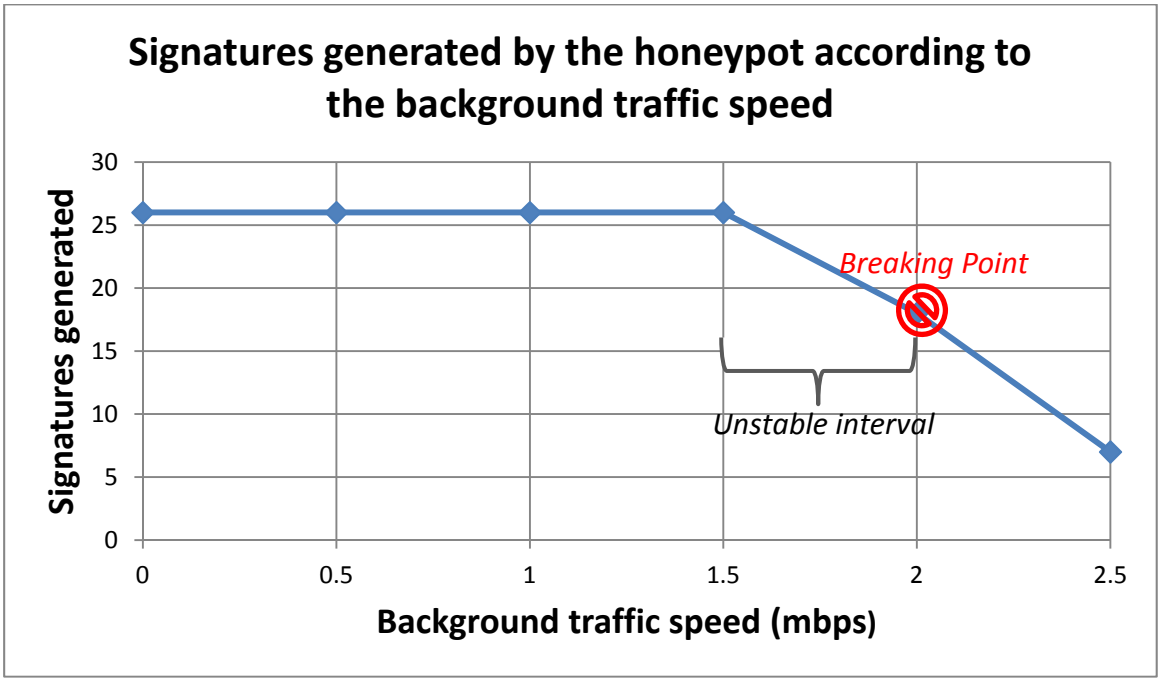


Figure 29 - Chart attacks detected according to the background traffic speed

The results show that when the background traffic speed increases, the injection time of the 50 XSS attacks increases but the number of attacks detected stay similar with 26 signatures generated (equivalent to the first experiment). The increase in the latency shows that the honeypot queue the network packets destined to the webserver. The repercussions of this queuing mechanism during high loads of traffic accrue the honeypot struggle to reply to each single data packets. However, collected results shows that the honeypot never drops

packets and always find the similar number of XSS attacks than in the first experiment (before the breaking period).

However, when the background traffic reaches a speed of 2mbps, the honeypot generates an error and stop itself after certain period of time, as seen Figure 30:

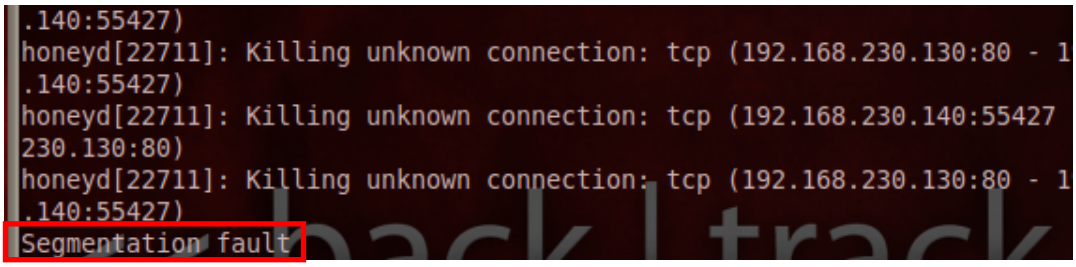


Figure 30 - Error message HoneyD with background traffic at 2mbps

This error called “segmentation fault” is generated because the honeypot is trying to access memory that the CPU cannot physically address. In other words, the honeypot is using all the available memory available to inspect every packet. The segmentation fault error results in sending a dump core signal that terminates the honeypot process.

At 2mbps, the honeypot had the time to detect 18 XSS attacks in 67seconds before segmentation fault error. At 2.5mbps, the honeypot detected 7 XSS attacks in 43 seconds before segmentation fault error. These results are explained in the following way: The XSS injection in this experiment is realised using the command “wget”. This command connects and downloads every pages found in the file “XSS_attack.txt”, which is composed of 50 XSS attacks. This TCP connection involves a 3 way handshake to connect to the webserver and then, the download of the page (error page in this case). Therefore, quite a lot of interaction with the webserver is required when injecting XSS attacks. During high loads of background traffic, this interaction takes an increasing time, because the queuing mechanism used by the webserver can receive multiple background packets between each step of the 3 ways handshake. Consequently, in high loads of traffic, the XSS struggle to be injected (increase the latency) and the honeypot struggle to inspect each packet. This result in a segmentation fault stops the honeypot after a certain period, which explains why the number of signature generation and latency decrease after the breaking point of 2mbps. From this experiment, an unstable interval is determined to be between 1.5 and 2mbps.

6.6 Results analysis compared to expected results

This section analyse the evaluation results in comparison to the expected results that have been supposed in the chapter Design 4.5.

The script effectiveness expected results should have generated a signature for every XSS attacks containing “<script> *random_string* </script>” tags or equivalent tags with the following hexenconded value:

```
“script> random_string script%3E”
”%3Cscript%3E random_string 3C/script%3E”
```

”%3Cscript%3E *random_string* %3C%2Fscript%3E”

The Bash script implemented did a good job in detecting successfully all the values above. However, different “script” tags attack went undetected such as the following attack that inserts % character within the script tags to evade the filtering process:

Hexencoded value:
%252529%25253B%25253C%25252Fscript%25253E
ASCII equivalent:
%%)%%;%%<%%/script%%>

Moreover, some of the XSS attacks were using different tags to inject some JavaScript, such as tags <b> or <img>, and went undetected too.

The NIDS within Lab environment expected results should have shown that the number of alerts raised by Snort is identical to the number of signatures generated by the honeypot. However, during the result analysis, the NIDS detected more XSS attacks that the number of signatures implemented, which had been generated by the honeypot. This surprising result was investigated in the section 6.4. The investigation shows that the honeypot signature generation process always ensure that signatures are not duplicated. Therefore if two attacks have the same attack pattern, only one signature will be generated. This process explains why during the NIDS experiment, Snort was able to detect more attacks than the honeypot had generated signatures: one Snort signatures can detect multiple XSS attacks using the same pattern.

The NIDS in Real World simulation expected results should have been the same that the Lab environment experiment. The result analysis proved that the same number of attacks was detected after the DARPA background traffic injection. This means that the patterns of the signatures generated were specific enough not to raise any False Positive results within background traffic injection.

The honeypot in Real World simulation expected results were that HoneyD should have dropped XSS attacks within high load of background traffics. However, the results analysis showed a different behaviour. During an increasing load of background traffic, HoneyD could detect the same number of XSS attack than previously but HoneyD was taking more and more time to reply to the XSS injection. However, when the background traffic speed reached 2mbps, HoneyD did a “segmentation fault” due to an insufficiency of allocated RAM. This error stopped HoneyD and induce that not all the XSS attacks were detected.

