AN INVESTIGATION OF NETWORK COUNTERMEASURE AGAINST FAST SELF-PROPAGATING MALWARE

Muhammad Aminu Ahmad

A thesis submitted in partial fulfilment of the requirement of the University of Greenwich for the degree of Doctor of Philosophy

February 2017

DECLARATION

I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

Muhammad Aminu Ahmad

First Supervisor: Dr Steve Woodhead

Second Supervisor: Dr Diane Gan

ABSTRACT

A self-propagating malware is a malicious software program that spreads itself across the Internet by exploiting flaws in software systems and therefore capable of launching attack against vulnerable Internet hosts. Fast self-propagating malware poses a security threat to hosts that are connected to the Internet because the speed of their propagation is very high and causes disruption of services across the Internet. Thus it becomes crucial to effectively detect and contain the propagation of fast self-propagating malware on the Internet.

This thesis presents a mechanism for the detection and containment of fast self-propagating malware. The thesis initially presents an overview of self-propagating malware and the need for a solution to counter the propagation of this class of malware. The thesis also presents a comprehensive literature survey to identify research gaps and limitations of previously reported worm detection and containment systems. Based on the identified limitations and shortcomings, an improved detection and containment scheme has been developed to counter the spread of fast self-propagating malware. The developed scheme, termed NEDAC, uses a cross-layer architecture to provide a combined countermeasure solution against fast self-propagating malware, i.e., a detection technique at network layer and a containment technique at data link layer. Furthermore, an improved testing environment, termed V-Network, has been developed for high fidelity malware experimentation and testing of countermeasure systems. An evaluation framework has been developed and used to test the NEDAC scheme along with other previously reported countermeasure systems using known and contemporary self-propagating malware. The NEDAC scheme demonstrated a better performance than the previously reported countermeasure systems.

CONTENT

FIGURES

TABLES

NOMENCLATURE

Abbreviations

ACLs	Access Control Lists
ARP	Address Resolution Protocol
CAIDA	Center for Applied Internet Data Analysis
CGI	Common Gateway Interface
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
DARPA	Defence Advanced Research Projects Agency
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarized Zone
DNS	Domain Name System
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
KVM	Kernel-based Virtual Machine
LLMNR	Link-Local Multicast Name Resolution
MAC	Media Access Control
NEDAC	Network Detection and Data Link Containment
NetBIOS	Network Basic Input/Output System
NIDS	Network Intrusion Detection System
NTP	Network Time Protocol
NVD	National Vulnerability Database
RAM	Random Access Memory
RDP	Remote Desktop Protocol
RIP	Routing Information Protocol
ROC	Receiver Operator Characteristics
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SYN	Synchronisation
TCP	Transmission Control Protocol

Telnet	Telephone Network
TN	True Negative
TP	True Positive
TTL	Time-To-Live
UDP	User Datagram protocol
VLAN	Virtual Local Area Network
VMM	Virtual Machine Monitor

Symbols

Pm	Per million Internet hosts
Rip	Routable IP address space
Sp	Susceptible Population

Chapter 1

GENERAL INTRODUCTION

1.1 Introduction

The Internet today provides an unprecedented medium that connects different devices together in order to share information across the globe. This plays a key role in our personal daily activities and operations of different organisations and governments. Additionally, most critical and financial systems are largely dependent on the Internet for their communications and data transmissions. As a result, the Internet must be kept available continuously and secured against malware and other malicious activities that may compromise its integrity.

Network worms are a class of malware that self-propagate on the Internet by exploiting vulnerabilities and flaws in software systems that have not been acknowledged (zero-day worms) or patched at the time of outbreak. The Internet has experienced a number of worm outbreaks such as Slammer, Code Red and Witty (Tidy et al., 2014) and Conficker (Soltani et al., 2014) that caused disruption of services and significant financial losses to government, transportation and other institutions ranging from millions to billions of US Dollars (Fosnock, 2005) as summarised in Figure 1.1. Additionally, the Stuxnet worm was discovered in 2010, which targeted industrial control systems in order to cause damage (Chen and Abu-Nimeh, 2011; Falliere et al., 2011). Stuxnet led to the release of other variants such as Duqu (Chien et al., 2012), Flame (Goyal et al., 2012) and Gauss (Bencsáth et al., 2012) for cyber espionage. The number of worm outbreaks reduced

from 2004 due to change in malware author motivation from causing network disruption to financial gain, which is more beneficial. Therefore fast scanning worms became less useful but the technical threat remained. This threat makes network worms to be a viable option for causing serious damage and disruption to the Internet even without malicious payloads as in the case of the Slammer worm (Tidy et al., 2014).

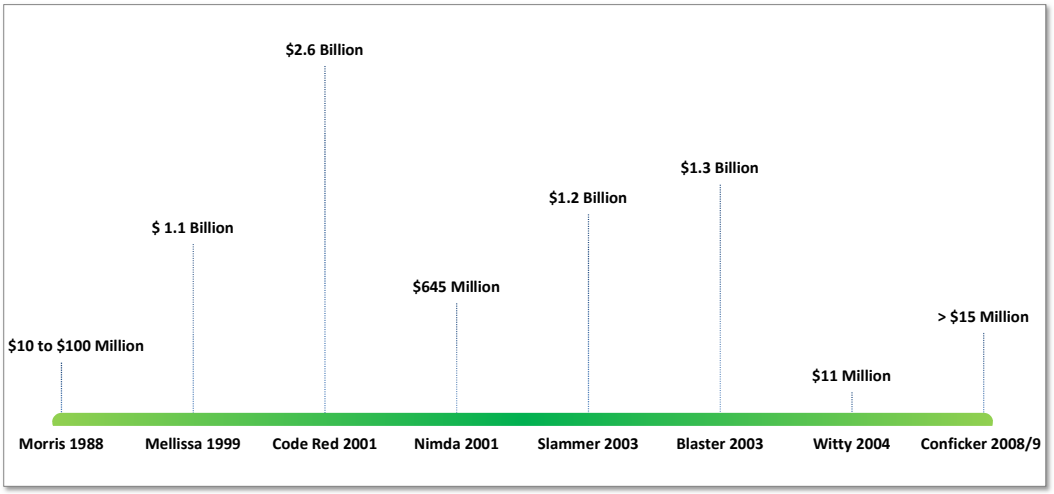


Figure 1.1: Financial losses caused by previous worm outbreaks on the Internet

A vulnerability is a flaw or weakness in software, services or security policies that allows buffer overflows or remote code execution on a target host (Tidy et al., 2014). This enables an attacker to send worm code capable of infecting vulnerable hosts on the Internet. An attacker can also transfer a backdoor to a vulnerable host, which can be used to acquire complete control of a victim host. Vulnerabilities that can be exploited by a worm continue to be published by system vendors including the Microsoft RDP (CVE-2012-0002) of 2012 (CVE, 2012) and ShellShock (CVE-2014-6271) (CVE, 2014a) and Drupal (CVE-2014-3704) (CVE, 2014b) of 2014. These vulnerabilities can be exploited by sending a crafted datagram to a target host, which allows remote code execution. The present threat of worm outbreak therefore remains clear.

1.2 Motivation

Common countermeasures to malware are signature-based anti-virus software, network intrusion detection systems and host intrusion detection systems, which along with network firewalls, counter the effects of a large proportion of contemporary malware. However, the ability of such systems to counter the effects of zero-day fast scanning network worms is limited due to the zero-day (or close to zero-day) nature of many such malware samples combined with the propagation delay for the distribution of signatures. Therefore, the high propagation and infection rates of fast scanning network worms pose a significant security threat, with consequent damage to networks and the Internet. Thus, it is important to effectively identify and counter the propagation of fast scanning network worms, before causing damaging to networks and the Internet using a state of the art detection and containment mechanism. This mechanism must first work without the need to rely on signatures as in the case of common anti-virus software and signature based intrusion detection systems. Methods which rely on content signatures are unlikely to detect zero-day scanning worms because the signatures are unknown at the point of the outbreak. Secondly, due to the high propagation rates of zero-day fast scanning network worms, the traditional approach of waiting for patches to be released by vendors to fix vulnerabilities is not viable (Ahmad and Woodhead, 2015).

A range of behavioural detection and suppression mechanisms has been reported in previously published security research work. However, there are limitations and shortcomings in the reported mechanisms. The mechanisms that use payload information (e.g. the technique reported by Kaur and Singh (2014)) to identify worm datagrams have limitations such as computational complexity , management overhead (Jyothsna et al., 2011), high rates of false positives (Li et al., 2008) and incur significant delays in deployment and detection (Kim et al., 2012). Rate limiting techniques (e.g. those reported by Shahzad and Woodhead (2014b)) alone can only slow worm infections. Techniques that use rate of successful or failed connections (e.g. the technique reported by Rasheed et al. (2009)), and some datagram-header based anomaly detection systems

(e.g. those reported by Srivastava and Giffin (2010), Comar et al. (2013) and AlEroud and Karabatis (2013)) are specific to TCP-based worms and/or consume resources in order to keep track of distinct connection and host information, most especially in large networks (Ahmad and Woodhead, 2015).

Thus, it is desirable to have a fast and accurate worm countermeasure solution that is capable of detecting and containing the propagation of fast scanning network worms before they cause damage to systems and networks without affecting benign network traffic; that is the false-positive rate must be low. Furthermore, numerous countermeasure techniques (Liao et al., 2013) have been developed by security researchers to mitigate worm attacks, but the speed at which zero-day fast scanning network worms propagate is very high for the techniques to act effectively.

1.3 Key Research Contributions

This thesis presents a worm countermeasure mechanism that uses a sensitive network level detection system to identify fast scanning network worms. The mechanism also uses a data link containment system to block an identified worm infection, which enables the isolation of an infected host from a local network. The cross-layer technique enables the mechanism to provide a powerful and combined countermeasure solution for fast scanning network worms. A network level solution was chosen because it provides an overall picture of datagrams moving across the network and the number of devices involved in detecting malware are fewer compared to a host based solution. Furthermore, an improved virtualised network environment with large scale and advanced features has been developed for worm experimentation and testing of countermeasure systems. The performance of the countermeasure mechanism has been evaluated using previous worms and contemporary pseudo-worms developed based on contemporary vulnerabilities.

1.4 Thesis Outline

This section presents the organization of the remaining parts of the thesis. Chapter 2 presents a comprehensive survey of related research work in the area of worm detection and containment, and then worm and countermeasure testing environments. Chapter 3 presents the developed worm detection and containment mechanism. Chapter 4 presents the environment used for worm experimentation and countermeasure testing. Chapter 5 presents details of the evaluation programme used to test the performance of the developed worm countermeasure solution. Chapter 6 presents details of the experimentation conducted and the results obtained. Finally, Chapter 7 concludes the thesis and discusses possible future work.

Chapter 2

SURVEY OF RELATED WORK

2.1 Introduction

A network worm is a class of software that is potentially dangerous, because of its highly virulent nature. Fast scanning network worms are a particularly dangerous sub-class of such malware, because hosts infected by a fast scanning network worm send out worm infectious datagrams to random IP addresses rapidly seeking more vulnerable hosts as seen during the outbreak of the Slammer worm (Jamil and Chen, 2006), which caused network congestion and disruption of services (Tidy et al., 2014).

The self-propagation behaviour of network worms has attracted the attention of cyber security researchers due to the speed with which fast scanning network worms infect a population of vulnerable hosts. There have been significant recent advances in studying the infection pattern of network worms using different techniques and mechanisms (Liao et al., 2013), with the aim of developing a range of countermeasure systems. Despite significant research efforts that have been devoted to preventing the spread of worms, the propagation of fast scanning network worms poses a serious security threat to networks and the Internet. This is because fast scanning network worms propagate very quickly through networks and across the Internet (Chen et al., 2003). Some fast scanning network worms can potentially infect almost all vulnerable hosts in a short period of time such as the case of the Code Red and Slammer worms (Tidy et al., 2014). Additionally, the rapid advances of high speed Internet connectivity and network technologies has

enabled fast scanning network worms to potentially propagate at a speed much faster than human countermeasures can respond (Wang et al., 2014).

Thus, the threat posed by the rapid spread of fast scanning network worms requires an automated technique that is relatively fast to detect and counter such malware because the speed of their propagation is high for manual responses such as patching, router-/firewall reconfiguration or waiting for the release of new signatures (Jamil and Chen, 2006).

This chapter begins by presenting an overview of network worms, their propagation characteristics and the vulnerabilities they exploit. The chapter then presents a survey of the mechanisms proposed in literature to detect and contain such worm propagation. The survey concentrates on identifying research issues in the development of fast scanning network worm detection and countermeasure systems and the environments used to evaluate the virulence of the worms and countermeasures.

2.2 Network Worms

A network worm is a malicious software program that propagates itself across a network by infecting hosts and in some cases launching malicious activities. The infection is achieved by exploiting vulnerabilities in host network services that have not been patched or unacknowledged (zero-day worms) at the point of an outbreak (Tidy et al., 2014).

A host that has been infected by a worm actively discovers and spreads a copy of the worm to other vulnerable hosts on the Internet (Ahmad and Woodhead, 2015). Unlike a virus that searches for files in a computer system to which to attach itself or requires some sort of user intervention for propagation (Yang et al., 2013), a worm has the ability to search for new targets and propagates itself across networks (Wang et al., 2014). The propagation of a worm uses either TCP or UDP transmission schemes (Li et al., 2008). In TCP transmission, the worm requires a three-way handshake to establish a connection with the vulnerable host before transferring a copy of the worm code. Therefore, when the worm sends out a TCP SYN datagram to initiate a connection, it must wait until

it receives a corresponding SYN/ACK or timeout datagram from the host it is trying to probe. On the other hand, UDP is connectionless, so UDP worms do not require a connection to be established before infection can begin. The implementation of the worm is normally self-carried and included in the first datagram sent to the target, which make UDP worms spread very fast as seen in the Slammer outbreak (Tidy et al., 2014). Fast scanning network worms often cause harm to services and networks such as denial of services (Moore et al., 2003a), damage (Shannon and Moore, 2004), deploying malicious programs (Moore et al., 2003b) and other cyber crimes (Falliere et al., 2011).

A worm infection often begins with a single host, which then targets a set of IP addresses, searching for more vulnerable hosts. If the worm successfully hits a vulnerable host, it then transfers over a copy of itself to the new host, which begins executing the worm code. Network worms can be broadly classified as scanning worms and topological worms based on the target discovery strategy (Wang et al., 2014). Scanning worms propagate themselves on the Internet by probing addresses looking for vulnerable hosts. Figure 2.1 depicts the propagation behaviour of a scanning worm.

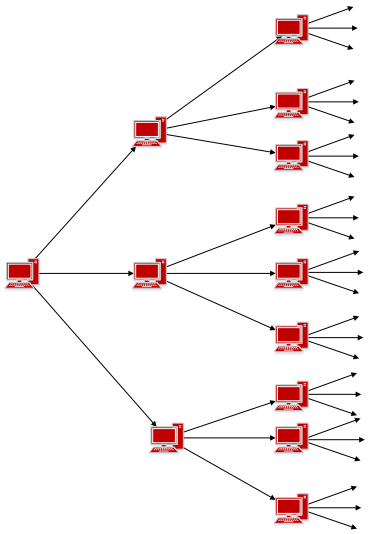


Figure 2.1: Propagation of a scanning worm

A topological worm infects topological neighbours based on the local information found on the victim hosts such as URL addresses on the disc of an infected host and use them to conduct more probing attacks. Figure 2.2 depicts the propagation behaviour

of a topological worm. Topological worms generally spread more slowly than scanning worms (Weaver et al., 2003), particularly fast scanning worms.