6.7 Conclusion

The results analysed during the evaluation have demonstrated the effectiveness of the Snort signatures generation mechanism invented. The experiments showed that in a Lab and simulated Live environment, the Bash scripts created were able to successfully extract specific information from HoneyD’s log file and to parse it into an ACL or Snort template.

This fulfils a gap found in the literature review, which concerned automating the generation process of Snort signatures.

The injection of 50 random XSS attacks found on the XXSed.com website showed that 26 Snort signatures were generated by the honeypot. During the NIDS lab experiment, the signatures were successfully implemented into Snort and were able to detect 32 XSS attacks. This surprising result was due to the ability of some signatures to detect multiple attacks that used the same pattern. The True Positive Ratio calculation showed that the overall number of detected attacks was 64%, which is 4% above the recommended ratio seen in the chapter Technologies Review 2.3.3. This result did not change during the NIDS Live experiment and proved that the signatures created were specific enough not to trigger any False Positive alarms within background traffic.

Finally, the Snort signature generation was tested by sending XSS attacks and increasing background traffic to the honeypot. HoneyD with the Bash scripts did successfully detect every attack for a background traffic speed up to 1.5mbps. However, at 2mbps the honeypot could not handle the huge traffic load resulting in a segmentation error, due to insufficient RAM. Therefore, a breaking point is identified at 2mbps and an unstable interval is determined between 1.5 and 2mbps.

The next chapter will use these conclusions to determine whether the initial aims of the project have been carried out successfully and to critically evaluate the work achieved in this project.

Chapter 7. Conclusion

7.1 Introduction

The previous chapter, the evaluation, has shown that it is possible to detect XSS attacks and generate ACLs/Snort signatures by using HoneyD's log file. The experiments, based in a cloud environment, show that the signatures are accurate and, when implemented in the NIDS Snort, are able to detect XSS attacks without False Positive results. The main aim of this project was to implement a cloud-based honeypot and use it to automatically detect XSS attacks and generate Snort signatures. Therefore, the main aim of this project has been successfully achieved.

This chapter aims in giving a conclusion of the work that has been carried out during this project. To do so, the initial objectives will be compared to the objectives achieved and a main conclusion will outline the findings. In addition, a critical analysis of the work achieved will assess the work carried out and future work in the same area will be proposed. A grant chart illustrating the project schedule is provided Appendix F. and the initial research proposal in Appendix G.

7.2 Meeting the objectives

This section aims to compare the objectives defined in the introduction chapter and the work that has been carried out. The initial three objectives were the following:

1. Review and investigate the existing literature about security issues related to the cloud computing. In addition, critically evaluate the previous work in generating IDS signatures and compare different honeypot systems. Finally, analyse a specific attacking method and ways to detect it.
2. Design some scripts to generate IDS signatures every time that an attack is detected. Design some cloud-based experiments that will show the effectiveness of the scripts created in detecting attacks and generating IDS signatures.
3. Conduct the final evaluation based on the results collected from the experiments. This evaluation should present determine the effectiveness of the scripts created.

7.2.1 Objective 1

The first objective was met by splitting the literature investigation in two chapters: Technologies Review and Literature Review. The technologies review chapter was used to provide background knowledge about the technologies used throughout the project. Following that, the literature review analysed the literature and compared, in depth, the technologies and the work carried out by previous researchers. Investigations about the

cloud computing security issues, generating IDS signatures, honeypot systems and hacking techniques were undertaken. Furthermore, in order to have an awareness of the laws concerning honeypot implementation, an additional section was created concerning legal issues. Finally, analysis of the features and capabilities of the honeypot "HoneyD" and the NIDS "Snort" were respectively carried out in Appendix A. and Appendix B.

By successfully carrying out the first objective, based on the literature investigation, a gap within the IDS signature generation was identified, an appropriate honeypot system was chosen and detection methods of a hacking technique, known as Cross-Site Scripting (XSS), were examined.

7.2.2 Objective 2

The second objective has been met by creating a Bash script to automatically detect XSS attacks and generate Snort signatures. Additionally, research supervisor Bill Buchanan recommended creating another script to generate some ACLs to block the attacker (Saliou 2005). This work was undertaken by creating a second Bash script. Finally, a third script was created to automatically transfer the signatures generated by the honeypot to the NIDS Snort.

An experiment was designed in order to test the effectiveness of the scripts created to generate Snort signatures/ACLs. In addition, a set of two experiments were designed to test the efficiency of the Snort signatures by implementing them in the NIDS within Lab and Live environment. Finally a last experiment was designed to test the efficiency of the honeypot HoneyD within increasing background traffic. All of these experiments were designed to be implemented within the cloud computing of Edinburgh Napier University. The data collected was kept for further analysis in the next objective.

7.2.3 Objective 3

The third objective was met by conducting an evaluation to determine the effectiveness of the scripts created. This evaluation required the data collected during the implementation. Using this data, the effectiveness of the Snort script created has been evaluated in terms of True Positive Ratio. The results collected from the additional three experiments were also evaluated with a True/False Positive Ratio. Finally, the results of the evaluation were compared with the results expected in the Design chapter. This was done to determine whether the behaviours of the scripts/HoneyD were predictable or not.

7.3 Conclusions

This project has succeeded in creating some Bash scripts able to detect XSS attacks and cause the generation of ACLs/Snort signatures. During the achievement of this project the following findings have been made.

A honeypot system is able to simulate network services and log any interaction with these services. The log file of the honeypot is useful to retrieve a number of important information about intruders, such as IP addresses, ports, timestamps, services and commands typed in. HoneyD, the honeypot studied in this thesis, does not provide any features to filter the information written in the log file. This means that the log file is huge with a lot of information. In order to detect XSS attacks, a Bash script was created to filter specific fields found in the honeypot log file and compare them with well-known tags "<script> *random_strings* </script>" used in XSS attacks. When a match was found, the random strings between the script tags was extracted and injected within a template of a Snort signature. In order to automatize the system a step further, a Bash script was created to automatically transfer the file containing all the generated Snort signatures from the Honeypot into the NIDS.

Moreover, a third Bash script was created to automatically create Access Control Lists. This script extracted every source IP addresses from the honeypot log file and injected them into an ACL template. The honeypot was in the production network, therefore, any interaction with it was classified as malicious activity and an ACL was created to block the attacker. However, ACLs were not implemented into the boundary router and were created just to show the possibility to automatically generate them.

The experiments undertaken in this project showed that the Snort signatures generated by the honeypot, once implemented in Snort, were able to detect 64% of a random set of XSS attacks. By adding background traffic retrieved from the DARPA website, the NIDS was still able to detect successfully the same percentage of XSS attacks. This means that no False Positive Results were generated by the background traffic, which induce that the Snort signatures are specifics enough to only detect XSS attacks. Finally, the last experiment consisted in estimating the honeypot efficiency by injecting XSS attacks and increasing background traffic. Results showed that the latency of the honeypot was increasing according to the background traffic speed. This did not have any effect on the Bash script in detecting XSS attacks and generating Snort signatures. However, a breaking point was identified when the background traffic reaches 2mbps, the honeypot generated an error and stopped. Therefore, an unstable interval was determined between 1.5mbps and 2mbps.

7.4 Critical analysis

Having shown that all objectives of this project have been met and conclusions have been done, this section provides a critical analysis by analysing the limitations of the work carried out. The different methodologies discussed in the literature review will be compared for this purpose.

It must be acknowledged that there are few limitations to the scripts realised. The first limitation concerns the Bash script used to generate Snort signatures. This script was based on detecting the presence of the scripts tags, which seemed to be well-used tags in XSS

attacks, according to (Mookhey, et al., 2010). This detection limited the number of XSS discovered to the number of script tags found out in the honeypot log file. Despite the 64% of alerts generated based on a random set of XSS attacks, this result could have been higher if other tags were implemented in the bash script, such as injection through image field "<img> *Random_string* </img>".