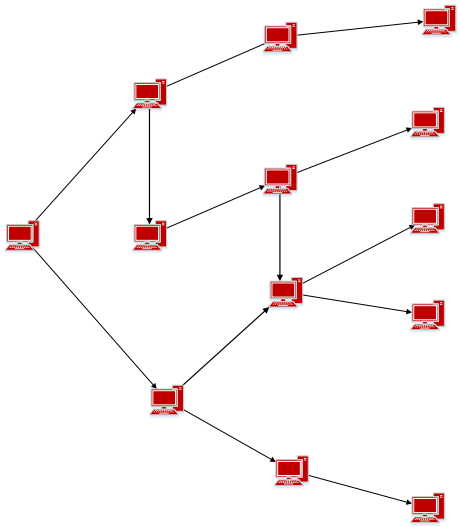


Figure 2.2: Propagation of a topological worm

2.2.1 Scanning Worms

A scanning worm spreads by employing various target discovery techniques in order to exploit a vulnerability (Weaver et al., 2003). Upon infecting a vulnerable host, the worm uses the victim host to spread copies of itself automatically by probing a set of IP addresses (Wang et al., 2014). This technique of probing a set of IP addresses or working through a set of ordered addresses to identify vulnerable hosts is known as scanning. The scanning strategy can be classified as random, permutation, localized, hit-list, stealthy or routable scanning (Staniford et al., 2002; Zou et al., 2006), as detailed below.

- **Random Scanning:** The random scanning technique is the most common and simple target finding strategy, where worms generate pseudo-random IP addresses and then try to connect to them. Most scanning worms propagate by probing pseudo-random addresses looking for vulnerable hosts. Upon infecting a vulnerable host, the copy of the worm is then transferred to the infected host, which in turn enables the victim to infect other vulnerable hosts. The limitations of random scanning are:

1. Probing addresses multiple times.
2. Scanning addresses that are not used.
3. No determination of whether hosts are already infected.
4. Often generates a large amount of unusual traffic and therefore is relatively easy to detect.

- **Permutation Scanning**: In permutation scanning, an infected host is assigned with part of a pseudo-random permuted IP address space to scan and infect vulnerable hosts, i.e., dividing up the address space within which each new worm instance scans for new vulnerable hosts. The scanning behaviour starts randomly, then the infected hosts work their way through the permutation to find vulnerable hosts. In addition, if the worm encounters an already infected host, it assumes that another worm is working through that sequence of permutation, and therefore chooses a new random point and proceeds with the scanning. This strategy resolves some of the problems of random scanning whilst maintaining random probing behaviour.

- **Localised Scanning**: The localised scanning technique probes vulnerable hosts within the subnet of an infected host. The probe initially infects vulnerable hosts on the Internet, and then targets addresses that are close to the infected host. This technique has the potential to successfully spread rapidly within internal networks because the worm has already crossed firewalls and other perimeter security systems.

- **Hit-list Scanning**: The hit-list scanning technique uses a pre-compiled list of host IP addresses that are potentially vulnerable. This list is compiled by scanning the Internet slowly and stealthy over a long period to evade any suspicion, collecting information from publicly available resources such as SHODAN (Goldman, 2014) or manual reconnaissance. Hit-list scanning can be extremely fast, but requires the compilation of the target address list.

- **Routable Scanning**: The routable scanning technique narrows the scanning

space by probing IP addresses from within the routable IPv4 address space instead of the whole IPv4 address space, which increases the spreading speed of a worm. Routable scanning also reduces the chances of detection due to a reduction in the number of "host unreachable" ICMP return datagrams and so makes the worm less easy to detect.

2.2.2 Worm Activation

After a worm probes an IP address and transmits a copy to the infected host, the worm is then activated through self-activation or a scheduled process (Weaver et al., 2003). In self-activation, the worm will initiate its execution using commands with the privilege of the exploited service. Alternatively, a scheduled process on the infected host is used to execute the worm.

2.2.3 Worm Payload Execution

The final action of a network worm after infecting a vulnerable host is to execute its payload, which depends on the payload carried by the worm, or, indeed whether it has one (Weaver et al., 2003). The payload is the part of the worm that carries the data identifying the goal of the worm author. The payload format and its propagation can occur using different mechanisms. Some worms make the payload size variable by padding the payload with garbage data, or fragment the worm payload differently and reassemble the pieces at the target; these are called monomorphic worms (Li et al., 2008). A worm that changes its payload dynamically by scrambling the program, so every instance of the worm looks different but functions in exactly the same way is known as a polymorphic worm (Kaur and Singh, 2014). The payload can carry code for destructive purposes (Shannon and Moore, 2004), financial crime (Goyal et al., 2012), stealth and other malicious activities (Bencsáth et al., 2012). The worm can also carry no payload (Moore et al., 2003a).

2.3 Wormable Vulnerabilities

A vulnerability is a flaw or weakness in software, services or security policies that allows buffer overflows or remote code execution on a target host. This enables an attacker to send worm code capable of infecting vulnerable hosts on the Internet. An attacker can also transfer a backdoor to a vulnerable host, which can be used to acquire complete control of a victim host.

A vulnerability is said to be wormable (Nazario et al., 2004) if it is network reachable, has the potential to provide remote code execution and network access and does not require human interaction (Tidy et al., 2014). Individual vulnerabilities can be researched through a number of online sources that provide details of identified vulnerabilities such as the Common Vulnerabilities and Exposures (CVE) system (CVE, 2014c) and the National Vulnerability database (NVD) (NVD, 2014). The CVE and NVD systems focus on providing details for a range of vulnerabilities and keep notes of whether a vulnerability is network reachable or requires human interaction if exploited. Additionally, Symantec Connect (Symantec, 2015) provides working exploits for some vulnerabilities. These sources provide the necessary information for assessing the wormability of many vulnerabilities.

2.3.1 Vulnerabilities Employed in Historical Worm Outbreaks

Some of the fast scanning network worm outbreaks that have been experienced on the Internet are Code Red, Nimda, Slammer and Witty. The vulnerabilities exploited by these worms are detailed below.

- **Microsoft IIS Indexing Service DLL Vulnerability:** This was the vulnerability exploited by the Code Red worm in 2001 (Berghel, 2001) and Nimda also in 2001 (CERT, 2001; Mackie et al., 2001). The vulnerability was reported on 18th June, 2001 by eEye Digital Security (eEye Digital Security, 2001). It was a remote buffer overflow vulnerability in all versions of the IIS Web server software, at the time, which used the default installation process including several Internet Service

API (ISAPI) extensions or dynamic linked libraries. The extension, idq.dll (Indexing service), was a component of Microsoft Index Server that provided support for administrative scripts (.ida files) and Internet Data Queries (.idq files), which enabled searching data on a web server (Cowie et al., 2001). A security vulnerability was discovered in "idq.dll" that allowed a remote attacker to initiate a buffer overflow attack, which in-turn caused a denial of service or remote code execution on the server. The remote code execution capability enabled an attacker to gain complete control of the server, including changing web pages, reformatting the hard drive or adding new users to the local administrators group (Dolak, 2001).

- **Microsoft SQL server 2000 Vulnerability:** This was the vulnerability exploited by the Slammer worm in 2003 (Moore et al., 2003a). The vulnerability was found in Microsoft SQL Server 2000 and Microsoft Desktop Engine (MSDE) 2000, which exhibited two buffer overflow vulnerabilities that could be exploited by a remote attacker without the need for authentication (Joukov and Chiueh, 2003). The attack could be channelled over UDP and successful exploitation of the flaw allowed complete control over a victim host.

- **Internet Security Systems (ISS) Vulnerability:** This was the vulnerability exploited by the Witty worm in 2004 (Shannon and Moore, 2004). The vulnerability affected the protocol analysis module (PAM) for monitoring application traffic in ISS firewall products. A routine in the PAM that monitored ICQ server responses contained a series of stack based buffer overflow vulnerabilities, which assumed an incoming UDP datagram on port 4000 to be an ICQv5 server response without sanity checking (eEye Digital Security, 2004).

2.3.2 Contemporary Wormable Vulnerabilities

Some more contemporary wormable vulnerabilities include the Microsoft RDP vulnerability of 2012 (CVE-2012-0002) and the ShellShock vulnerability of 2014 (CVE-2014-6271), which are summarised below.

- **Microsoft Remote Desktop Protocol (RDP)**: The Microsoft RDP is a proprietary protocol used to remotely access Windows-based hosts across a network. This vulnerability (CVE-2012-0002) was identified to exist in Microsoft Windows XP, Vista, 7, Server 2003 and Server 2008. The vulnerability can be exploited by sending a crafted datagram to the TCP or UDP port 3389 on a host running RDP. The vulnerability potentially allows remote arbitrary code execution. An attacker can then use this to send copies of the malicious datagrams to any vulnerable host, making the vulnerability wormable.

- **ShellShock:** Bash (bourne-again shell) is a common UNIX command processor that uses a text-based window for executing user and application commands passed to it. Bash is used in many versions of Linux, UNIX and Mac OS X, including versions which run web servers employing the Common Gateway Interface (CGI). Among the commands passed to Bash by applications are those which allow the application to set environment variables. The ShellShock vulnerability (CVE-2014-6271) affects the way environment variables are set by the applications, i.e., the Bash shell processes the trailing strings after function definitions in the values of environment variables, which allows an attacker to add malicious code to the environment variable and arbitrary execution of code when the environment variable is set on a vulnerable host.

2.4 Worm Detection Techniques

Previously reported worm countermeasure systems have used detection techniques at either the host or network perimeter levels (Smith et al., 2009). A brief description of each technique is set out below.

2.4.1 Host Level Detection System

A host level detection system uses end-host information to detect anomalous behaviour such as buffer overflows, correlating network data to memory errors, and looking for

patterns in system calls (Li and Shad, 2014). This technique requires deployment on every host to detect a network or malware attack (Chen et al., 2014). Host level worm detection systems reported in literature include COVERS (Liang and Sekar, 2005) Vigilante (Costa et al., 2005) and SWEEPER (Tucek et al., 2007). Host level worm detection systems are difficult to manage centrally and they are vulnerable to compromise when a host is infected (Jamil and Chen, 2006).

2.4.2 Network Level Detection System

Network level detection systems are usually deployed at the network perimeter to protect the entire network of hosts by monitoring inbound and outbound network traffic (Li et al., 2008). Therefore network level detection systems become more desirable compared to host level system due to a lower installation and management overhead (Li and Shad, 2014). Network level intrusion detection systems can mainly be categorized into signature-based and anomaly-based detection systems (Jyothsna et al., 2011).

2.4.3 Signature-based Network Detection System

A signature-based detection system maintains a database of signatures for previously known attacks (Liao et al., 2013). An alarm is raised if a datagram in the network matches a signature in the database (Kim et al., 2012). The signature database is updated frequently in order to efficiently detect new threats (Kumar and Sangwan, 2012). A number of signature-based detection systems exist, including Snort (Khamphakdee et al., 2014) and Bro (Sharma and Sharma, 2015). Snort and Bro use static signatures, which are a set of previously known byte sequences considered to be malicious. Although Snort has an anomaly detection component that monitors TCP protocol anomalies, such as data on SYN packets and data received outside the TCP window (Khamphakdee et al., 2014), it does not detect zero-day worm attacks. Thus the drawback of Snort and Bro is the inability to detect new attack whose signature is absent from the database (Beigh and Peer, 2014). Autograph (Aljawarneh et al., 2016) and EarlyBird (Punithan et al., 2016) were developed to generate signatures dynamically for Snort and Bro by analysing

the content of network traffic to find common byte sequences. These detectors together with Snort or Bro can detect the presence of zero-day worms, but cannot detect polymorphic worms (Kaur and Singh, 2014). Additionally, these detectors can be misguided into labelling legitimate traffic as malicious (Chung and Mok, 2006; Newsome et al., 2006; Perdisci et al., 2006). Generally, signature-based systems are effective in detecting known attacks and are easy to deploy (Liao et al., 2013), but limited in maintaining a large database of previously and newly identified threats (Ahmad and Woodhead, 2015). Additionally, the efficiency of the detection algorithm is affected by the large size of signature database (Li et al., 2008).

2.4.4 Anomaly-based Network Detection System

An anomaly-based detection system examines network traffic in order to build a profile of the normal behaviour and then raises an alarm for events that deviate from the normal profile (Liao et al., 2013). In contrast to signature-based systems, anomaly-based systems can detect new attacks (Li et al., 2008). Anomaly-based detection systems look for deviations from a normal profile, based on the datagram header information or payload information (Cheema et al., 2009). Datagram-header based anomaly detection systems use datagram header information to build a model of normal traffic flow of data and then attempt to detect any deviation from the normal behaviour of the network traffic observed. Payload-based anomaly detection systems observe the payload content of network traffic to identify a byte pattern that indicates the presence of a worm.

Significant research efforts in anomaly-based network intrusion detection systems led to the existence of numerous approaches (Jyothsna et al., 2011; Liao et al., 2013) that identify the presence of worms using datagram header information or payload information. A selection of the main techniques are summarised in Sections 2.5 and 2.6.

2.5 Datagram-header Based Anomaly Detectors

Datagram-header based anomaly detection systems use datagram header information to build a model of normal traffic flow and then attempt to detect any deviation from the normal behaviour of the network traffic observed (Ahmad and Woodhead, 2015). Datagram-header based systems also observe the rate of failed and successful transmission of datagrams to detect an attack (Li and Shad, 2014; Weaver et al., 2004). Sections 2.5.1 through 2.5.3 present a selection of the main techniques that use datagram-header information.

2.5.1 Anomalous or Unused Header Information

Most datagram-header based anomaly detection systems look for anomalous or unusual port number and IP addresses, i.e. ports and IP addresses not observed during training on normal non-hostile traffic. Among the approaches that use datagram-header information are those that monitor source and destination IP addresses and ports, and Ethernet header fields such as the techniques reported by Mahoney and Chan (2001), Gu et al. (2004), Qin et al. (2004) and Comar et al. (2013).

2.5.1.1 Packet Header Anomaly Detection (PHAD)

Mahoney and Chan (2001) developed the Packet Header Anomaly Detection (PHAD) technique, which learns the normal ranges of values for each datagram header field at the (Ethernet), network (IP), and transport/control layers (TCP, UDP, ICMP). PHAD examines 33 datagram header fields, and was designed to be as protocol independent as possible. PHAD uses probability to rate anomalies in detection mode based on the rate of anomalies observed during a training phase; the rarer the detection anomalies, the more likely they are to be hostile when identified in detection mode. During the training phase, a datagram field that has been observed n times with r distinct values, must have r "anomalies" during the training period. If the rate continues, the probability that the next observation will be anomalous is approximated by $\frac{r}{n}$. In detection mode,

each anomalous datagram header field is assigned a score that is inversely proportional to the probability of being anomalous at time t as shown in equation 2.1. The sum of the scores is finally taken to decide if the datagram should be considered anomalous.

$$Score_{field} = \frac{tn}{r} \tag{2.1}$$

PHAD was evaluated using the 1999 DARPA off-line IDS evaluation dataset (Lippmann et al., 2000). PHAD was trained on seven days of attack free network traffic (week 3) collected from a sniffer between the router and victim machines, and tested on nine days of traffic from the same point (weeks 4 and 5, except week 4 day 2, which is missing), during which there were 201 attacks (183 in the available data). The results of the evaluation showed a successful detection of 72 out of the 201 attacks. PHAD detected four out of the seven IP address probe attacks available in the dataset. PHAD generated 20 alarms; eight true positives and twelve false positives.

The shortcoming of the PHAD technique is the need for a training phase, which delays the time of deployment and when the system can start the detection process. Furthermore, changes in the state of a network such as new hardware and software installation and upgrades will potentially cause more false positives due to new range of values in header information not observed during training phase. Additionally, PHAD has a limitation because the technique did not take containment into account.

2.5.1.2 Destination-Source Correlation (DSC)

Gu et al. (2004) developed an algorithm, termed Destination-Source Correlation (DSC), which correlates incoming and outgoing traffic. A training exercise was conducted in a local network to keep record of normal profile of infection-like scan rates, where each port witnessed in the traffic will be recorded including the local destination host address. Thus, if a host received a datagram on port i, and then starts sending datagrams destined for port i, it becomes a suspect. Then if the immediate outgoing scan rate for the suspect host deviates from a normal profile, the suspicious victim is considered to be infected.

The algorithm then outputs a worm alert using a local warning system, and indicates the victim IP address and its scan rate. The DSC algorithm was developed with the aim of detecting fast scanning network worms.