Another limitation is the set of XSS attacks collected from the XXSed.com website which is a database of recent XSS vulnerabilities (XSSed, 2011). This set of attack had a major factor in the experimental results, concerning the detection of scripts tags. However, no selection or filtering was realised over the choice of these XSS attacks. In order to keep a realistic approach the first 50 XSS attacks were used. Therefore, using different XSS attacks would have led to different results.

However, the work carried out in this project contains some strength, able to fulfil gap found during the literature review. The first strength is to automatically generate ACLs and Snort signature based on the attacks detected, which is not found in the existing methodology. Indeed, the Bash scripts created during this thesis reveal to be handy because they do not require any user input.

In addition, this automation is pushed a step further, with a second strength, by creating a Bash script that transfer the snort signatures automatically between the honeypot and the NIDS. This simple script has never been found or thought about in the existing literature and reveals to be really useful in automatically updating the NIDS with new signatures.

7.5 Future Work

The work presented in this project has achieved some requirements that fulfil gaps found in the literature. However, further work in the enhancement of the scripts presented and in additional experiments could be the aim of future projects.

Further work in the detection of attack and generation of Snort signatures could be carried out by using a different type of detection engine. Indeed, the work carried out in this project enables the detection of specific "script" tags. However, as reviewed in the previous section, there are many different type of tags used to inject XSS. Further work in this area could be carried out in order to create a universal detection mechanism able to detect multiple tags.

An additional feature could be to detect different kind of attacks. The investigation undertaken during the literature review showed that the most used hacking technique is SQL injection. By using a similar work scheme and simulating a database by the means of a honeypot, SQL injection could be detected. Indeed, by modifying the detection engine of the Bash Snort_script, in order to detect common SQL injection, could lead to be the subject of a future project.

Another feature could be the creation of a script able to implement automatically in the router the ACLs generated by the honeypot. This would require a similar methodology that the one used to transfer the Snort signatures: a connexion between the honeypot and the

router can be automatically made by using a passwordless SSH connexion. However, there one issue that needs to be taken in consideration. The ACLs should expire after a certain amount of time, because most of the IPs nowadays are attributed automatically and renewed by the Internet Service Provider (ISP) every time the router is shutdown. Therefore, attackers IPs will change and there is no point in blocking too many IPs or it will increase the router's latency.

An additional experiment could be to give an internet access to the honeypot and leave it for a couple of days/weeks to gather data. This type of experiment has not been realised in this project to avoid taking any security risks. However, this experiment could lead in detecting new XSS injection methods and make statistics about the geographical position where the attacks are coming from.

Chapter 8. Works Cited

Adam Kiezun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst. 2011. *Automatic Creation of SQL Injection and Cross-Site Scripting Attacks.* Vancouver, Canada : ICSE'09, 2011. 978-1-4244-3452-7.

Agastya, Harish. 2011. *Security threats to evolving data centers.* s.l. : Trend Micro, 2011.

Allen, Julia. 2004. *Intrusion detection: Implementation and operational issues.* 2004.

Amazon. 2011. Amazon Web Services. [Online] 2011. http://aws.amazon.com/.

Awad Johny, Derdemezis Andreas. 2009. *Honeynets and honeypots: Implementation of a High Interaction Honeynet.* 2009.

Bleikertz, Sören, Schunter, Matthias and Probst, Christian W. 2011. *Security Audits of Multi-tier Virtual Infrastructures in Public Infrastructure Clouds.* s.l. : IBM, 2011.

Bogobowicz, Michael. 2011. What are the main use cases for virtualization? *Quora.* [Online] 01 01 2011. [Cited: 19 09 2011.] http://www.quora.com/What-are-the-main-use-cases-for-virtualization.

Buchanan W, Graves J, Bose N, Macfarlane R, Davison B, Ludwiniak R, 2011. Performance and student perception evaluation of cloud-based virtualised security and digital forensics labs, HEA ICS Conference.

CERT. 2008. CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. *CERT.* [Online] CERT, 2008. [Cited: 27 10 2011.] http://www.cert.org/advisories/CA-2000-02.html.

—. 2007. Denial of Service Attacks. *CERT.* [Online] 2007. [Cited: 20 10 2011.] http://www.cert.org/tech_tips/denial_of_service.html.

—. 2011. Understanding Malicious Content Mitigation for Web Developers. *CERT.* [Online] 2011. [Cited: 20 10 2011.] http://www.cert.org/tech_tips/malicious_code_mitigation.html.

Christian Kreibich, Jon Crowcroft. 2004. *Honeycomb Creating Intrusion Detection Signatures Using Honeypots.* JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom : University of Cambridge Computer Laboratory, 2004.

Christian Seifert, Ian Welch, Peter Komisarczuk. 2006. *Taxonomy of Honeypots.* WELLINGTON : VICTORIA UNIVERSITY OF WELLINGTON, 2006.

Christodorescu, Mihai, et al. 2009. *Cloud Security Is Not (Just) Virtualization Security.* Chicago, Illinois, USA : IBM, 2009. 978-1-60558-784-4/09/11.

Gobel, Jan. 2007. *Amun: A Python Honeypot.* Germany : University of Mannheim, 2007.

Grønland, Vidar Ajaxon. 2006. *Building IDS rules by means of a honeypot.* 2006.

Harmonyguy. 2011. Recent Facebook XSS Attacks Show Increasing Sophistication. *Social Hacking.* [Online] 21 04 2011. [Cited: 27 10 2011.] http://theharmonyguy.com/2011/04/21/recent-facebook-xss-attacks-show-increasing-sophistication/.

Honeynet. 2011. About the Honeynet Project. *The Honeynet project.* [Online] 2011. [Cited: 17 11 2011.] http://www.honeynet.org/about.

honeywall. 2008. Honeywall project. *The honeynet project.* [Online] 24 04 2008. [Cited: 2011 09 27.] https://projects.honeynet.org/honeywall.

Hyang-Ah Kim, Brad Karp. 2004. *Autograph: Toward Automated, DistributedWorm Signature Detection.* s.l. : Carnegie Mellon University, 2004.

J. Levine, R. La Bella, H. Owen. 2003. *The use of honeynets to detect exploited systems across large enterprise networks.* s.l. : IEEE Information Assurance Workshop, 2003.

James Newsome, Brad Karp, Dawn Song. 2005. *Polygraph: Automatically Generating Signatures for Polymorphic Worms.* Carnegie Mellon Univ., Pittsburgh, PA, USA : Security and Privacy, 2005 IEEE Symposium on , 2005. 1081-6011 .

Jarabek, Christopher. 2009. *A Review Of Cloud Computing Security: Virtualization, Side-Channel Attacks, And Management.* Alberta, Canada : University of Calgary, 2009.

Jim, Trevor, Swamy, Nikhil and Hicks, Michael. 2007. *Defeating Script Injection Attacks with Browser-Enforced Embedded Policies.* Alberta, Canada. : ACM, 2007. 978-1-59593-654-.

Joho, Dieter. 2004. *Active Honeypots.* Zurich, Switzerland : s.n., 2004.

Koot, Matthijs. 2007. *Intrusion Detection System - Lesson #4: Honey{pots,nets,walls}.* Amsterdam : s.n., 2007.

Koziol, Jack. 2003. *Intrusion Detection with Snort.* s.l. : Sams, 2003. 978-1-57870-281-7.

Liston, Tom. 2003. Tom Liston talks about LaBrea. *Labrea sourceforge.* [Online] 2003. [Cited: 17 11 2011.] http://labrea.sourceforge.net/Intro-History.html.