The DSC algorithm was tested in order to evaluate its false positive rate using trace files collected from two distinct sources; Waikato Internet monitoring project (WAND) (WAND, 2001) and Georgia Institute of Technology, College of Computing (GTTrace). The data from WAND was used to train DSC; 80% of the trace was used as training data and 20% was used for testing the false positive rate. The experiment established a normal profile of scan rate for every port immediately after infection-like behaviour. The experiment focused on traffic from some representative TCP ports, i.e., $21, 22, 23, 25, 80, 139, 445$ and UDP ports, i.e., $53, 1434$. The local warning system was tested using a worm model developed in the datagram level network simulator GTNetS (Georgia Tech Network Simulator). The results of the experiment showed 0.19% and 3.08% infection percentage using /12 and /16 networks.

The limitation of the DSC algorithm was the inability to detect worms that use two or more attack vectors. As a result, Qin et al. (2004) improved the technique using a combination of HoneyStat and the modified version of the DSC algorithm to solve the limitation of DSC implementation.

2.5.1.3 Modified DSC

Qin et al. (2004) proposed a modified DSC algorithm that monitors IP or media access control (MAC) addresses to defend against IP address spoofing. The idea of this technique is that HoneyStat can be used to capture scans on different ports, and therefore covers what DSC cannot detect. The modified DSC algorithm used three bloom filters denoted as D_{i-1},D_i, and S_i, for every port, which track the destination addresses at time ticks and for scans directed at the network, and the source addresses for traffic (SYN and UDP) originating at time tick i from the network. Thus, at every time tick i, suspicious victim's IP address and the number of scans the victim sends out (i.e., scan rate) is recorded. If the scan rate deviates from a normal profile, the suspicious victim

is regarded as a real victim infected by a worm. Then a worm alert is generated and indicates the victim IP address and its scan rate.

The modified DSC algorithm was evaluated using the discrete time-based AAWP (Analytical Active Worm Propagation) model (Chen and Ranka, 2004) for simulation and detection time analysis. The performance of the modified DSC algorithm was analysed under three different worm scan strategies; random scan, routable scan and divide-conquer scan. The detection performance was measured in terms of infected percentage of the whole vulnerable host population on the Internet when a worm alert was generated. The results of the infection percentages for random, routable and divide-conquer scanning using /12 and /16 networks were 0.82% and 13.11%, 0.19%, and 3.05%, and 0.19% and 3.01% respectively.

The two implementations of the DSC algorithm required a training phase, which delays the time of deployment and when the system can start the detection process and did not take containment into account. Additionally, host-based implementation of the algorithm incurred a high installation and management overhead most especially in large networks (Li and Shad, 2014).

2.5.1.4 Machine Learning

Comar et al. (2013) developed a framework to detect and isolate malicious network traffic and classify them as known malware, a variation of known malware or new malware. The technique used machine learning to detect and classify layer-3 and layer-4 headers by introducing a tree-based feature transformation to overcome issues due to imperfections of the data and to construct more informative features for malware detection. Initially, malicious and benign flows are isolated, then a collection of classifiers is used to categorise each instance of a specific malware class; to identify new and known malware. The framework was evaluated using network traffic from an Internet Service Provider. The dataset contained $216,899$ flows, out of which only $4,394$ flows were labelled as malicious and categorized into one of the 38 known classes. Experiments were conducted to compare the classification performance of tree-based features against other

baseline approaches for handling data imperfection issues. The results of the framework gave a higher area under the ROC curve.

The shortcoming of the framework is the need a training phase, which delays the time of deployment and when the system can start the detection process. Finally, the framework did not take containment into account.

2.5.2 First or Failed Contact

Most scanning network worms (TCP-based) initiate connection requests to pseudo-random IP addresses, which results in a number of failed connections (Li et al., 2008). As a result, some approaches, such as those reported by Williamson (2002), Jung et al. (2004), Weaver et al. (2004) and Li and Shad (2014) used the status of a connection request and first-contact connection to detect worm behaviour.

2.5.2.1 Rate limiting

Williamson (2002) used source and destination IP addresses of datagrams to limit the rate at which malware can try to connect to new IP addresses. In this mechanism, a connection is delayed in order to slow the propagation of malware. The system has one component to determine connection requests to new hosts and another component that limits the rate of those connections. Whenever a host makes a request, the system verifies the newness of the destination IP address using a short list of recent connections. The request is then added to a delay queue if it is new, otherwise it will be processed as normal. The delay time is a compromise between reducing the impact of false alarms and limiting the spread of malicious activities in the network.

The mechanism was evaluated with normal web traffic (http) using the browsing behaviour (time, url visited) of five researchers collected over a five month period as reported by the author. The results of the evaluation showed Williamson's scheme to be effective in slowing down worm propagation. The scheme also demonstrated simplicity and ease of management.

The scheme incurred large delays for legitimate applications when the delay queue was saturated with worm traffic (Wong et al., 2006). Another possible consequence of the scheme is the size of the delay queue; small queue size slows down worm propagation and increases false positive rate, while large queue size reduces false positive rate and increases worm propagation (Weaver et al., 2004).

2.5.2.2 Threshold Random Work (TRW)

Jung et al. (2004) proposed an algorithm which they termed the Threshold Random Walk (TRW), derived from sequential hypotheses to identify malicious remote hosts. The algorithm is based on the fact that scanning malware is more likely to attempt to access hosts and services that do not exist than legitimate remote hosts, since they lack precise knowledge of which hosts and ports on the target network are currently active. The algorithm requires that a remote host attempt to establish a new TCP connection to a local destination host is marked 0 if the connection attempt is successful i.e. if there is corresponding TCP reply from the destination local host. On the other hand, failure to establish a successful TCP connection marks it as 1 because the connection attempt is either made to an inactive host or to an inactive service on an active host. The algorithm then decides whether a remote host is a scanner based on the count of its successful and failed connection attempts. The technique considers two hypotheses, H_0 and H_1, where H_0 is the hypothesis that a given remote host is benign and H_1 is the hypothesis that such host is a worm scanner.

The performance of the algorithm was evaluated using trace-driven simulations with Lawrence Berkeley National Laboratory and International Computer Science Institute datasets (Paxson, 2005). Due to the complexity of the random walk calculation in the algorithm and its reliance on failed connection, Weaver et al. (2004) improved the TRW algorithm.

2.5.2.3 Modified TRW

Weaver et al. (2004) simplified the TRW scheme by considering all new connections

to be a failure until a response is received. The technique uses a scan detection and suppression mechanism; caches are used to track the activity of both new connections and IP addresses to reduce the random walk calculation in the algorithm. The algorithm drops a datagram if it does not match an existing and successfully-established connection after a predefined threshold count. The technique detects and stops a scanning worm if its failed connection attempts exceed 10 with a low false positive rate. The algorithm keeps track of both TCP and UDP datagrams and is suitable for both hardware and software implementations.

The algorithm was evaluated using hour-long traces of datagram headers collected at the Lawrence Berkeley National Laboratory. The results of the evaluations showed that the scheme had a good sensitivity of detecting scanning worms with a low false positive rate. The modified TRW scheme generated false negatives due to saturation of the connection cache, which caused a new scan attempts not to be recorded because it aliases with a prior established connection. Thus, the sensitivity of the scheme increased with an increase in the size of the connection cache in order to reduce aliasing.

2.5.2.4 Self-propagating Worm Observation and Detection (SWORD)

Li and Shad (2014) proposed a worm detector, which they termed Self-propagating Worm Observation and Detection (SWORD). SWORD comprises two main modules; a Burst Duration Detector (BDD) and a Quiescent Period Detector (QPD). The BDD module encompasses a burst detection algorithm to prevent fast scanning network worms by creating a window for every different size of first-contact connections, i.e, the algorithm computes a threshold for a "2-connection" burst, a "3-connection" burst, a "4-connection" burst, and so on, up to a maximum burst size. The algorithm measures multiple different durations observed for each burst size, during a training stage, and then bases the threshold on the minimum duration observed for a given burst size. The QPD module ensures that quiescent periods in network activity do not disappear because of constant worm scanning. The idea is a normal host will exhibit regular quiescent periods where it does not make any first-contact connections. Therefore, if a host does not

display quiescent periods as observed during the training stage, and has been "active" for a long period, then QPD determines that such a host is infected by a worm that is scanning the network. This is achieved by measuring the mean and standard deviation of all the active periods that are separated by a quiescent duration. These values are used to generate a threshold duration for active periods, which is the mean plus β times the standard deviation. The value of β can be tweaked for different environments to minimise false positives. If a host has an active period exceeding the threshold duration for any quiescent period, it is likely infected by a worm. The co-existence of the two techniques enables SWORD to detect worm behaviour and makes it hard for a worm to avoid detection. The SWORD detector uses k-means clustering (Ahmad and Dey, 2007) to separate hosts into groups such that different thresholds can be applied to different groups of hosts.

The performance of SWORD was evaluated against other detectors in four distinct environments; campus, enterprise, department and wireless. Generally, SWORD demonstrated a good performance in detecting evasive worms. However, SWORD incurred overhead in storing different thresholds for all the connection burst sizes, which requires more storage space. An increased computational requirement is also needed for SWORD to examine the connection history and determine violation of any of the thresholds. Additionally, SWORD required a training phase and did not take containment into account.

2.5.3 Domain Name System (DNS) Activities

Another detection approach used to identify scanning worms utilises the DNS activities of hosts to detect worm propagation such as mechanisms reported by Whyte et al. (2005) and Shahzad and Woodhead (2014b).

2.5.3.1 DNS-based Rate Limiting

Whyte et al. (2005) used DNS-based rate limiting to suppress scanning worms in an enterprise network. The observation was that scanning worms often use numeric IP addresses instead of the qualified domain name of the system (Ganger et al., 2002),

which eliminates the need for a DNS query. In contrast, the vast majority of legitimate publicly available services are accessed through the use of the DNS protocol; the network service that maps numeric IP addresses to corresponding alphanumeric names. Therefore the main idea behind this technique is that the absence of DNS resolution before a new connection is considered anomalous. The technique was implemented in software and used DNS anomalies to detect scanning worms between enterprise network cells and from enterprise networks to the Internet. An anomaly is considered if there is no observed DNS resolution before initiating a new connection a number of times.

The technique was evaluated by installing the software prototype on a commodity PC with a Linux operating system and a 10/100 network interface card. One week of network traffic was collected at a firewall in front of Carleton University research labs. The results of the evaluation showed a successful detection of scanning worms with a low false positive rate. The technique also slowed down the worm propagation.

2.5.3.2 DNS-based Rate Limiting and Awareness

Shahzad and Woodhead (2014b) proposed a scheme termed rate limiting plus look ahead (RL+LA) that is based on the correlation of Domain Name System (DNS) queries and a destination IP address of an outgoing TCP SYN or UDP datagrams leaving a network boundary. The RL+LA scheme also utilizes a communication scheme between a set of peer networks using a custom protocol termed Friends. The main idea in this counter-measure scheme is that the lack of a DNS lookup action prior to an outgoing TCP SYN or UDP datagram to a new destination IP address is used as a behavioural signature, while the Friends protocol spreads reports of the event to potentially vulnerable and uninfected peer networks within the scheme. In the RL+LA algorithm, for any TCP SYN or UDP datagram leaving the network, the algorithm looks for a corresponding DNS lookup in a network DNS cache, which saves the result of all DNS lookups along with the corresponding source and destination IP addresses. If such an entry is absent, it adds the source IP address to a counter cache and increments a corresponding count value. A threshold value, N, is defined in the counter cache and once the value is reached

the algorithm blocks outgoing traffic from the offending host using iptables and reduces N to $N/2$. The algorithm then sends an alert message using the Friends protocol to internal peer routers and to the border gateway router, which in turn forwards the alert to external peers in the scheme using the same protocol.

The scheme was evaluated using a software prototype in a virtualized host testbed that consists of six fully routable class C networks (192.168.0.0 to 192.168.5.0). The scheme used a developed network daemon that behaves in a way similar to the Slammer worm. The results showed the capability of the scheme in successfully detecting and slowing down scanning worms.

2.5.3.3 Limitations

The implementation of the DNS-based technique in the mechanisms reported by Whyte et al. (2005) and Shahzad and Woodhead (2014b) assigned a uniform and static threshold for the DNS-anomaly detection. This will potentially generate false positives in the presence of background traffic and other traffic demands that do not require DNS lookup. The mechanisms also left internal network hosts open to worm attack from local and remote infected hosts. Finally the implementations did not differentiate legitimate traffic that used numeric IP address from malicious traffic during worm scanning, which added more counts to the threshold and therefore increased potential false positives.

2.6 Payload Based Anomaly Detectors

Payload-based anomaly detection systems use protocol payload information to build a network traffic profile during a training phase and then use the developed profile to detect intrusions if there is a deviation of behaviour from the established norm by network traffic during the detection phase.

2.6.1 Statistical Distance

Wang and Stolfo (2004) proposed a payload-based anomaly detection method known as PAYL to detect and generate signatures for zero-day worms. PAYL uses a training phase to create a profile during normal operation and produce a byte frequency distribution as a model for normal payload. Based on this information, a centroid model is created and then during the detection phase, the Mahalanobis distance of each datagram payload from the centroid model is calculated. A datagram is considered to be anomalous based on its distance from the normal behaviour. If the distance is larger than a threshold value for the incoming traffic for port i, then such a datagram will be marked as suspect for port i and then placed in the buffer. An outgoing datagram from port i, that is detected as anomalous by the anomaly detector, is compared with the buffer content. A similarity score is computed and then Longest Common Substring (LCS) and Longest Subsequence (LCSeq) of any two compared strings are generated. If the similarity score is greater than a threshold value, the outgoing traffic is blocked and then a signature is generated in the form of LCS and LCSeq.

The effectiveness of PAYL was evaluated using three datasets of real worm traffic; EX, W and W1. The EX dataset was from an external commercial organisation, while W and W1 datasets were derived from two web servers at the Computer Science Department of the University of Columbia. Worm traffic, such as Code Red and Code Red II, were inserted into each dataset during the evaluation, which contained 40 worm datagrams. The result showed PAYL performed well in detecting Code Red attacks, buffer overflow attacks and other TCP-based attacks. The drawback of PAYL is the need for a training phase.

2.6.2 Statistical Dispersion

Kim et al. (2012) proposed a payload-based intrusion detection scheme using a standalone device that can be deployed on networks. The system is designed with a variable and dynamic training windows size. During the training period, the system activates

itself as soon as the window hits the size of 100 datagrams, and then continues with the update until it reaches the desired stage. The scheme employed the detection method reported by Kim and Nwanze (2008). During the training phase, the mean and standard deviation scores for all datagrams were computed. A global threshold value is then determined based on these scores and a global tuning parameter. The parameter is chosen by a network monitor based on the traffic characteristics of the system, which is used to adjust the value of the threshold. In the detection stage, a score is computed by counting the number of datagram bytes that fall outside the range defined for each byte.

The physical prototype of the mechanism was implemented using a Gumstix Overo Tide embedded computing platform for evaluation. The results of the evaluation showed that the performance of the scheme is proportional to the training window size, i.e, increasing the training window size will give the scheme a better performance. Although the scheme reduced the training period by automatically activating itself as soon as the window hits a number of datagrams, the robustness of the scheme is improved with an increased size of the training window. Finally, the scheme did not take containment into account.

2.6.3 Limitation

Generally, payload-based detection mechanisms are unable to detect polymorphic worms or a traffic that has been encrypted. The mechanisms also require complex numerical computation during detection phase.

2.7 Worm Containment

Detecting the propagation of network worms without automatic containment does not provide a countermeasure solution in itself. As a result, different containment methods have been used by previously reported research work, which can broadly be categorised into slowing down worm infection and blocking (Li et al., 2008). Slowing down worm infection is an approach used to reduce the speed of worm propagation to give more time for human countermeasures to be applied. This is the method used by Williamson

(2002) and rate limiting techniques (Wong et al., 2006). The speed of worm propagation is often so high that it is impractical for human intervention to effectively take place before a high proportion of the vulnerable population is infected. This makes slowing down worm infection an unsuitable technique for fast scanning network worms. Blocking is to isolate network traffic or a host that shows worm-like behaviour from the network to prevent further infection. Blocking can be implemented at the network and data link levels based on the address of the infected host or datagram content (Li et al., 2008).