McDonald, Kevin. 2011. Tackle your client's security issues with cloud computing in 10 steps . *Search Security Channel.* [Online] 06 2011. [Cited: 23 09 2011.] http://searchsecuritychannel.techtarget.com/tip/Tackle-your-clients-security-issues-with-cloud-computing-in-10-steps.

McMillan, Robert. 2009. Cloud computing a 'security nightmare,' says Cisco CEO. *Computer World.* [Online] 22 04 2009. [Cited: 23 09 2011.] http://www.computerworld.com/s/article/9131998/Cloud_computing_a_security_nightmare_says_Cisco_CEO.

Mishra, Dhanada K. 2004. *International Conference on Controls, Automation & Communication Systems.* s.l. : Allied Publisher Pvt limited, 2004. ISBN 81-7764-726-1.

MIT. 2011. Massachusetts Institute of Technology. *DARPA Intrusion Detection Evaluation.* [Online] 2011. [Cited: 26 10 2011.]

http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999/training/week1/index.html.

Mohammed, M.M.Z.E., et al. 2010. *Accurate signature generation for polymorphic worms using principal component analysis .* Rondebosch, South Africa : Univ. of Cape Town, 2010. 978-1-4244-8863-6 .

Mookhey, K. and Burghate, Nilesh. 2010. Detection of SQL Injection and Cross-site Scripting Attack. *Symantec.* [Online] Symantec, 02 11 2010. [Cited: 27 10 2011.] http://www.symantec.com/connect/articles/detection-sql-injection-and-cross-site-scripting-attacks.

Niels Provos, Thorsten Holz. 2007. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* s.l. : Addison Wesley Professional, 2007.

OWASP. 2011. XSS (Cross Site Scripting) Prevention Cheat Sheet. *OWASP The Open Web Application Security Project.* [Online] 2011. [Cited: 28 10 2011.] https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet.

Paul Baecher, Markus Koetter,ThorstenHolz, Maximillian Dornseif. 2006. *The Nepenthes Platform: An Efficient Approach to Collect Malware.* Mannheim : s.n., 2006.

Perilli, Wendy. 2009. The Benefits of Virtualization and Cloud Computing. *Virtualization Journal.* [Online] 10 03 2009. [Cited: 19 09 2011.] http://virtualization.sys-con.com/node/870217.

Peter Mell, Timothy Grance. 2011. *The NIST Definition of Cloud computing.* Gaithersburg : s.n., 2011. 800-145.

Provos, Neil. 2007. *Developments of the Honeyd Virtual Honeypot.* 2007.

Provos, Niels and Holz, Thorsten. 2007. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* s.l. : Addison Wesley Professional, 2007. 0-321-33632-1.

Provos, Niels. 2007. Service Scripts. *HoneyD.* [Online] 2007. [Cited: 02 11 2011.] http://www.honeyd.org/contrib.php.

R. Perdisci, Dagon D., Wenke Lee, Fogla P., Sharif M. 2006 . *Misleading worm signature generators using deliberate noise injection.* Berkeley/Oakland, CA : Security and Privacy, 2006 IEEE Symposium, 2006 . 1081-6011 .

Radcliffe, Jerome. 2007. *CyberLaw 101: A primer on US laws related to honeypot deployment.* s.l. : SANS Institute, 2007.

Ried, Stefan and Kisker, Holger. April 2011. *Sizing The Cloud - Understanding And Quantifying The Future Of Cloud Computing .* s.l. : Forester research, April 2011.

Ristenpart, Thomas, et al. 2009. *Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds.* Chicago, Illinois, USA : CCS'09, 2009. 978-1-60558-352-5/09/11.

Saliou L, William J Buchanan, James Graves, Jose Munoz. 2005. Novel framework for automated security abstraction, modelling, implementation and verification, ECIW, pp 303-312

Scsami. 2011. Hypervisor. *Wikipedia.* [Online] 07 2011. [Cited: 16 11 2011.] http://en.wikipedia.org/wiki/Hypervisor.

Singh, Amit. 2004. An Introduction to Virtualization. *Kernelthread.* [Online] January 2004. [Cited: 19 09 2011.] http://www.kernelthread.com/publications/virtualization/.

Snort. 2011. *Snort User Manual 2.9.1.* s.l. : Snort, 2011.

Spitzner, L. 2002. *Honeypots: Tracking Hackers.* s.l. : Pearson Education Inc, 2002.

Spitzner, Lance. 2010. Honeypot Farms. *Symantec.* [Online] 02 11 2010. [Cited: 17 11 2011.] http://www.symantec.com/connect/articles/honeypot-farms.

—. 2010. Honeypots: Are They Illegal? *Symantec.* [Online] 02 11 2010. [Cited: 26 09 2011.] http://www.symantec.com/connect/articles/honeypots-are-they-illegal.

Sumeet Singh, Cristian Estan, George Varghese, Stefan Savage. 2004. *Automated Worm Fingerprinting.* San Diego : University of California, 2004.

Sun, Changhua, et al. 2009. *A More Accurate Scheme to Detect SYN Flood Attacks.* Rio de Janeiro : IEEE, 2009. 978-1-4244-3968-3.

Tang, Xinyu. 2011. *The Generation of Attack Signatures Based on Virtual Honeypots.* Wuhan, Hubei China : s.n., 2011. 978-0-7695-4287-4.

Timm, Kevin. 2010. Strategies to Reduce False Positives and False Negatives in NIDS. *Symantec.* [Online] 03 11 2010. [Cited: 15 09 2011.] http://www.symantec.com/connect/articles/strategies-reduce-false-positives-and-false-negatives-nids.

Vanover, Rick. 2009. Type 1 and Type 2 Hypervisors Explained. *Virtualization Review.* [Online] 24 06 2009. [Cited: 19 09 2011.] http://virtualizationreview.com/blogs/everyday-virtualization/2009/06/type-1-and-type-2-hypervisors-explained.aspx.

Venezia, Paul. 2008. VMware ESX Server 3.5. *Techworld.* [Online] 11 02 2008. [Cited: 16 11 2011.] http://review.techworld.com/virtualisation/599/vmware-esx-server-35/.

Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, Somesh Jha. 2005. *An Architecture for Generating Semantics-Aware Signatures.* Madison : University of Wisconsin, 2005.

VMware. 2011. VMware vSphere Hypervisor™ (ESXi). *VMware.* [Online] VMware Inc., 2011. [Cited: 29 09 2011.] http://www.vmware.com/products/vsphere-hypervisor/overview.html.

WHID. 2011. Web-Hacking-Incident-Database . *WHID.* [Online] 2011. [Cited: 20 10 2011.] http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database.

William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. 2006. *A Classification of SQL Injection Attacks and Countermeasures.* Georgia Institute of Technology : IEEE, 2006.

Wolf. 2010. Security Issues and Solutions in Cloud Computing. *Wolf Halton's Opensource & Security.* [Online] 25 06 2010. [Cited: 23 09 2011.] http://wolfhalton.info/2010/06/25/security-issues-and-solutions-in-cloud-computing/.

XSSed. 2011. XSS archive. *XSS attacks information.* [Online] 2011. [Cited: 17 11 2011.] http://www.xssed.com/.

Appendix A. *Understanding HoneyD*

HoneyD is a low interaction honeypot created by Neil Provos (Provos, 2007) and released under GNU Public license (GPL). It is a small daemon able to simulate a networking stack of virtual hosts. Virtual hosts have specifics preconfigured personalities, which mean that they are associated with unallocated IP addresses and configured to run arbitrary services. Services are generally scripted in Perl, Shell or Python and a couple of them can be found on the author's website. They can simulate different TCP/IP services like HTTP, FTP, Telnet, SMTP, SSH, IIS and many more.

A.1. Features

HoneyD supports many features and is very flexible for simulating networks infrastructures. The following gives a brief overview of the main features supported by HoneyD and come from a book written by HoneyD's creator (Provos, et al., 2007):

- Simulate small to large network topologies: Neil Provos stated that HoneyD is able to simulate up to 65536 virtual hosts with different personalities. Each virtual host is allocated with an un-used IP address, so it is possible to interact with thousands of hosts at the same time.
- Configuration of personalities: Virtual hosts can be associated with multiple services (HTTP, SMTP, FTP, etc.), which aimed to interact with an adversary. Additional HoneyD features include proxy connections to other machines, passive fingerprinting to identify remote hosts and random sampling for load scaling.
- Simulate operating systems at TCP/IP stack level: HoneyD is able to deceive fingerprinting tools like Nmap or Xprobe and make them believe that the virtual honeypot is running specifics operating systems and services. Moreover, policies as fragment reassembly policy and FIN-scan policy can be adjustable in order to increase the realism of the system.
- Configurable network characteristics: The routing topologies can be configured with specifics latency, packet loss and bandwidth characteristics. In addition, HoneyD is able to integrate of physical machines within the virtual environment and to distribute operations of its network topology via GRE tunnelling.

- Subsystem virtualization: With subsystem virtualization, HoneyD can run real UNIX applications (web/mail/FTP servers, etc.) under a virtual IP address controlled by HoneyD.

A.2. Architecture overview

HoneyD's architecture is build using five components: a configuration database, a packet dispatcher, protocol handlers, a personality engines and optional routing components, as seen on Figure 31.

Incoming packets entering the network are routed to the packet dispatcher. The packet dispatcher inspects the checksum of each packet, to detect accidental errors and then inspects the protocol handler. HoneyD supports only ICMP, TCP and UDP, which results in dropping packets for any other protocols. Once the inspection is done, the packet dispatcher query the configuration database to see if the packet's destination IP address corresponds to a specific template (virtual host). When there is no template that corresponds to an IP address, a *default* template is used. Then, the packet is dispatched to its specific handler ICMP, TCP or UDP. By default, ICMP request are replied by a *destination unreachable* message, unless specified in the personality database. TCP and UDP protocols establish connection to arbitrary services. For a TCP datagram, the framework checks the three way handshakes to see if a packet is part of an established connection. If the packet encloses a connection request, the framework creates a new process to emulate the required service. If the packet is part of a previous connection, the framework forwards it to its existing service. For a UDP datagram, there is no possibility to know if a packet is from a previous connection because UDP does not ensure handshaking dialogs. Therefore, the UDP datagram is directly sent to the appropriate service. Finally, before leaving the framework, packets are routed to the personality engine. The personality engine modify the packet's content in order to change the IP source, to make it appears to come from the network stack of the configure OS.

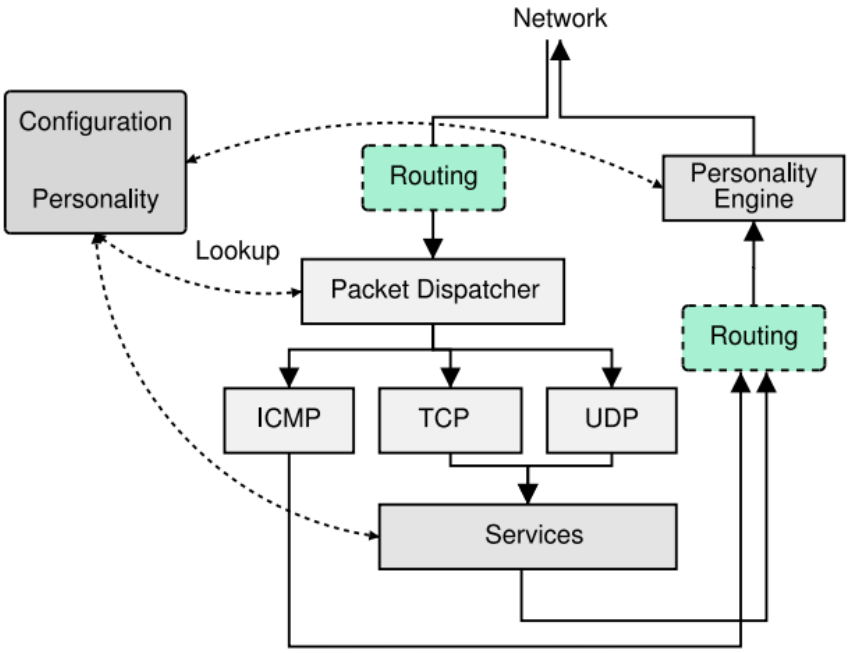


Figure 31 - HoneyD architecture (Provos, 2007)

A.3. Configuration file

HoneyD's configuration file is used to define templates. A template is associated with: a name, a service(s) associated, open port(s) and IP address(es). The configuration file is by default *honeyd.conf* and a simple example of it looks like the following:

```
create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
add windows tcp port 80 "./scripts/iisemul8.pl"
set windows default tcp action reset
bind 192.168.0.10 windows
```

This configuration is used to create a template named '*windows'*. The template personality '*Windows NT 4.0 Server SP5-SP6'* is used to deceive fingerprinting tools by making the system looks like genuine. Open ports are configured using '*add windows tcp port 80'* open the tcp port 80 and associate a script to emulate a Windows Web Sever IIS by using the perl script '*iisemul8.pl'*. Any other TCP query that are not for the port 80 will be reset '*default tcp action reset*'. Finally, to associate this template to an IP address, the keyword '*bind'* is used, followed be the IP address and the template's name.

This configuration file is a simple example of simulating one host with one service. However, HoneyD is much more flexible and can emulate up to 65536 host with multiple services. Additionally, other hosts like routers can be emulated and the routing topology (latency, loss and bandwidth) can be configured in honeyd's configuration file. Therefore, a big network with multiple hosts and services can become extremely complicated to configure.

A.4. Logging capability

HoneyD has capability to provide two different logging modes: packet level and service level.

The packet level logging mode is enable by using the –l flag. This logging mode contains the timestamp of the packet, the protocol used, the source/destination IP addresses and the source/destination ports. Additionally, the letters S (start connection), E (end connection) or – (neither S nor E) display if the connection start, end or part of a previous connection. The last field displays how many bytes have been transmitted by the honeypot and which type of OS was identified. The following output is an example of the packet level logging file:

```
2011-10-07-04:09:30.9342 tcp(6) S 192.168.230.1 3337 192.168.230.132
80 [Windows 2000 RFC1323]
2011-10-07-04:09:31.3346 tcp(6) - 192.168.230.1 3336 192.168.230.132
280: 52 S [Windows 2000 RFC1323]
2011-10-07-04:09:31.9319 tcp(6) - 192.168.230.1 3336 192.168.230.132
280: 48 S [Windows XP SP1]
2011-10-07-04:09:32.0325 tcp(6) E 192.168.230.1 3337 192.168.230.132
80: 315 6149
```

The service level logging mode is enable by using the –s flag. This mode contains the logging outputs of the emulated services (HTTP, FTP, telnet, etc.). It gives detailed information

about the interaction of an attacker with the honeypot. The log file organise each connection attempt between –MARK- and -ENDMARK- tags. The following output is an example of the service level logging file. In that example, the honeypot had the service wu-ftpd.pl installed with an open port 21. The log file shows that an attacker interacted with the virtual FTP server at a specific time using the user anonymous and password neesus@nessus.org. Actually, that was not an attacker but a vulnerability scanner named Nessus. It was used to check if the honeypot was logging information correctly. Once connected, the vulnerability scanner sent a DELE (delete) command. This is just an example to show that every commands typed by an intruder are recorded by the honeypot.

```
--MARK--,"Wed Oct 7 04:36:51 EDT 2011","wu-ftpd/FTP",
"192.168.230.1", "192.168.230.132", 6119,21,
"USER anonymous
PASS nessus@nessus.org
DELE /
",
--ENDMARK—
```

Appendix B. *Understanding Snort*

Snort is a widely use network IDS/IPS (intrusion detection/prevention system) that can take action against specific packets going across a network. Snort is one of the most successful IDS on the market because it is a fast, flexible, lightweight and open-source IDS. Snort's approaches to inspect the traffic combine the benefits of signature, protocol and anomaly-based packets inspection. Additionally, Snort can be used with four different modes:

- Sniffer mode: Snort listens to the network traffic and prints it on to the screen.
- Packet logger: Snort record the network traffic and save it into a file.
- IDS mode: Snort record only the network traffic matching specifics rules. Rules have been previously configured by the administrator.
- IPS mode: Snort takes action against specific network traffic matching the rule file. This network traffic can be logged, dropped or raised an alert.