2.7.1 Network Level Blocking

Network layer devices provide traffic blocking capabilities using access control lists (ACLs), which are commands that control the movement of datagrams through an interface of a network layer device (Belhaouane et al., 2015). The datagram filtering process assists in restricting host access to a network by preventing traffic originating or destined to the host (Chate and Chirchi, 2015). The datagram filtering decision is made based on header information such as source and destination IP addresses and source and destination ports. This is the method used by most of the reported detection system that used traffic blocking to stop the propagation of an identified worm infection. These include the techniques reported by Jung et al. (2004), Weaver et al. (2004), Whyte et al. (2005) and Shahzad and Woodhead (2014b).

The drawback of traffic blocking, particularly at router level, is that the internal network hosts are still open to worm attack, because the local infected host can still send worm datagrams destined to vulnerable hosts within the network segment.

2.7.2 Data Link Level Blocking

At the data link level, network switches (managed) provide access control capabilities using MAC ACLs, which control network traffic based on fields in Ethernet MAC and VLAN headers (Bierman et al., 2013). This can be used to block network traffic based on the source MAC and destination MAC addresses, Ethernet type field values and VLAN identifiers. This capability of isolating a single host in a local network segment has the

potential to block the spread of a worm infection within a local network segment as well as to remote hosts.

MAC-based ACLs are supported by most of the available managed network switch products such as the Cisco Catalyst 4500 series (Cisco, 2016), 3Com 3CDSG10PWR (3Com, 2016), TP-LINK Tx series (TPLINK, 2016) and D-Link DGS-3000 series (DLINK, 2016). Although vendors are trying to incorporate self-defending features in their products such as Cisco Network Admission Control, which redirects suspicious traffic to a honey pot or blocks it using a firewall (Kumar et al., 2016), data link level switches are not equipped with an automatic worm detection or containment capability. Although MAC ACLs can be used to block an identified infected host, the reported worm countermeasure techniques rather employed the network level blocking technique. Thus, it is desirable to explore the use of an automatic data link level access control method for worm infected hosts.

2.8 Testing Environment

To fully understand the propagation behaviour and infection patterns of fast scanning network worms, there is a need for a safe and convenient environment that is isolated from the Internet in order to analyse the behaviour of this malicious software. Large scale network worm outbreak scenarios are difficult to simulate due to the complexity and resources required in setting up a controlled environment for worm propagation and countermeasure testing (Floyd and Paxson, 2001). The approaches that are mainly used by security researchers to test the effects of worms and other network attacks are mathematical models (Fei et al., 2009), simulation (Floyd and Paxson, 2001), emulation (Calheiros et al., 2010) and virtualisation systems (White and Pilbeam, 2010). Mathematical models can provide a high level of scalability, but they are not accurate for Internet worm simulation. This is due to their limitations in modelling some features of the Internet such as topology, heterogeneity, dynamism, traffic and bandwidth routing and network congestion (Perumalla and Sundaragopalan, 2004).

2.8.1 Simulation Systems

Simulation systems use tools and processes to imitate and model a real network environment. Simulation is an accepted and widely used technology in studying the epidemic of network worms; example simulation tools include NS-3 and SSFNet. The NS-3 simulator (Henderson et al., 2008) is an open-source simulation system that has been developed and widely used in research and education relating to computer networks and the Internet. It provides features including support for virtualisation software, scalability and modularity. The Scalable Simulation Framework Network (SSFNet) simulator (Yoon and Kim, 2009) was developed with various network simulation applications and topologies, traffic and scalability. To enhance scalability and provide ease of use and distributed network simulation, Riley (2003) developed the Georgia Tech Network Simulator (GTNetS). Another powerful software simulation package is the Optimized Network Engineering Tool (OPNET). OPNET (Chang, 1999) is capable of simulating a large range of communication systems from a single LAN to global satellite networks and can be used for discrete event simulations. Despite advances in using parallel/distributed execution capabilities to develop datagram-level simulators, researchers have been unable to achieve a simulated network size similar to that of the IPv4 address space (Floyd and Paxson, 2001). This is due to the processing and memory requirements needed (Perumalla and Sundaragopalan, 2004). Some simulation systems use a Finite State Machine (FSM) to represent a network node such as the Internet Worm Simulator (IWS) and Parallel Worm Simulator (PWS). Thus, these provide a high level of scalability and less resource requirement than datagram level simulators. IWS (Tidy et al., 2015) achieved a scale to the size of IPv4 address space and PWS (Wei et al., 2005) provides good Internet topology at the autonomous system level.

Generally, simulation systems have limitations in modelling heterogeneity, topology and the granularity of the Internet. They also have limitations in scale, incur high processing and memory requirements and cannot model the full range of operating system features (Perumalla and Sundaragopalan, 2004).

2.8.2 Testbeds

Testbeds are another widely used technology that utilise real applications and operating systems in an emulated or virtualised environment. The challenges in developing testbeds are re-usability of experiments from a baseline, preventing malware from escaping out of the controlled environment and a trade-off between complexity and scale (van Heerden et al., 2013).

2.8.2.1 Emulation Systems

Emulation (Calheiros et al., 2010) takes the properties of a system and reproduce them with a different type of system. This allows the use of computing nodes and network links to form a system with more emulated elements than real elements. A node in the emulation system represents a real host running in the form of software. To achieve a large-scale emulation, multiple nodes can be instantiated on a single physical machine. Emulation systems have the characteristics of simulation and real world systems and they have the potential to achieve high fidelity, scale and effectiveness (Perumalla and Sundaragopalan, 2004). Emulab (Hibler et al., 2008) and DETERLab (Mirkovic et al., 2010) are some of the emulation systems which have been used by security researchers in the past. Emulab (Hibler et al., 2008) was developed at the University of Utah to provide researchers with an environment suitable for network and operating system testing.

Emulab enables users to develop their experiments with a set of PCs connected in a customised topology via a graphic user interface and then load the desired operating system on to the PCs for experimentation. The DETERLab (Mirkovic et al., 2010) enables user access to a specified set of nodes with custom operating systems. The DETERLab also allows experimentation with network topology using switches, firewalls, physical nodes, and offers attacks and defence tools. However, DERTERLab has limited physical resources and scale for worm outbreak experiments and other network attacks (Murillo and Duarte, 2013).

Emulation systems are generally better than simulation systems in effectiveness because

they allow for interaction with the operating system and serve as a compromise between simulation and real world systems. Nevertheless, emulation systems such as DETERLab and Emulab have limited scale of experiments and physical resources available for users; therefore, they are not suitable for large scale experimentation of worm propagation (Perera et al., 2013). Additionally, emulation systems have performance drawbacks because each instruction on the guest system has to be translated by the software bridge for the host system (White and Pilbeam, 2010).

2.8.2.2 Virtualisation Systems

Virtualisation systems present a platform that is more able to perform realistic tests because it offers a more realistic way of presenting the technology and applications (Perumalla and Sundaragopalan, 2004). Virtualisation systems use the technique of separating resources and services from the underlying physical delivery to form an environment with virtual machines and other network infrastructure on a host. Virtualisation systems provide speed performance which is better than emulation systems because the guest hosts can directly access the physical host hardware (White and Pilbeam, 2010). Some virtualised testbeds includes V-NetLab (Sun et al., 2008), Vise (Årnes et al., 2006), vGround (Jiang et al., 2006) and VMT (Shahzad et al., 2013).

V-NetLab (Sun et al., 2008) is a virtualised testbed that comprises virtual machines interconnected using hubs and switches that are implemented in software. V-NetLab utilizes network virtualisation at the data link layer in order to allow for the re-use of the same set of IP addresses in different virtual networks. V-NetLab is also designed to enable the virtual networks to be accessed remotely without the need for physical access to the hardware. ViSe (Årnes et al., 2006) is a virtualised platform developed to test malware attacks against a range of operating systems and evaluate them using intrusion detection systems. The testbed contains 10 operating systems and 40 exploits (both local and remote) against the programs running on the operating systems. vGround (Jiang et al., 2006) is an environment that comprises three virtual enterprise networks connected by three virtual routers capable of hosting hundreds of virtual machines. VMT (Shahzad

et al., 2013) is a virtualised network testbed developed for zero-day worm analysis and countermeasure testing. The testbed comprises four enterprise networks with a few hundred virtual machines.

Virtualisation systems require relatively high computational resource in order to host a large number of virtual nodes, however, they have the potential to achieve high fidelity and effectiveness as noted by Perumalla and Sundaragopalan (2004). Perumalla and Sundaragopalan (2004) also noted that virtualised systems are not limited to scalability at the expense of an amount of computation power. Furthermore, White and Pilbeam (2010) noted that virtualisation systems provide speed performance better than emulation systems because the guest hosts can directly access the physical host hardware.

2.9 Methodology of Worm Detection

The reviewed network anomaly-based detection systems used datagram header information, payload information or both to detect the presence of worm infection and propagation. The techniques used by the detection systems to determine abnormal behaviour in a network can be classified as statistical analysis, connection status, traffic correlation and destination addresses visited.

- *Statistical analysis*: This is the use of network traffic information to statistically characterise the normal behaviour of a network. This aspect requires a training phase in order to model the normal activity of a network, and a detection phase to determine a deviation in the normal behaviour observed during the training phase. This is the technique used by payload-based detectors and some datagram-header based detectors such as the techniques reported by Mahoney and Chan (2001), Comar et al. (2013) and Wang and Stolfo (2004). Therefore, for each datagram inspected during the detection phase, the system must measure the statistical score of the datagram and then compares the result with a normal profile. However, the dependence of the technique on the need for a training dataset increases management overhead due to difficulties involved in collecting,

filtering and processing the data, especially for detection systems that use payload information to identify worm datagrams (Smith et al., 2009). Additionally, the detection system has to be trained for a period of time to understand a normal network profile, which hinders deployment and leaves the network vulnerable to attack (Kim et al., 2012). Finally, most such detection systems (e.g. those reported by Mahoney and Chan (2001), Comar et al. (2013) and Wang and Stolfo (2004)) require complex numerical computation (Jyothsna et al., 2011; Liao et al., 2013) to identify worm datagrams in the detection phase and incur a high rate of false positives, especially when new systems are introduced into the network or the normal behaviour of the network changes for some other reason (Li et al., 2008).

- *Connection status:* A number of targets contacted by a scanning worm are inactive IP addresses, which result in a number of failed connection (TCP based scanning). This is the technique used by Jung et al. (2004), which has limitations such as the inability to detect UDP-based worms because there is no return traffic from the destination to the source which indicates success or failure. Additionally, if a worm uses hit-list scanning or a peer-to-peer network interaction log, the failed connection rate will significantly reduce. An improved way of detecting worm activity using connection status is to consider all connections to be a failure until a response is received, i.e, a datagram is marked suspicious if it does not match a set of existing and successfully-established connections, which reduces the dependence of the system on failed connection rate. This is the technique used by Weaver et al. (2004), but the technique incurs the same limitation of inability to detect UDP-based worms. Generally, detecting worms using connection status requires enormous resources to keep track of distinct connection and host information, which is resource consuming and therefore not suitable for use in large networks (Li et al., 2008).

- *Traffic correlation*: A vulnerable host is infected by a worm after receiving an infectious datagram from another infected host. The idea here is an infected host

that is transmitting datagrams must have received an infection from another host. Therefore correlating inbound and outbound traffic enables the detection of worm scanning and propagation. This is the technique used by Gu et al. (2004) and Qin et al. (2004). The technique correlated inbound and outbound ports to detect fast scanning network worms because in most cases an infected host transmits datagrams that are destined to the same port. This technique is promising in detecting worm scanning but limited in detecting worms that used multiple attack vectors because the inbound and outbound traffic differ (Stafford and Li, 2010).

- *Destinations visited*: A scanning worm transmits datagrams to wide range of different destinations in order to propagate widely. Therefore the pattern of visiting new destinations by a normal host differs from that of a scanning worm. Therefore, monitoring frequent contact to new destinations enables the detection of a scanning worm. This is the technique used by Williamson (2002) and Li and Shad (2014). The techniques monitored the rate of first contact to different destinations within a window of time. Another approach to monitoring destination addresses is to utilise the DNS activities of hosts to detect worm propagation. A scanning worm transmits datagrams using numeric IP addresses without the need for DNS resolution, which deviates from the normal behaviour of accessing the vast majority of legitimate publicly available services. This is the technique used by Whyte et al. (2005) and Shahzad and Woodhead (2014b). Generally, monitoring destinations visited is promising in detecting worm scanning and propagation including events where a worm uses a hit-list or topological information because the worm must contact new destinations in order to spread the infection (Li and Shad, 2014; Wang et al., 2014).

Having identified the classes of techniques used to determine abnormal worm-like behaviour in a network and their limitations, it is clear that traffic correlation and monitoring the destinations visited provide a better coverage and performance in detecting scanning worms than using statistical analysis and connection status. This is due to their ability to detect both TCP-based and UDP-based worms and effectively provide an early detec-

tion of worm scanning and propagation. Furthermore, monitoring destinations visited provides more detection coverage than traffic correlation because if properly used, it can detect worms that used multiple attack vectors. Therefore, it is desirable to further study the use of destination contacts to detect scanning worms.

2.10 Research Issues

Having discussed anomaly-based detection systems, this section identifies some research issues based on the implementations of the reported mechanisms and the techniques employed to detect and contain worm propagation. The section also presents identified issues that relate to the virtualised testing environments reported. The limitations are:

- *Deployment delay and resource consumption:* The survey identified that the reported worm detection techniques either used statistical analysis, traffic connection status, traffic correlation or destination contacts to determine abnormal behaviour in a network. Techniques that used statistical analysis of header or payload content such as those reported by Mahoney and Chan (2001), Wang and Stolfo (2004) and Kim et al. (2012) incur management overhead and hinder deployment due to the need to collect data for training (Jyothsna et al., 2011; Liao et al., 2013). The techniques also incur a high rate of false positives, especially when new systems are introduced into the network or the normal behaviour of the network changes (Li et al., 2008). The techniques that used connection status such as those reported by Jung et al. (2004) and Weaver et al. (2004) cannot detect UDP-based worms and require significant resources to keep track of distinct connection and host information, which makes them unsuitable for use in large networks (Li et al., 2008).

- *Uniform and static threshold*: The reported worm detection mechanisms used a uniform and static threshold for all hosts in a network irrespectively of differences in their connection behaviours and the services they offer. Hosts in a network exhibit divergent behaviours and therefore require different thresholds. For example, a

server host tends to respond to client requests and runs more applications and other network services or daemons that may connect to different destination IP addresses. On the other hand, a client host used to access the Internet makes connections in a pattern different from a server host. Therefore, the transmission behaviour of a server host is different from a client host. The rate of Internet usage also differs by time of day, i.e. in most cases, Internet usage is significantly higher during working hours (Citicaka, 2014). This is the peak period for hosts in a network to generate larger traffic volume due to higher usage. As a result, it is anomalous for a host to make frequent connections to different destinations during a quiescent period. Therefore, thresholds applied to hosts during active and quiescent periods should differ in order to reduce the possibility of false positives during peak periods and enforce strict security measures during the quiescent period. A quiescent period gives scanning worms enough time and bandwidth to propagate across networks and the Internet, and there is also limited or no human intervention. Thus, due to the different behaviour of hosts in a network and varying rate of traffic by the time of day, the choice of having different and dynamic thresholds for network hosts will potentially improve the performance of a detection mechanism.