B.1. Snort components

Snort relies into the following four major components that are each critical to intrusion detection system (Koziol, 2003), as seen Figure 32.

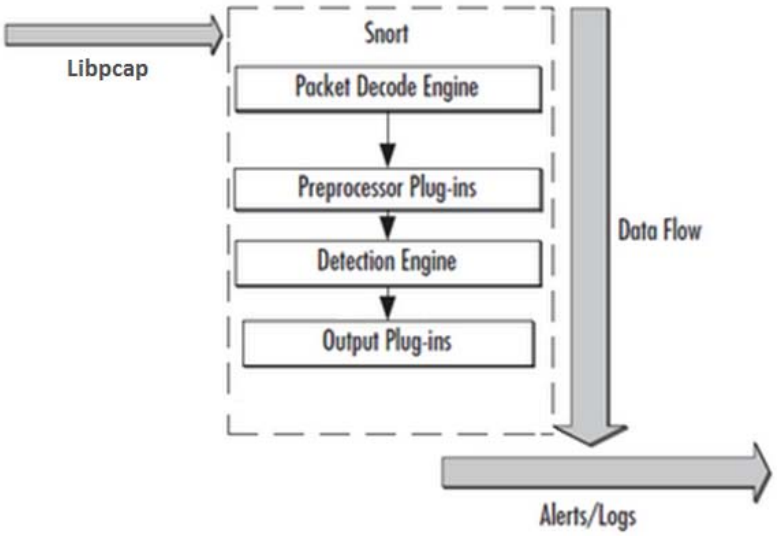


Figure 32 - Snort components overview based on (Snort, 2011)

Snort has no native packet capture facility yet, so in order to collect packets Snort needs an external library named *Libpcap*. This component is used to sniff packets directly from the network interface card and pass them into the second component: *packet decoder*. The Packet decoder component translates specific protocol elements into an internal data structure. Once the packets are translated, they are forwarded to the third component: *pre-processors*. The Pre-processors plugins can be used either to examine packets for suspicious activity or to normalize traffic so that the next component can interpret them. The *Detection Engine* compares the packets to its rule files, in order to builds attack signatures. The alerts raised are sent to the *Output plugins* that is used to dump alerting data to another resource (database, pop-up alert, SNMP center, etc.) or file.

B.2. Snort configuration file

At start up, Snort reads its configuration file *snort.conf* to determine, among other, the rule files' paths. These rules are text-based files generally categorized into different groups; for example, the file *ftp.rules* contains a selection of rules to detect ftp attacks and exploits. The following configuration file can be customized by adding or deleting rule files' paths.

```
##################################################
# Step #9: Customize your Shared Object Snort Rules
  . . .

# include $SO_RULE_PATH/bad-traffic.rules
# include $SO_RULE_PATH/chat.rules
# include $SO_RULE_PATH/dos.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/ftp.rules
```

B.3. Rule format

Each rule has two logical parts: rule header and rule options. The rule header contains criteria for matching a rule against data packets and specifies what action the rule undertakes. The rule options contain a customisable message and additional criteria for matching a rule against data packet. The following rule comes from the file *ftp.rules* and rise an alert with the message "FTP DELE attempt" every time the host receives a packet with the content "DELE" from an established connection and on the FTP port.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP DELE attempt";
flow:to_server,established; content:"DELE"; classtype:attempted-
admin; sid:1975; rev:8;)
```

B.3.1. Rule header

The rule header is the first section of the rule. It shows which action will be taken on which type of packets.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
```

The following section is a quick overview of the syntax used above:

- **Alert**: Action to undertake when the packet matches the rule. Other options are available: log, pass, dynamic and activate.
- **ICMP**: protocol being used. Other options available: UDP, IP and TCP.
- **$EXTERNAL_NET**: keyword for any outside IP address.
- **Any**: source port.
- **->** indicates the direction of the conversation. Other option available for a bi-communication: <>.
- **$HOME_NET**: keyword for the local network.
- **21**: destination port (FTP).

B.3.2. Rule options

Rule options follow the rule header and are enclosed by a pair of parentheses. There can be multiple options inside the parentheses, so to differentiate each one of them, options are separated by a semicolon. An option is composed of a keyword and an argument; they are separated by a colon.

```
(msg:"FTP DELE overflow attempt"; flow:to_server,established;
content:"DELE"; classtype:attempted-admin; sid:1975; rev:8;)
```

The following is an overview of the rule options seen above:

- **Msg**:"FTP DELE overflow attempt": message displayed by the alert.
- **Flow:** to_server,established: Detection plugin that check if the packets is part of an existing session.
- **Content**:”DELE”: Snort tries to match this content against packets received on port 21.
- **Class-type**:attempted-admin: alerts are categorized in multiple class-type. Categories have different priorities in order to ease the user’s reading.
- **Sid**: Snort rule unique identifier. This number is unique for each rule. By convention, local rules (self-made) should be above 1000000. The previous rule is an official rule from the Snort website, therefore the sid is under 1000000.
- **Rev**: Revision number of the rule.

B.4. Standard output alert and logging

Snort output alerts can be raised in diverse ways:

- *Alert_ fast* generates alerts sequently in a one-line file.
- *Alert_full* creates a directory for each IP and fills it with decoded packets traces.
- *Log_tcpdump* logs packets that can be opened by TCPdump.
- *CSV* logs packet into CSV files that can be imported into databases.
- *XML* logs alerts that can be read as a webpage.
- *Alert_syslog* logs alert for syslog servers.

By repeating the previous example, Snort configured in *Alert_fast* will generate the following alert message:

```
[**] [116:1975:8] FTP DELE overflow attempt [**]
```

This alert is composed of [**], three numbers and a message. The [**] is only there to ease the readability by an user. The three numbers correspond in order to the Generator ID (*GID*), Signature ID(*sid*) and Revision ID(*rev*). The *sid* and *rev* have been seen in the previous section and are used to identify a rule. The GID is used to tell the user which Snort’s component generated this alert. Finally, the message corresponds to the one entered in the *msg:””* field option, seen last section.

Appendix C. *Script analysis*

The following provides a line by line analysis of the two main scripts created during this thesis.