- *Suppression of worm infection*: The next step after detecting a worm infection is to slow down the propagation or automatically block traffic from the infected source using address blocking or content blocking techniques. Address blocking is to drop any traffic from a host identified to be infected while content blocking drops datagrams that contain the worm signature. The reported worm countermeasure techniques that use a containment technique (e.g. those reported by Williamson (2002), Wang and Stolfo (2004), Weaver et al. (2004), Whyte et al. (2005) and Shahzad and Woodhead (2014b)) blocked a host identified as infected at the network boundary router and at the network level, i.e, access control for the infected host is implemented for outbound traffic at the layer 3 level, which contained the worm infection within a network segment. Using network level access control on outbound traffic typically only slows down the infection rather than stopping it,

because it leaves internal network hosts vulnerable to further worm attack and allows inbound worm traffic into the network. Thus, it is desirable to explore the use of data link level access control to block an infected host. Additionally, upon detecting a worm infection, it is also desirable to implement access control for inbound worm traffic into a network segment.

- *Limited features for worm experimentation in testbeds*: The review has identified that virtualisation systems have the potential to achieve high fidelity, scale, effectiveness and speed at the expense of an amount of computation power, in order to host a large number of virtual nodes (Perumalla and Sundaragopalan, 2004; White and Pilbeam, 2010). However, the reported virtualised testing environments (e.g. V-NetLab (Sun et al., 2008), Vise (Årnes et al., 2006), vGround (Jiang et al., 2006) and VMT (Shahzad et al., 2013)) have limited scale of experiments and provide no support for background traffic. In addition, vGround (Jiang et al., 2006) is platform dependent, i.e., it supports Linux-based worm experimentation and VMT (Shahzad et al., 2013) does not provide a management interface for re-usability of experiments.

The identified limitations and shortcomings of previously published research can be summarised as follows;

- *A delay between deployment and start of the detection process:* This is the time needed for data collection to train a detection scheme to learn the normal profile of a network before the start of a detection process. This is the technique used by machine learning based detection schemes and payload-based detection schemes.

- *A high resource consumption:* This is the use of large system resources such as memory to keep a record of every connection and host information during the detection process. This is the technique used by schemes that use first or failed destination contact to identify worm infection.

- *A uniform and static threshold for all network hosts:* This is the use of the same

threshold for network hosts to determine an excess in the level of anomalies exhibited by a host. This technique is used by most detection schemes.

- *Suppression of worm infection rather than stopping it by countermeasure schemes:* This is the process of blocking an identified worm infection by a countermeasure scheme. The detection schemes that uses rate limiting, failed contacts, DNS activities and correlation of inbound and outbound traffic suppress worm infection by containing it at the network boundary router or network level, which allow the infection to continue within the local network.

- *Limitation in scale of experiments:* The reported virtualised network environments have limited scale of experimentation. Large scale of experimentation has the potential for more fidelity and effectiveness in worm experimentation (Perumalla and Sundaragopalan, 2004).

- *Lack of provision for background traffic:* The reported virtualised network environments do not provide support of background traffic, which is essential in assessing the performance of a detection schemes in terms of false positives.

- *Limitation in re-usability of experiments and operating system dependency:* The reported virtualised network environments are not equipped with tools that facilitate managing and repeating experiments using either the same or different set up. Some of the environments support UNIX-based worm experimentation only.

2.11 Research Questions and Objectives

Having identified some limitations associated with the previously reported worm detection systems and testing environments, the following research questions were established:

> *Is it possible to develop an effective worm countermeasure scheme that has a lower delay in comparison to the existing open source schemes?*

Additionally, in order to empirically assess whether the above research questions have been answered, and to what extent, the following research tools need to be developed:

1. *An improved and suitable testing environment in comparison with the existing virtualised testing environments.*

2. *The use of previous and contemporary malware to assess the effectiveness of the proposed solution.*

To address the above questions, the research work needs to achieve the following main objectives:

1. *Development of a more accurate worm detection method without compromising temporal performance in comparison with the existing schemes.*

2. *Use of a containment mechanism that blocks an identified worm infection from spreading within a local network and to remote hosts.*

Generally, traffic correlation techniques such as that reported by Gu et al. (2004) and monitoring destinations techniques such as those reported by Whyte et al. (2005) and Shahzad and Woodhead (2014b) are promising in detecting worm scanning and propagation. The techniques also have the potential to exhibit low false alarms and detect worms that use multiple attack vectors if carefully implemented. Therefore it is desirable to explore the use of these techniques to identify the presence of a worm. Thus the research work seeks to

- Improve the traffic correlation and DNS-based detection schemes and apply the techniques on server and client hosts in a network respectively.

- Use a dynamic threshold scheme for network hosts due to varying rate of traffic by time of the day.

- Use access control to block an identified infected host from transmitting datagrams and a network access control to block inbound worm datagrams.

These serve as the basis for further research analysis and development of an improved anomaly-based detection and containment mechanism. Chapter 3 discusses the details

of the improved detection and containment mechanism. Chapter 4 discusses the environment developed for the analysis of worm propagation and testing countermeasure mechanisms. Chapter 5 discusses the details of the evaluation programme developed for worm propagation experiments and countermeasure testing.

2.12 Summary

The early part of this chapter presented a brief overview of malware and network worms. The chapter then presented a range of previously reported worm detection and containment schemes. These schemes are broadly categorised as datagram-header based anomaly detection systems and payload-based anomaly detection systems. Datagram-header based anomaly detection schemes use protocol header information to detect anomalies. Payload-based detection schemes use the content of datagram payloads to observe anomalies in byte frequency. The chapter also presented a range of reported techniques used by security researchers for worm propagation and countermeasure testing. The chapter then presented some identified research issues, which serve as the basis for the development of an improved worm detection and containment mechanism and a suitable environment for empirical evaluation of countermeasure mechanisms. Finally the chapter presented the research questions that the remainder of this thesis document endeavours to address.

Chapter 3

THE NEDAC SCHEME

3.1 Introduction

This chapter presents the worm detection and containment scheme proposed based on the research gaps identified in the survey of related work. The chapter is set to answer the research question outlined in Chapter 2and therefore attain the two objectives set out in Section 2.10. Section 3.2 presents a brief background on worm detection and containment schemes. Section 3.3 presents an overview of the approach used for the proposed worm countermeasure mechanism. Section 3.4 presents the design of the detection and containment techniques used in the countermeasure mechanism. Section 3.5 presents a comparative analysis of the proposed worm countermeasure with previously reported worm detection schemes while Section 3.6 summarises the chapter.

3.2 Background

According to Chen et al. (2014), fast scanning network worms are a virulent class of network worms that exploit wormable vulnerabilities and immediately propagate pervasively by launching attacks against Internet hosts and services, thereby posing a major threat to the security of the Internet. Most individual scanning worms exploit the same vulnerability throughout their propagation and try to propagate the infection to more vulnerable hosts. This propagation behaviour of contacting a significant number of new destination addresses characterises a scanning worm. In addition, the unique traffic

pattern of rapidly connecting to new distinct destination addresses in order to infect a wide range of vulnerable host serves as a fundamental behaviour of fast scanning network worm propagation. Thus, it is desirable to focus on this fundamental behaviour in order to detect fast scanning network worms because it makes evasion very difficult (Li and Shad, 2014). The behaviour of contacting new destinations by probing pseudo-random IP addresses is inevitable by a worm that is seeking to propagate, and therefore is important for a detection mechanism to look for worm behaviour in the way new destinations are contacted (Li and Shad, 2014). The active probing of IP addresses over the Internet makes it possible for a network-based detection system to detect the worm behaviour.

The survey of related work presented in Chapter 2 has shown that detection schemes that use correlation of inbound and outbound traffic and those that monitor destinations visited by hosts in a network have the ability to identify both TCP-based and UDP-based worms, unlike the techniques reported by Jung et al. (2004) and Weaver et al. (2004), which have the potential to detect TCP-based worms only. The schemes also exhibit low false alarms compared to the techniques reported by Mahoney and Chan (2001), Wang and Stolfo (2004) and Comar et al. (2013), and if carefully implemented, can detect worms that use multiple attack vectors (Stafford and Li, 2010).

Among the reported schemes that used correlation of inbound and outbound traffic and monitoring destinations contacted by hosts are the schemes reported by Gu et al. (2004), Qin et al. (2004), Whyte et al. (2005) and Shahzad and Woodhead (2014b) respectively. The limitations of these schemes are their limited ability to accurately differentiate worm traffic and benign traffic that behaves in a similar way to worm traffic and suppression of identified worm infection rather than stopping it completely. This is because the schemes maintained a count of worm activities for each host and therefore during worm propagation, benign traffic that exhibits worm behaviour is marked as suspicious and counted as worm datagrams. This makes it difficult to accurately apply a countermeasure solution due to the relatively high probability of false positives.

Thus, it has been identified that modifying the traffic correlation and DNS-based detec-

tion schemes, and applying the techniques to server and client hosts respectively, has the potential to improve the accuracy of the detection mechanism. Furthermore, applying a data link and network layer access control methods upon identifying a worm infection, for outbound and inbound worm traffic respectively, has the potential to block an infected hosts from spreading the infection within its local network segment and to/from remote hosts. These techniques have the potentials to provide a powerful combined detection and containment solution against fast scanning network worms. The proposed scheme is also likely to provide a better containment solution that blocks an identified worm infection from spreading within a local network and to/from remote hosts.

3.3 Approach

The proposed worm detection and containment scheme, termed NEDAC, is a cross-layer mechanism that uses a network layer detection system and a data link layer containment system, with a connection maintained between the two systems to enable continuous data transmission. The detection system uses two different techniques to detect worm infection from client hosts and server hosts respectively. The detection system maintains a list of server IP addresses in order to differentiate client and server hosts in a network. Client hosts are defined as network hosts which typically consume Internet services (e.g. workstations, laptops, tablets, smartphones, etc.) while server hosts are network hosts used to serve client requests (e.g. web servers and email servers).

The detection system keeps track of inbound and outbound TCP and UDP datagrams and ARP request datagrams for a window of time with value T to determine worm behaviour that exceed a threshold with value V (for TCP and UDP datagrams) and W (for ARP request datagrams). The threshold is a maximum allowable count of worm behaviours a host can send before T has elapsed. However, the containment system uses the MAC address of a host that has been identified (by the detection system) as infected to block all traffic originating from the host.

A threshold applied to a host in a network also depends on the time of day, i.e., active

and quiescent network periods are defined. An active period is considered to be a period during which hosts in a network generate a large volume of Internet traffic to several destinations. This is normally during business hours and working days, which vary by geographical location and country. A quiescent period is when hosts in a network are less active or idle; this is normally during non-business hours. As a result, the thresholds applied during the two periods vary because Internet usage during off-peak period reduces by 60% as examined by Citicaka (2014). The reduction in Internet usage was also observed during live network traffic collection at University of Greenwich computer laboratories. Although backups and batch jobs typically run out of normal business hours, the number of destinations visited during backup are relatively low and consistent. Therefore, before applying a threshold to a client or server host, the detection system determines the time of day and then applies the appropriate threshold.

3.4 NEDAC Architecture

The architecture of the NEDAC detection system is presented in Figure 3.1. The detection part of NEDAC comprises a Traffic Classification (TC) engine, a Threshold Determination (TD) engine, three detection sub-systems namely ARP-based detection (AD), server host worm detection (SWD) and client host worm detection (CWD), an exempt table and three caches to keep track of connection and host information; namely input/output (I/O) cache, resolution and no-resolution caches. The containment part of NEDAC comprises the containment system and a record that keeps details of contained hosts and their time stamps.

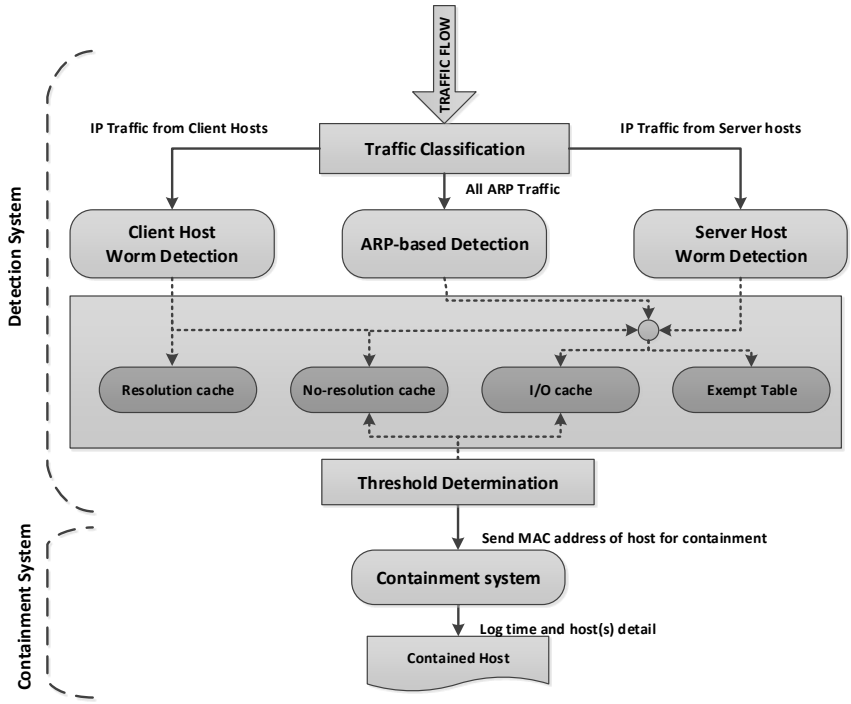


Figure 3.1: NEDAC worm detection architecture

The TC engine is used to classify TCP or UDP traffic that is destined to or originates from client and server hosts in a protected network and then passes the result to the CWD or SWD sub-system, while ARP traffic is passed to the AD system. The exempt table elements are IP address and port number combinations that are exempt from the detection system for known systems that legitimately communicate using IP addresses directly. The resolution cache is used to keep record of all DNS resolutions made by hosts in a network, i.e, IP address of host that make a resolution and the resolved IP addresses. The no-resolution cache is used to keep record of hosts that send datagrams without a prior DNS resolution. The TD is used to assess the level of worm behaviour exhibited by a host in order to determine counts above the relevant threshold.

The AD sub-system identifies ARP requests (from all network hosts) to internal inactive local IP addresses and then uses the TD to assess the level of worm behaviour. If a threshold with value W is exceeded, the TD invokes the countermeasure. The CWD and SWD sub-systems process the received TCP and UDP traffic from TC and then assess the behaviour of the client and server hosts in the network respectively. The

CWD and SWD sub-systems also use the TD to determine whether the level of worm activity observed from a host passes a threshold with value V, and if true, then the TD invokes the countermeasure. The CWD and SWD sub-systems are made up of a Traffic Inspection Module (TIM) and a Detection Module (DM).

- **TIM:** This module receives traffic flow from the network and classifies the traffic into inbound and outbound. The module extracts the required header information of each inbound TCP or UDP datagram and records the information in the I/O cache. Outbound datagrams are forwarded to the Detection Module.
- **DM:** This module receives outbound traffic and then detects worm infection and propagation by monitoring datagram header information. The module then uses TD to pass information to a countermeasure solution upon detecting an infected host in a network.

Figure 3.2 shows the flow diagram of the NEDAC mechanism with all the detection sub-systems. The pseudo-code of the NEDAC mechanism is presented in Appendix A. Details of the CWD and SWD techniques are set out in sections 3.4.1 and 3.4.2, while details of the data link containment system are set out in Section 3.4.3.