C.1. Snort script

```
#SNORT SCRIPT with a line by line analysis
#!/bin/ksh
count=1000000
(
while true
do
#The nawk option is used to look into msg.log (honeypot log file) and
retreive the lines that start with “GET”. Then the cut option, copy the
second argument of this line. The sort and uniq options classify the
lines retreived and delete duplicates. The output is sent in
“content.log”.
nawk -F /^ '/GET/ {print $0}' msg.log | cut -d' ' -f2| sort | uniq  >
content.log
#The grep option is used to copy, from content.log, every lines that
has a “script>” or “script%3E” keyword in it. These lines are then
handled by the sed options, which are used to retreive the content
between the “script” tags (or their hex-encoded equivalents). The
output is sent in “content_script.log”.
grep 'script>\|script%3E \|script%3e' content.log | sed -e
"s/.*<script>//;s/<\/script>.*//" | sed -e
"s/.*%3Cscript%3E//;s/%3C\/script%3E.*//" | sed -e
"s/.*%3Cscript%3E//;s/%3C%2Fscript%3E.*//" > content_script.log
#The cat option is used to print the file content_script.log. The
pipeline is used to associate the cat command with the read comman. The
read command allows that each line will be read one by one and inserted
in the variable “$line”.
cat content_script.log |
(
while read line
do
#Each line of the content_script.log is compared to the file
snort.rules, which contains previous signatures created. If a line
match a signature, it means that the signature for this attack has
already been created. In that case an echo 'Snort signature '$line'
already existing' is done.
res=`grep -c $line snort.rules`
#If no match are found, the variable line is inserted within a Snort
signature template.
      if [ $res == 0 ];
            then echo 'alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:" XSS attempt: script injection detected ";
flow:to_server,established; content:"'$line'"; nocase; sid:
'$count';)'\'>> snort.rules count=`expr $count + 1 `
```

```
#If a line match a signature, it means that the signature for this
attack has already been created. In that case an echo 'Snort signature
'$line' already existing' is done.
            else echo 'Snort signature '$line' already existing'
      fi
done
)
#The script wait 30seconds before starting a new loop.
 sleep 30
done
)
```

C.2. ACL Script

```
#ACL_script.sh
#!/bin/ksh
count=200
(
while true
do
nawk -F"[\",]" '/MARK/ {print $9}' msg.log | cut -d' ' -f1| sort | uniq
| grep -v '^$'> IP_src.log
#The nawk option is used to look into msg.log (honeypot log file) and
retreive the lines that start with “MARK”. Then the cut option, copy
the first argument of this line. The sort and uniq options classify the
lines retreived and delete duplicates. The output is sent in
“IP_src.log”. This command exctract the IP source addresses from the
honeypot log file and paster it into “IP_src.log”.
cat IP_src.log |
(
while read line
do
res=`grep -c $line ACL.log`;
if [ $res == 0 ];
#IPs in “IP_sc.log” and “ACL;log” are compared. If there is no match,
the IP is injected into the ACL template below and printed into
“ACL.log”
      then echo ''$count' deny tcp host '$line' any'\>> ACL.log
      count=`expr $count + 10 `
#If the IPs already exists, the following echo is done.
else echo 'IP match found'
fi
done
)
#The script wait 30seconds and do a loop.
 sleep 30
done
)
```

Appendix D. *Configuration of a password less SSH connection*

The following is a guide to set-up a SSH Client for password-less login to a Server using public-private key certificates.

D.1. Server side (honeypot)

1.1. *Generate public/private dsa key pair*

```
#sshd-generate
```

1.2. *Start the ssh service*

```
#/etc/init.d/ssh start
```

1.3. *Edit the following file*

```
#sudo nano /etc/ssh/sshd_config
```

Make sure these lines are available and not commented:

```
    PubkeyAuthentication yes
    AuthorizedKeysFile %h/.ssh/authorized_keys
```

D.2. Client side (Snort)

2.1. *Edit the following file*

```
    #sudo nano /etc/ssh/ssh_config
Make sure these lines are available and not commented:
    IdentityFile ~/.ssh/identity
    IdentityFile ~/.ssh/id_rsa
    IdentityFile ~/.ssh/id_dsa
```

2.2. *Login into SSH server*

```
    ssh 192.168.230.140
```

Issue the following command:

```
    ssh-keygen -t dsa
```

2.3. *Copy the public key into target SSH server*

```
    sudo ssh-copy-id -i ~/.ssh/id_dsa.pub ben@192.168.230.140
    Choose username as ben
```

A SSH address is generated:

```
    ben@192.168.230.140
```

2.4. *Passwordless connection created*

Now that the keys have been exchanged, a SSH passwordless connection is possible using the following command:

```
    ssh ben@192.168.230.140
```