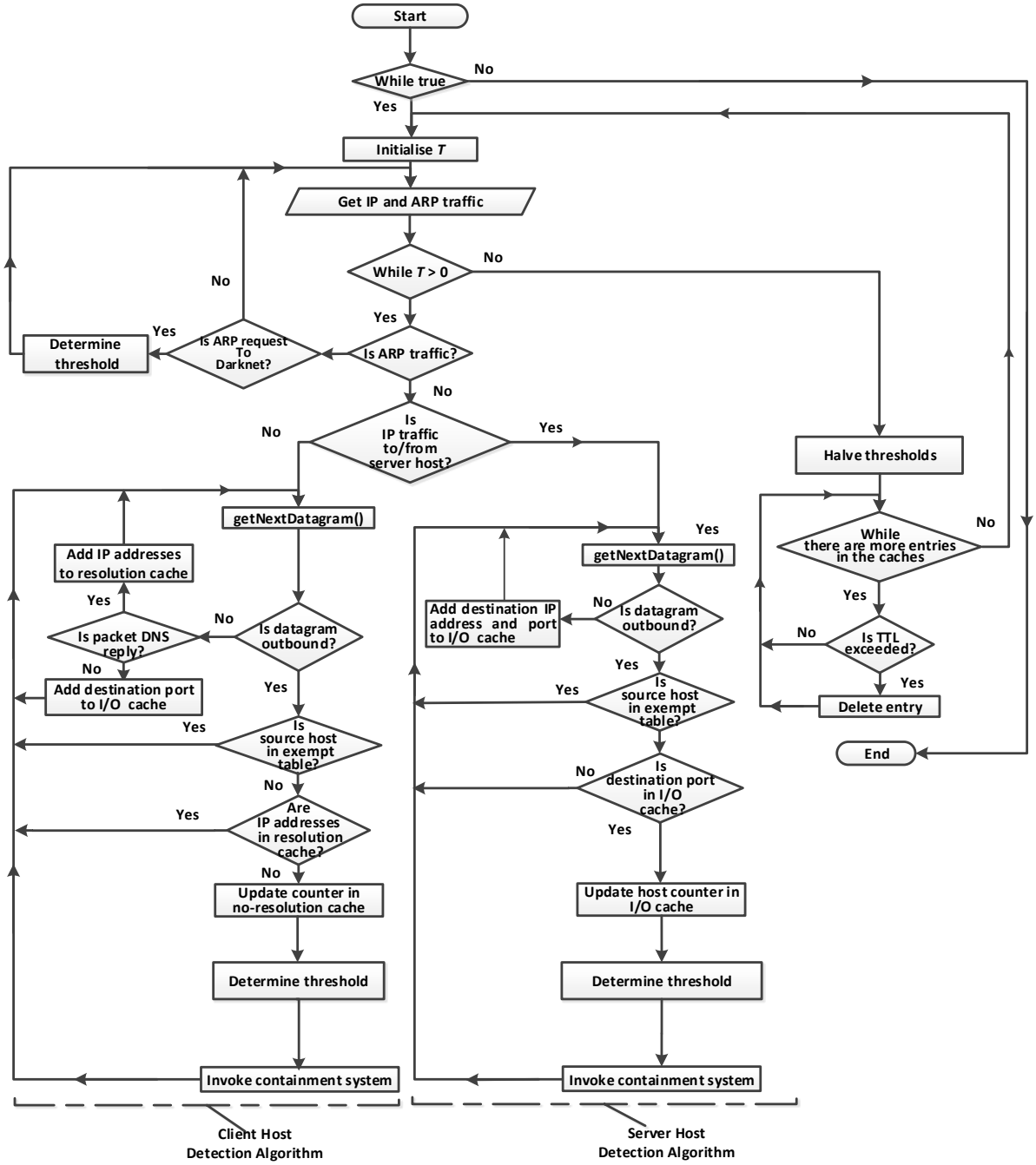


Figure 3.2: NEDAC detection system

3.4.1 Client Host Worm Detection

The CWD technique is detailed using the activity diagram presented in Figure 3.3.

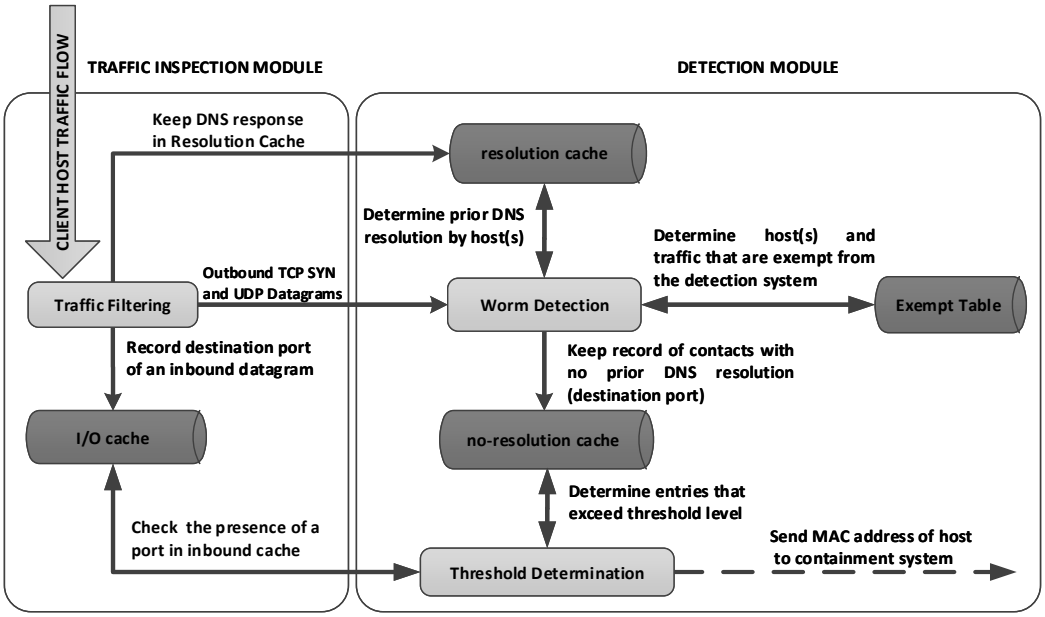


Figure 3.3: NEDAC worm detection architecture for client hosts

The TIM comprises a Traffic Filtering engine and has access to the I/O and resolution caches. The Traffic Filtering engine is used to classify inbound and outbound datagrams and passes the record to the I/O or resolution cache and the Worm Detection engine in the DM respectively. The I/O cache is used to keep the destination port of all inbound datagrams (excluding DNS record) while the resolution cache is used to keep the IP address of a host that made a DNS resolution to a destination and the resolved address. The DM has a Worm Detection engine and access to the TD engine, exempt table and resolution and no-resolution caches.

The CWD algorithm (Figure 3.2 left) uses the Worm Detection engine in DM to compare the source IP addresses and ports to entries in the exempt table and the resolution cache. This is to determine a white-listed host and a prior DNS resolution for an outbound datagram respectively. If there is a miss in both, the algorithm records the destination port of the outbound datagram and the internal source host IP address in the no-resolution cache, increments the counter associated to the destination port for such hosts, and then uses the TD engine to determine whether the entry exceeds a threshold. This continues for datagrams destined to distinct destination port that exhibit abnormal behaviour and upon exceeding a threshold with value V, the TD engine invokes the data

link containment system. Then the TD checks for the presence of the suspect port in the I/O cache and if there is a hit, an access control list (ACL) is updated to block all inbound datagrams, at the network boundary router, destined to the suspect port in the network segment. A time-to-live (TTL) is provided for entries in all the caches, which is the lifespan for datagrams information held in the caches. The default TTL value for DNS (86400 seconds) (Elz et al., 1997) is applied to the resolution cache and 64 seconds, which is the TCP and UDP TTL value for most operating systems (Lippmann et al., 2003), is applied to the no-resolution and I/O caches. Furthermore, the TD decrements the counters in the no-resolution and I/O caches by half after the expiration of the timing window of T, and then checks all caches to identify and remove entries with an expired TTL.

3.4.2 Server Host Worm Detection

The SWD technique is detailed in the activity diagram presented in Figure 3.4.

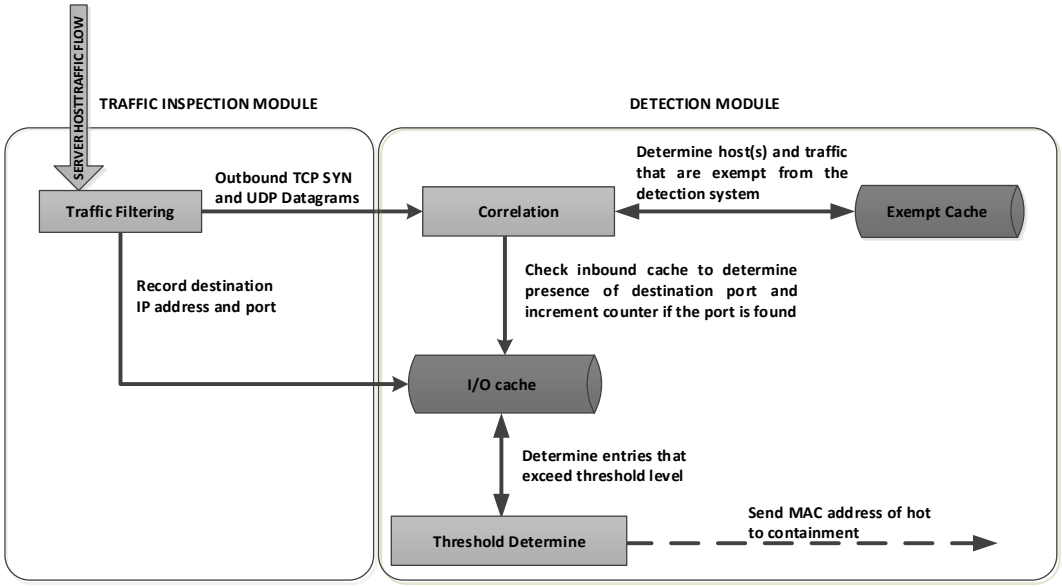


Figure 3.4: NEDAC worm detection architecture for server hosts

The TIM of the SWD comprises a Traffic Filtering engine that classifies inbound and outbound traffic and then sends the datagrams to the I/O cache and Correlation engine respectively in the DM. The DM has a Correlation engine and access to the TD engine, exempt table and the I/O cache. The Correlation engine determines worm behaviour

from server hosts by associating inbound and outbound traffic information.

The SWD algorithm (Figure 3.2 middle) monitors traffic to and from server hosts with the aid of the Traffic Filtering engine in TIM. The algorithm uses the Traffic Filtering engine to record the destination port and IP address of an inbound datagram in the I/O cache and passes outbound datagram information to the correlation engine. To detect worm behaviour, the algorithm checks the presence of the destination port of outbound TCP SYN and UDP datagrams in the I/O cache. The behaviour is deemed suspicious if the destination port is found in the I/O cache, and therefore a counter is incremented. If the outbound datagram is UDP, an additional verification of the destination IP address is made to determine a reply UDP datagram, i.e., if the destination IP address does not match the IP address recorded in the I/O cache, the behaviour is marked as suspicious and therefore a counter is incremented. SWD then uses TD to determine any count above threshold, invoke both countermeasure techniques and decrement and clear entries in the same manner as CWD.

3.4.3 Containment System

The next stage after detecting the presence of a worm is to counter the attack. Containment is to prevent the spread of a worm from an infected host to vulnerable hosts. The containment technique used by NEDAC is a data link access control method in order to block an identified infected host from sending traffic within the local network, and thereby also to remote external targets. Data link level containment using a switch has the potential to offer a defence by isolating an identified infected host and therefore stops a worm propagation to local and remote hosts. If this can be achieved, it will represent an improvement over the network level access control used by previously reported worm countermeasure schemes, which only stop worm traffic destined to remote vulnerable host while leaving internal vulnerable hosts open to worm attack.

The vast majority of managed network switches support the use of MAC address ACLs to block traffic originating from and destined to a host in a local network. However, switches do not have the capability to detect a worm infection or collaborate in providing

an automatic countermeasure solution for worm propagation. Thus, the containment system of NEDAC is an automated system that has been designed to use the MAC address access control method on a switch upon receiving a request for such service from the detection system. The containment algorithm of NEDAC is detailed in the control flow diagram presented in Figure 3.5. The algorithm receives the MAC address of an infected host from the detection system and then blocks all traffic originating from the host using the MAC address access control method on the relevant switch. Then the system logs the host details and time stamps.

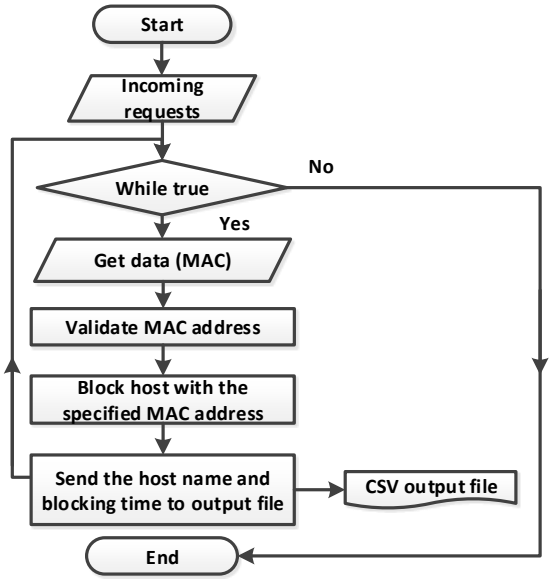


Figure 3.5: NEDAC containment system

3.5 NEDAC Comparative Analysis

This section presents a comparative analysis of NEDAC and the previously reported worm countermeasure schemes that used traffic correlation and destination contacts to determine worm behaviour in a network. The chosen schemes are the DSC technique reported by Gu et al. (2004) and the DNS-based rate limiting techniques reported by Whyte et al. (2005) and Shahzad and Woodhead (2014b). The NEDAC detection system uses modified versions of the DNS-based and traffic correlation techniques to detect worm behaviour from client and server hosts respectively. The specific design improvements

relating to worm detection and worm containment, provided by NEDAC, are presented in sections 3.5.1 and 3.5.2 respectively.

3.5.1 Detection System

The NEDAC mechanism applies a DNS-based detection technique to client hosts because they mainly consume Internet services, which in most cases requires a DNS resolution. Therefore the absence of DNS resolution by a client host for a significant number of destination contacts, within a short time period, is anomalous. On the other hand, server hosts are used to serve client requests, so server hosts listen for client requests and then respond to the requests. Server hosts do not initiate new Internet connections in most cases and therefore require less DNS resolution than clients, and in some cases, none at all. Therefore, the use of the inbound and outbound traffic correlation technique has a better potential to effectively detect abnormal behaviour for server hosts and is thus employed as a detection technique for server hosts by NEDAC.

The NEDAC detection system also uses ARP request datagrams to identify worm scanning to inactive local IP addresses by both client and server hosts. This further strengthens the detection capabilities of the mechanism in identifying worm infection and propagation because, in most cases, only a malicious activity such as worm scanning sends datagrams to an inactive local IP address in a network.

The NEDAC detection system uses a dynamic threshold policy based on time of day as described in Section 3.3, in an effort to reduce the possible rate of false positives during the active period (when hosts in a network generate large and frequent Internet traffic) of network usage and enforce a more strict policy during the quiescent period.

Thus, the use of specific worm detection techniques for client and server host together with the ARP-based detection component, which is applicable to all hosts in a network, has greater potential to effectively detect worm infection and propagation than the two previously reported detection schemes. Additionally, the detection technique does not require training and therefore resolves the issue of the delay between system commission-

ing and the start of the detection process, and also reduces the overhead of collecting, filtering and processing data to train the system.

Furthermore, the CWD and SWD techniques observe anomalous traffic and maintain a count for distinct destination ports contacted by a client host, unlike the techniques reported by Gu et al. (2004), Whyte et al. (2005) and Shahzad and Woodhead (2014b) that maintained a counter for each host, i.e, in NEDAC a counter C_{ij} is maintained for each *ith* anomalous distinct destination visited by *jth* host. Therefore keeping the count of anomalous distinct destination ports contacted by a host enables NEDAC to accurately identify the destination port used by a fast scanning network worm. This differentiates worm traffic from other legitimate traffic that may exhibit similar behaviour, because the counter associated to the destination port of worm datagrams increases rapidly. It also enables NEDAC to detect worm infection and propagation including those that employ multiple attack vectors, which is a limitation of the DSC (Gu et al., 2004) algorithm as reported by Qin et al. (2004). Generally, this has the potentials to significantly reduce the possibility of false positives and accurately apply a network level countermeasure for identified worm traffic coming into a network. Thus, NEDAC has the potential to achieve the objective of having a more accurate detection mechanism in comparison with the existing schemes.

Finally, the NEDAC detection system uses TTL for DNS, TCP and UDP datagram information maintained in relevant caches. This is to address the issue of high resource consumption due to the need for keeping track of distinct connection and host information during the detection process, which provides the NEDAC detection system with potential for resource efficiency.

3.5.2 Countermeasure System

The NEDAC mechanism uses containment techniques at network and data link levels. Upon detecting a worm infection, the destination port used by the identified worm traffic is used to apply a network level countermeasure by blocking worm traffic coming in to a network. Additionally, the data link level containment system is used to block

all outbound datagrams from a host that has been identified by the detection system as infected. This is achieved by automatically applying an access control for infected host on a network switch. These countermeasure techniques provide NEDAC with the potential to not only block an identified worm infection from an internal network host, but also worm traffic coming into the network from a remote host, which is a combined countermeasure solution to protect internal and remote network hosts.

3.6 Summary

This chapter has presented the design of an improved worm detection and containment scheme termed NEDAC. The NEDAC design uses a cross-layer solution that detects worm infection at the network layer and then contains the infection using data link and network level containment solutions. The detection system of NEDAC applies different worm detection techniques for client and server host in a network and uses ARP request datagrams to inactive internal IP addresses to identify worm infection from both client and server hosts. A comparative analysis of NEDAC and two previously reported detection schemes has been presented to show the advances of NEDAC over the existing schemes. Having developed a countermeasure solution, the research work seeks to develop a safe and convenient environment as a research tool for empirical assessment of the developed worm countermeasure solution. Chapter four presents the details of the environment developed for evaluating NEDAC.

Chapter 4

THE V-NETWORK TESTBED

4.1 Introduction

Having proposed a countermeasure solution for fast scanning network worms, it is desirable to develop an environment that can be used to test and evaluate the proposed solution, with a view to quantifying the achievable improvements. Therefore a safe and convenient environment that is isolated from the Internet is needed.

This chapter presents the design and implementation of a virtualised testing environment that comprises suitable features for worm propagation and countermeasure systems. These include management facilities that simplify the creation of virtual host in large scale, re-usability and tearing down large number of virtual hosts and generation/replay of background traffic. Section 4.2 presents the background to virtualisation and virtualised testbeds. Sections 4.3 and 4.4 present the details of the testbed design and implementation respectively. Sections 4.5 and Section 4.6 present the details of the worm daemon and management scripts provided by the testbed to manage and facilitate worm propagation experiments. Finally, Section 4.8 summarises the chapter.

4.2 Background

Virtualisation (Chiueh and Brook, 2005; Sahoo et al., 2010) refers to the technology used to create an abstraction layer between computer hardware and an operating system and applications running on top of it. This layer provides infrastructural support using low-

level resources to create multiple virtual instances that are independent and isolated from each other, such as a virtual computer, an operating system (OS), a storage device, or computer network resources. Virtualisation consolidates workloads in a way that allows one physical machine to be multiplexed for many different users, which improves the efficiency of a data centre by allowing more work to be done on a smaller set of physical nodes (Hwang et al., 2013). This provides flexibility, portability, re-usability and less resource requirements and is therefore suitable to be used in research work. Virtualisation is most commonly implemented using a hypervisor, i.e., a software or firmware component used to create virtual system resources.

A hypervisor or Virtual Machine Monitor (VMM), which lies between hardware and an operating system, divides system resources, such as CPU, memory and storage into several independent smaller components. Each component known as 'virtual machine' is capable of running an operating system with the support of the CPU as if it is on a physical machine. The main advantage is the isolation provided for each component to operate without affecting others. Hypervisors are divided between Type 1 and Type 2 (Sherman, 2014). A Type 1 hypervisor runs directly on system hardware with each virtual machine running on top of the hypervisor. The Type 2 hypervisor runs on a host operating system, with each virtual machine running on top of the hypervisor. There are numerous hypervisors ranging from open-source hypervisors such as the KVM (Kivity et al., 2007) and Xen (Barham et al., 2003), to commercial hypervisors such as the VMware vSphere (Lowe, 2011) and Microsoft Hyper-V (Velte and Velte, 2009).

The survey of related work presented in Chapter 2 has identified that virtualisation systems have the potential to achieve high fidelity, scale, effectiveness and speed and are therefore suitable for malware experimentation. However, the previously reported virtualised testbeds reviewed in Chapter 2 have limited experimental scale, platform dependency, and provide no support for background traffic and management interface for re-usability of experiments.

Thus a virtualised network environment has been developed to facilitate controlled worm propagation and the testing of countermeasure systems. The testbed, termed Virtualised

Network (V-Network), has been developed with scalability, re-usability, portability and resource efficiency as desirable attributes. V-Network is aimed at studying the infection and propagation patterns of network worms and testing a range of countermeasure systems. V-Network is platform independent, which makes it convenient for UNIX-based or Windows-based experimentation, and is designed with the capability of resetting and re-running experiments from a standard baseline in a controlled environment either using a graphical user interface or command line scripts.

4.3 V-Network Design

The V-Network testbed contains four virtualised enterprise networks that comprise a number of virtual network cells. The virtual network cells contain LANs with a DHCP server for IP address management, a DNS server for name resolution, an NTP server to provide a time synchronization service for the virtual hosts, a logging server to keep a record of worm infection activities and routers for internal routing services. The virtual enterprise networks are connected together to enable data communication and routing services across the internetwork. The design was chosen to study how worm infection spreads across multiple networks that are physically and geographically separated and to facilitate the deployment of countermeasure systems. Figure 4.1 details the logical design of an enterprise network in the V-Network testbed. The V-Network testbed uses the Quagga routing suite (Ishiguro et al., 2007) to provide routing services. The VMware vSphere 5.5 (Lowe, 2011) hypervisor has been used for virtualisation services, which also comprises VMware vCenter Server for remote management of the ESXi servers. VMware vSphere was chosen for the development of the V-Network testbed due to its strong performance in comparison to KVM, Xen and Microsoft Hyper-V in the utilization of CPU, memory disk I/O and network I/O as determined by Hwang et al. (2013).

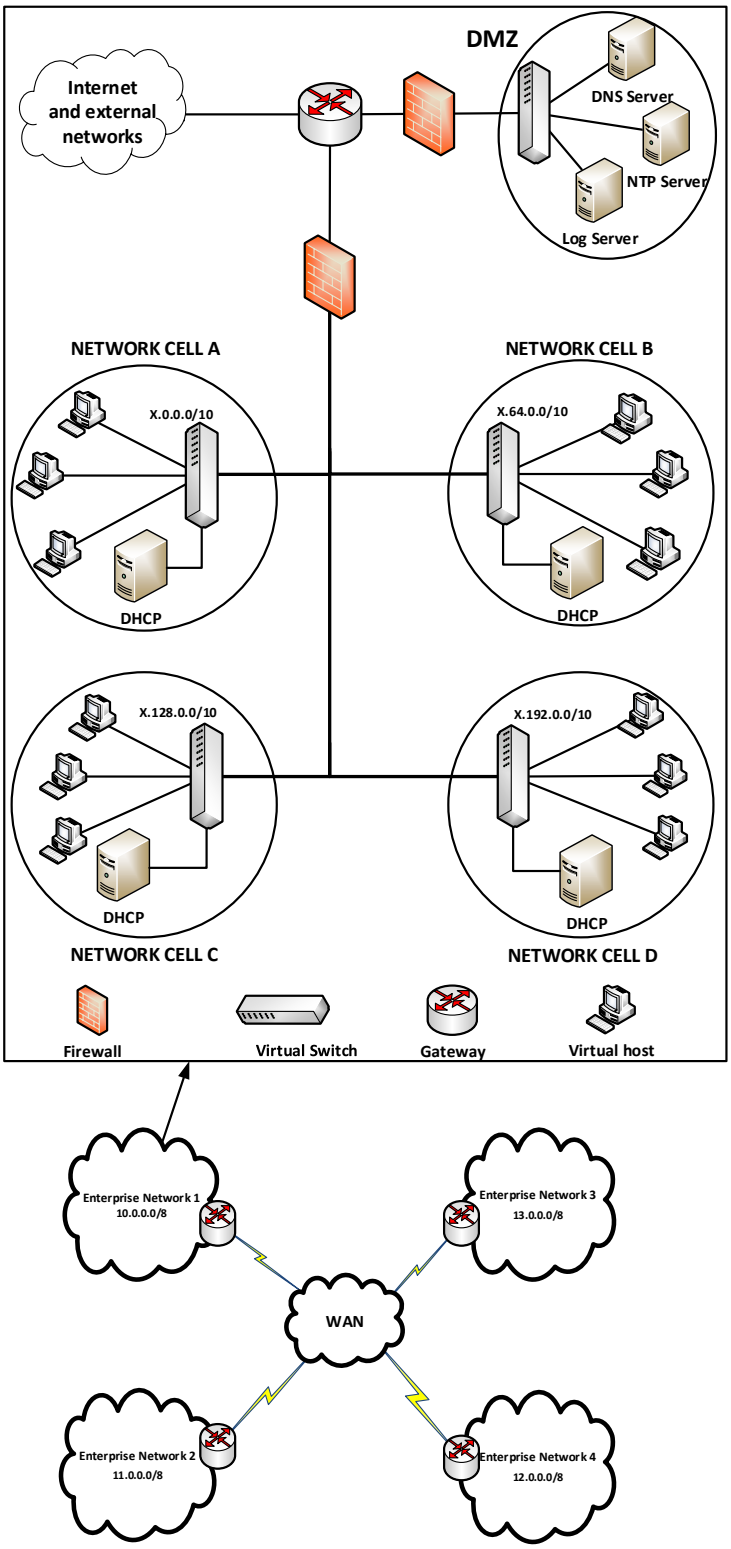


Figure 4.1: A V-Network enterprise network design with four virtualised LANs

4.4 V-Network Implementation

The V-Network testbed has been implemented using the following resources:

- Four servers running VMware ESXi 5.5 server for virtualisation services, each with an Intel Core i7 (12 virtual cores at 3.40 GHz) processor, 64GB of RAM and 2TB of hard disk storage capacity.
- Two servers running the Quagga routing suite to provide routing services, each with an Intel Core i7 (8 virtual cores at 3.40GHz) processor, 24GB of RAM and 1TB of hard disk storage capacity.
- A server running the NTP daemon for time synchronization across all hosts, with an Intel Core i7 (8 virtual cores at 3.40GHz) processor, 24GB of RAM and 1TB of hard disk storage capacity.
- A server running a custom-developed logging server daemon to keep record of host activities, with an Intel Core i7 (8 virtual cores at 3.40GHz) processor, 16GB of RAM and 1TB of hard disk storage capacity.
- A server running VMware vCenter server for managing the ESXi servers remotely, with an Intel Core i7 (8 virtual cores at 3.40GHz) processor, 16GB of RAM and 1TB of hard disk storage capacity.
- Two Ethernet switches.

Figure 4.2 presents the physical design of the V-Network testbed. Each ESXi server accommodates virtual machines in different "portgroups". The portgroups are attached to virtual switches in order to form a number of virtualised LANs within the V-Network testbed. The management network has been configured to enable administrative control over the V-Network ESXi servers. The vCenter Server is used to manage the entire vSphere environment, i.e., all of the ESXi servers in the V-Network testbed and other hardware resources using the "management portgroup" of each ESXi server.

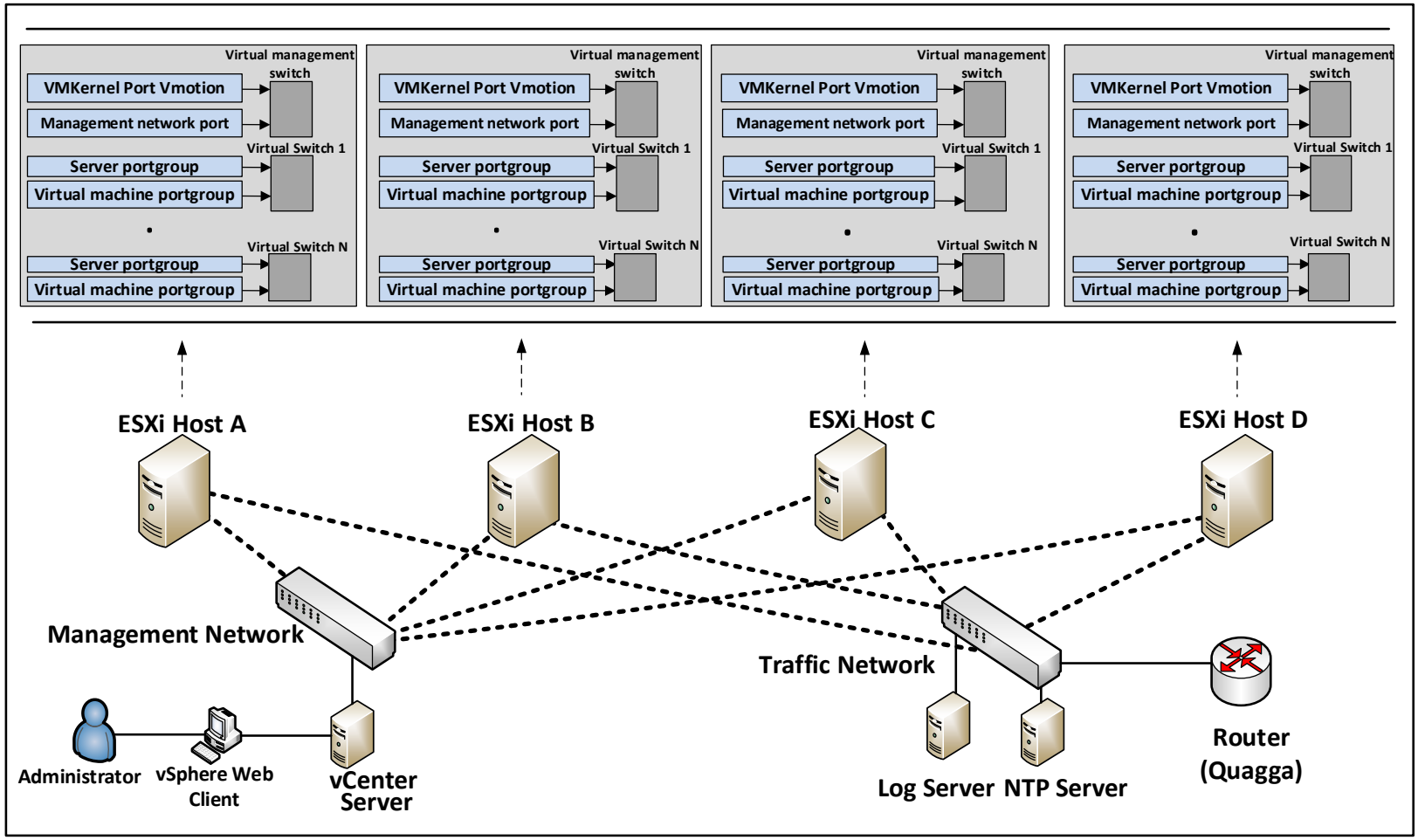


Figure 4.2: V-Network physical implementation

The vCenter Server also supports the use of PowerCLI (Dekens and Renouf, 2011), which enables the management of the vSphere environment using scripts. Thus, using PowerCLI through vCenter Server, scripts can be used to clone a desired number of virtual machines, take virtual machine snapshots, restore virtual machines and servers to a base state after an experiment, clean and re-create virtual machines, start and stop virtual machines and remotely manage the entire V-Network environment. The vSphere Web client is used to remotely monitor, control and manage the infrastructure through the vCenter Server using the GUI. The traffic network is used to provide other network services such as routing and time synchronization. The DMZ of the V-Network testbed is connected to the Internet through a firewall for NTP update prior to experimentation. The Internet connection is then removed before the experiments begin.

Figure 4.3 details the physical and logical design of the V-Network implementation. The virtualised enterprise networks were configured using four class A IP address spaces by default. The IP addressing can be reconfigured to any desired class and subnet depending on the requirements of the malware experimentation used. The virtualised enterprise networks were also connected together using the border router that allows access to other physical or virtualised networks and the Internet. The testbed supports the use of RIP, OSPF and BGP routing protocols.