Appendix E. *Snort Signatures Created*

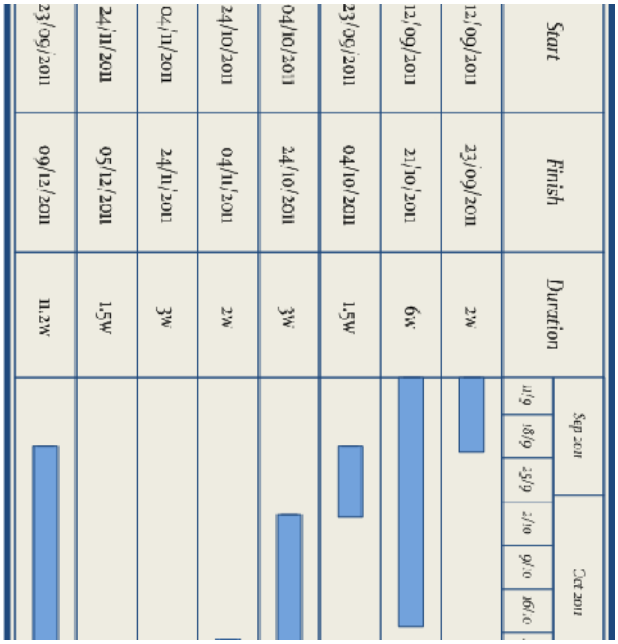
The following shows the Snort signatures created by the Bash script "snort.sh" after the injection of 50 XSS attacks. The 50 XSS are provided in the DVD attached to this thesis.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(document.cookie)"; nocase; sid: 1000000;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert(%27test%27)%3B";
nocase; sid: 1000001;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(%22BlackHacker[dot]Coder[at]Gmail[dot]com%22)"; nocase;
sid: 1000002;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert%28%2Fwww.r3t.n3t.nl%2F%29"; nocase; sid: 1000003;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert(/This-XSS-doesnt-
work-all-the-time/)"; nocase; sid: 1000004;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert('BugReport.ir";
nocase; sid: 1000005;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(%27HacKSever%27)"; nocase; sid: 1000006;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert(/xss/)"; nocase;
sid: 1000007;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert("I'm%20Back%20-
%20rstcenter.com%20-%20hackersblog.org")"; nocase; sid: 1000008;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert(1337)"; nocase;
sid: 1000009;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(/XSS%20by%20Fugitif/)"; nocase; sid: 1000010;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert(1)"; nocase; sid:
1000011;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert('AGD_SCORP')</Script>"; nocase; sid: 1000012;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(String.fromCharCode(88,83,83,32,98,121,32,66,117,103,66,
117,115,116,101,114,32,45,32,76,101,105,97,32,98,117,103,98,117,115,116
,101,114,46,99,111,109,46,98,114));"; nocase; sid: 1000013;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert('Agd_Scorp')";
nocase; sid: 1000014;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert%28document.cookie%29"; nocase; sid: 1000015;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"/hk/presspass/chinese/result.aspx?fm=01/01/2006T12/14/2006%22%
3e%3c/script%3e%3c&to=extarea%20cols=1000%20rows=1000%20style=%22positi
on:%20absolute;%20top:%200px;%20left:%200px;%22%20onmouseover=%22alert%
28%27Hacked%20By%20B34TR1B0X%20For%20LeaderHackers.ORG%27%2"; nocase;
sid: 1000016;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(/www.r3t.n3t.nl/)"; nocase; sid: 1000017;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(%22AGD_SCORP%20xD%22)%3C/Script%3E"; nocase; sid:
1000018;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"a=eval;b=alert;a(b(/XSS/.source));%3C/script37%22%3%3Cmarquee%
3E%3Ch1%3EXSS%20by%20Xylitol%3C/h1%3E%3C/marquee%3E"; nocase; sid:
1000019;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established; content:"alert("DaiMon")";
nocase; sid: 1000020;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert(%22BlackHacker[dot]Coder[at]Gmail[dot]com%22)"; nocase;
sid: 1000021;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"ipt>alert('XSS');</scr"; nocase; sid: 1000022;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"alert%28%2FXSS+By+RedTuninG%2F%29%3B"; nocase; sid: 1000023;)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"XSS attempt
detected"; flow:to_server,established;
content:"eval%28String.fromCharCode%2897%2C108%2C101%2C114%2C116%2C40%2
C34%2C88%2C83%2C83%2C32%2C102%2C111%2C117%2C110%2C100%2C32%2C98%2C121%2
C32%2C83%2C109%2C111%2C107%2C101%2C121%2C32%2C79%2C102%2C32%2C68%2C97%2
C114%2C107%2C99%2C48%2C100%2C101%2C32%2C72%2C97%2C99%2C107%2C99%2C105%2
C110%2C103%2C32%2C67%2C114%2C101%2C119%2C34%2C41%2C59%29%29—"; nocase;
sid: 1000024;)
alert tcp $EXTERNAL_NET any -> $HOMEET any (msg:"XSS attempt detected";
flow:to_server,established;
content:"alert%28document.cookie%29%3C%2fscript%3E&sa=N&tab=wb";
nocase; sid: 1000025;)
```

Appendix F. *Gantt Chart*

This Gantt Chart has been made at the beginning of the project to organise the work schedule of each chapter. In addition, an extra week was planned at the end of the project for proof reading and printing. During the project, this schedule has been followed and extra work was done over weekends and evenings when it was required. But this has been efficient and the writing of this thesis was finished two weeks before the hand in date, which gave more than enough time to show it to Professor Bill Buchanan, do minor updates, proof read and print it.



Appendix G. *Research Proposal*

G.1. Brief description of the research area - background

At the beginning of the academic year 2010/2011, Napier University has started a migration to the cloud computing using the platform VMware ESXi. The actual success of the cloud computing is due to the cost saving of this technology which reduce the hardware investment, save space and save power consumption (Georgieva, 2009).The cloud computing uses virtualisation and clustering in order to optimise the server utilisation. The platform VMware ESXi is installed directly on a physical stack (server) and gives the possibility to create/manage multiples Virtual Machines (V.M) which share the physical resources of the server. Instead of using expensive physical machine when a high-end hardware is needed, it is now possible to create the same VM within seconds and give it more resources on the fly if necessary. The benefit from the students' perspective of Napier University is an opportunity to manage their own virtual environment through any web browser, giving them the ability to complete practical lessons from any computer, anywhere in the world.

According to Daniel Petri, using virtualisation in cloud computing brings some new security issues, as virtualisation does not provide any more security and a Virtual Machine (V.M.) could be even less secure than a phisical machine (Petri, 2009). The addition of VMs is so fast and easily done that administrator sometimes forget to add them to the patch distribution system. These unpatched VM become quickly targeted by worms/intruders and result in using more network bandwidth and resources that usually needed. Moreover, Daniel Petri states multiple security issues in virtualisation as a VM infected by a virus could potentially infect another VM or even infect the hypervisor layer, which could result as a complete failure of the VM cloud.

Traditional networks use IDS (Intrusion Detection System) to detect malicious data packets. There are two types of IDS: anomaly-based and signature-based. This thesis covers signature based IDS. It means that the IDS detects an attack using a set of signatures which are periodically updated. However, signature-based IDSs lack to discover new attacks that are not included within the signature database.

In order to detect potential new attacks, this thesis is based on the implementation of a trap known as “honeypot”. Honeypots are not defensive security systems as IDS and Firewalls as they do not tend to protect the network but lean towards attracting the intruder. According to L. Spitzner a honeypot is *an information system resource whose value lies in unauthorized of illicit use of that resource* (Spitzner, 2003). Honeypots are fake information servers able to run multiple of services (FTP, SQL, SSH…) that are configured using weak security mechanisms, making them highly interesting for an intruder in search for a target. These traps are loaded with monitoring and tracking tools which make them able to log any activity resulting from an illicit access.

The aim of this project is to use the honeypot in order to create a new set of signatures for the IDS and compare the efficiency with the old ones.

G.2. Project outline for the work that you propose to complete

The idea for this research arose from:

A personal interest developed during the Honours project dissertation “Network- and Host Detection System of Botnets” where my researches aimed to the development, implementation and detection of own-made Botnet within virtual environment. During that project, I went through some white paper about the detection of Botnets using Honeypots systems. It seemed fascinating and made me want to know more about it.

Last year during the Honours degree, the cloud computing has been introduced to me in the Advanced Security and Network Forensics module. It really caught my interest and I was amazed about the aspect of virtualisation and clustering. Moreover, a conference in Napier University by Tabassum Sharif on the 9th March 2011 about the future of the Cloud computing made me realise that it will play an important role in the future and therefore that I should

study more about it.

I had my two ideas, network security and cloud computing, so I decided to mix them together and to work on my Master dissertation about the network security within the cloud computing based on the implementation of a honeypot.

The aims of the project are as follows:

Review and investigate the existing literature

- The literature review covers the VMware ESXi platform, Intrusion Detection Systems, Honeypot systems and Security/Vulnerability scanner.
- Design and implement a honeypot system
- The honeypot implementation aims to create a virtual network topology. This topology includes virtual routers and virtual hosts with different OS and services.
- Generate new signatures for the IDS
- A security scanner is going to attack the honeypot and try to exploit the different vulnerabilities previously configured. The honeypot is going to log those attacks and generate new signature for the IDS.
- Compare the new signatures
- The new signatures are going to be compared with the Snort basic ones to determine if they are more efficient in terms of True/False positive.

The main research questions that this work will address include:

- Which type of honeypot interaction is more adapted for a secure implementation?
- How to translate packets captured by the honeypot into an IDS signature?
- Are the signatures generated more efficient than traditional ones?

The software development/design work/other deliverable of the project will be:

Honeypot implementation within the cloud of Napier University.

The project will involve the following research/field work/experimentation/evaluation:

- Literature search and review

The literature research will be done through the search Engine of Napier University NUIN-link which is really helpful to have a look in multiple databases at the same time. Deeper research will be done on the databases “Science Direct” and “IEEE” which are some of the best one for networking white papers.

- Experimentations

Verify that the honeypot and monitoring tools are correctly working. Using Penetration tools, I will try to access the services offered by the honeypot (e.g. FTP or SSH servers). If the monitoring tools are working as they should, the honeypot should log the different attacks.

Generate new IDS signatures using the log file of the honeypot.

- Evaluation

The evaluation consists in comparing the IDS signatures against the signatures created by the

honeypot.

This work will require the use of specialist software:

All the software needed will be open source

This work will require the use of specialist hardware: No

The project is being undertaken in collaboration with: Edinburgh Napier University

G.3. Proposal References

Davis, David. 2009. Best Practices for Securing VMware ESX Server. *Petri IT Knowledgebase.* [Online] 2009. [Cited: 31 03 2011.] http://www.petri.co.il/secure-vmware-esx-server.htm.

Georgieva, Tsveti. 2009. Advantages of Virtualization. *suite101.* [Online] 17 Nov 2009. [Cited: 25 Mar 2011.] http://www.suite101.com/content/advantages-of-virtualization-a170746.

Jackson, Peter. 2005. *Detection of Netowrk Threats Using Honeypots.* Edinburgh : Napier UNiversity, 2005.

Petri, Daniel. 2009. What You Need to Know About Securing Your Virtual Network. *Petri IT knowledge.* [Online] 08 01 2009. [Cited: 24 03 2011.] http://www.petri.co.il/what-you-need-to-know-about-vmware-virtualization-security.htm.

Spitzner, L. 2003. Definitions and Value of Honeypots. *tacking-hackers.* [Online] 2003. [Cited: 25 03 2011.] http://www.tracking-hackers.com/papers/honeypots.html.