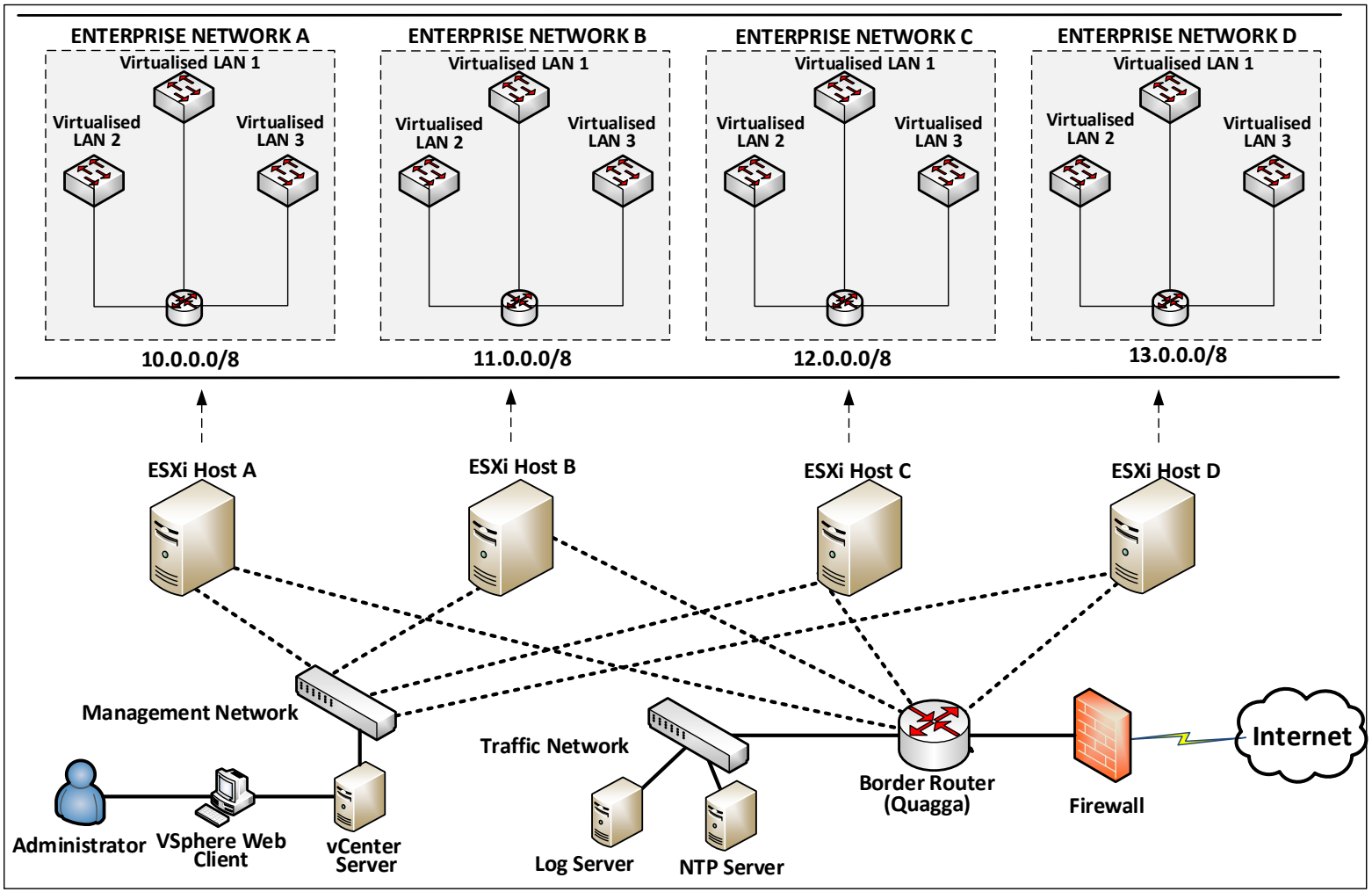


Figure 4.3: V-Network physical and logical implementation

4.5 Worm Daemon

The V-Network testbed uses a worm daemon developed by Shahzad and Woodhead (2014a) with the capabilities of facilitating a worm attack event using chosen worm characteristics. The worm system consists of both client and server modules capable of sending and receiving UDP datagrams. The client module is used to initiate a worm attack against desired targets. The virtual hosts are made vulnerable by running the server module, which listens on a specific UDP port and then, after receiving an "infection" datagram, continuously transmits "infectious" UDP datagrams. Upon infection, a vulnerable host will send its time stamp and IP address information to the logging server for record management. The logging server has been configured with a logging daemon that keeps the details of infected host addresses and infection time. This process will continue until full infection is achieved based on the details recorded on the logging server.

4.6 Configuration Scripts

The V-Network testbed uses utility scripts to manage the virtualised environment and facilitate worm experimentation. The utility scripts can be used for services such as virtual machine start-up and shut-down, cloning virtual machines, resetting the virtual environment to a baseline after experimentation and tearing down virtual machines on the ESXi hosts. To conduct an experiment, a base virtual machine is configured with the correct worm daemon and then cloned to the required number of virtual machines. The V-Network implementation comprises a number of customised utility scripts to facilitate large scale management of virtual machines. The utility scripts are detailed as follows:

Create-VMs: The Create-VMs script is used to create a number of virtual machine clones from a base virtual machine that has been configured with the correct worm daemon and network settings for experimentation. The script then places the developed clones in the same network environment with the base clone.

Start-VMs: The Start-VM script is used to power on a virtual machine or group of virtual machines.

Stop-VMs: The Stop-VMs script is used to shut down a virtual machine or group of virtual machines.

Pause-VMs: The Pause-VMs script is used to pause the activities of a virtual machine or group of virtual machines during an experiment.

Resume-VMs: The Resume-VMs script is used to resume the activities of a virtual machine or group of virtual machines.

Snapshot-VMs: The Snapshot-VMs script is used to snapshot a virtual machine or group of virtual machines. This can be done after starting virtual machines but before conducting an experiment. After the experiment, the virtual machines can be restored to their initial configuration in order to repeat the same set of experiments using Reset-VMs script.

Reset-VMs: The Reset-VMs script is used to reset a virtual machine or group of virtual machine to a standard base.

Move-VMs: The Move-VMs script is used to move a virtual machine or group of virtual machines from one network segment to another.

Teardown-VMs: The Teardown-VMs script is used to clear virtual machine(s) after an experiment to allow for a different configuration and experiment.

The utility scripts require the name of a virtual machine or the group name of virtual machines.

4.7 V-Network Comparative Analysis

The V-Network testbed has the following improved features in comparison with previously reported virtualised testbeds:

- Platform Independence: V-Network uses real-world operating systems, applications, and other networking software, which provides a platform that offers a realistic way of presenting the technology and applications. Unlike vGround (Jiang et al., 2006) that supports Linux-based worm experimentation only, V-Network has been designed to be platform independent, i.e., it supports Windows-based and Linux-based worm experiments.

- Management: V-Network provides utility scripts for managing the infrastructure to facilitate malware experimentation such as re-usability, resetting and tearing down virtual machines. This enables the development of multiple sessions of different worm experiments quickly and easily, contrary to VMT (Shahzad et al., 2013) that used a substantially manual method.

- Background traffic: Unlike VMT (Shahzad et al., 2013), vGround (Jiang et al., 2006) and ViSe (Årnes et al., 2006), V-Network provides support for generating traffic and replaying traffic collected in `.pcap` file format as background traffic. The traffic can be replayed as collected or in a client-server communication fashion.

These improvements can be summarise as presented in Table 4.1.

Table 4.1: V-Network comparative analysis with existing testbeds

Testbed	**Integration with physical network**	**OS Support**		**Background Traffic**		**Management**	
		Unix-based	Windows	Traffic Generation	Traffic Replay	GUI	Shell Script
vGround	No	Yes	No	No	No	No	Yes
ViSe	No	Yes	Yes	No	No	Yes	No
VMT	No	Yes	Yes	No	No	Yes	No
V-Network	Yes	Yes	Yes	Yes	Yes	Yes	Yes

4.8 Summary

This chapter has presented a virtualised environment that has been developed for the purposes of worm propagation experiments and testing countermeasure systems. The virtualised environment termed V-Network has been developed using the VMware hy-

pervisor and has a scale of 1200 virtual machines. V-Network can be managed either using a graphical user interface or command line scripts and it is platform independent, i.e. it supports the use of Windows and UNIX based systems. In contrast to previously reported testing environments, V-Network has higher scale, fidelity and ease of management and provide support for traffic generation and replay. Finally, having developed a countermeasure mechanism and a testing environment, it is desirable to have a programme that will be used to evaluate the countermeasure mechanisms in the developed testing environment. Chapter five presents the evaluation programme used for worm propagation and countermeasure testing, while Chapter six presents the experiments conducted during the evaluation and results obtained.

Chapter 5

AN EVALUATION PROGRAMME

5.1 Introduction

A developed worm countermeasure mechanism, termed NEDAC, and a virtualised testing environment, termed V-Network, were presented in chapters 3 and 4 respectively. Thus, it is desirable to develop an evaluation programme to test the worm detection and containment system in the developed testbed. This can be achieved by determining a suitable evaluation technique that enables running worm traffic against NEDAC using a variety of different settings in a set of repeatable and controlled experiments for performance measurement.

This chapter presents an evaluation programme designed to test NEDAC along with two previously reported worm detection schemes for comparative analysis, namely DSC and a DNS-based technique (henceforth termed DNS-RL). Section 5.2 presents the technique used to test the mechanism. Section 5.3 presents a brief description of the previously reported detection schemes used for comparative analysis. Section 5.4 presents the performance metrics and other derived attributes determined during the evaluation programme. Section 5.5 presents the background traffic used during the evaluation programme. Section 5.6 presents the worm daemons used to generate worm traffic. Section 5.7 presents the procedure followed during the evaluation programme. Finally, Section 5.8 presents the chapter summary.

5.2 Evaluation Technique

The evaluation of network-based security systems, such as network intrusion detection systems (NIDSs), requires the use of a dataset that closely reflects the characteristics of real network traffic. The approaches used by security researchers to evaluate network intrusion detection systems can be classified as testing without background traffic, testing using real-time traffic, testing using collected real traffic, testing using sanitized traffic, and testing using simulated traffic (Mell et al., 2003). Mell et al. (2003) noted that testing an NIDS without background traffic or using real-time traffic have performance and management limitations respectively. The former is limited in determining false alarms due to the absence of background traffic, which cannot determine the performance of an NIDS in the presence of background traffic. The latter has difficulties (if not impossible) in repeating an experiment and in determining false positives because it requires manual inspection of traffic logs or statistical sampling techniques to find attacks.

To evaluate an NIDS using a collection of real traffic trace, attack datagrams are injected in to the trace, which can later be replayed to test an NIDS. The NIDS can also be tested by replaying the traffic trace while running attacks concurrently. This approach enables repeating experiments and therefore facilitates the determination of false positives. The performance of an NIDS can also be determined at different levels of background activity. However, to evaluate an NIDS using simulated traffic, a testbed network is created with hosts and network infrastructure that can be successfully attacked and to generate background traffic on the network. The testbed is configured to generate background traffic using traffic generation tools, which can also be recorded for a playback. This approach enables repeatability because previously generated background activity can be replayed. The approach is also promising because it facilitates the determination of detection performance and false positive rates.

Thus, NEDAC along with the DSC and DNS-RL schemes were evaluated using pseudo-worm attack datagrams along with a set of collected real traffic traces in a controlled environment. The traffic traces were used as background traffic to assess the possibility of

false positives while worm attack datagrams were used to assess the detection capabilities of the schemes. The NEDAC scheme has two sub-systems that employ different detection techniques. The first system, which is server-based, used datagram header information to detect anomalies from server hosts in a network, and therefore does not require payload information. The other system, which is workstation-based, used DNS activities to determine anomalies from client hosts in a network, and therefore requires data from the payload of DNS reply datagrams.

5.3 Selected Detection Schemes for Comparison

This section presents a brief summary of the candidate detection schemes chosen for comparative evaluation against NEDAC. The detection schemes that used correlation of inbound and outbound traffic and destination contacts as a behavioural signature provided a better performance in identifying scanning worms with less complexity and false positives in comparison to schemes that used statistical analysis (Li et al., 2008). The schemes are also capable of detecting both TCP-based and UDP-based worms unlike the schemes that rely on connection status to detect worm propagation and infection.

5.3.1 Destination Source Correlation

Gu et al. (2004) proposed a correlation scheme termed Destination Source Correlation (DSC). The scheme detects the presence of a worm by correlating inbound and outbound traffic on a given port, i.e, if a host received a datagram on port i and then starts sending datagrams destined for port i, it becomes a suspect. If the behaviour exceeds a threshold, then an alarm is raised. The DSC scheme monitors outbound datagrams from a host for a duration of time. An algorithm was developed based on the description of Gu et al. (2004) that uses a network based solution to contain worm infection, i.e., block outbound datagrams from an infected host at network level. However, the existing acronym of the scheme was maintained.

5.3.2 DNS Rate Limiting

Whyte et al. (2005) and Shahzad and Woodhead (2014b) proposed a DNS-based technique to detect worm propagation in an enterprise network. The DNS-based schemes used the absence of DNS resolution by a host prior to contacting a new destination IP address. The schemes monitor the behaviour for a duration of time and a counter is maintained for each host that exhibits this behaviour. Upon a host exceeding a threshold, an alarm is raised and the host is blocked from sending outgoing traffic at the network level. An algorithm was developed based on the descriptions provided by Whyte et al. (2005) and Shahzad and Woodhead (2014b) and the acronym assigned to the algorithm is DNS-RL.

5.4 Performance Metrics

A range of metrics (Fink et al., 2002) are commonly used by security researchers to quantify the performance of a network intrusion detection systems. The key metrics required for this evaluation are summarised as follows:

True positive (TP) rate: This is the percentage of worm alarms raised when a worm is present.

True negative (TN) rate: This is the percentage of identified benign traffic when there is no worm.

False positive (FP) rate: This is the percentage of worm alarms raised when there is no worm.

False negative (FN) rate: This is the percentage of worm datagrams not identified by a detection system.

Accuracy: The accuracy of a detection system is used to measure the ability of the system to raise an alarm only when a worm is present. This metric is

derived as presented in equation 5.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Precision: This is the probability of identifying worm scanning when a detection system raises an alarm, i.e. fraction of the worm instances that are positively identified as worms. This metric assesses the usability of a detection system because alarms are useful only if there is a positive identification of worm traffic. This metric is derived as presented in equation 5.2.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Receiver Operating Characteristic (ROC): An ROC curve is used to illustrate the interaction between the TP rate and the FP rate. This is useful when analysing and comparing multiple detection systems.

5.5 Background Traffic

A set of legitimate traffic traces are required for the evaluation. The NEDAC scheme and most worm detection systems are designed to identify the presence of worm infection within subnets, therefore, it is desirable to use datasets that comprise traffic that occur in subnets. The available datasets that closely matched the evaluation requirements are the DARPA (Lippmann et al., 2000), CDX 2014 (USMA, 2016) and the LAB (see section 5.5.3) datasets. The datasets were chosen because they comprise traffic that occurred in subnets with a range of commonly used network protocols such as ARP, IP, TCP, UDP, DNS, TFTP, TLS/SSL, FTP, HTTP, HTTPs, IMAP4, POP3, SMTP, SNMP, SSH, DHCP, NTP and Telnet. Additionally, the DARPA and CDX traffic are labelled datasets that support security analysis. The three traces include a wide range of collected traffic from a number of network hosts with statistics as presented in Table 5.1.

Table 5.1: DARPA, CDX and LAB Trace Statistics

Trace	Datagrams				Active Hosts
	TCP	UDP	ARP	Total	
DARPA	14,517,826 (87%)	2,297,690 (13%)	11,540 (0.37%)	16,725,516	31
CDX	1,212,877 (94%)	62,768 (5%)	6,862 (1%)	1,282,507	20
LAB	3,276,725 (79%)	856,234 (20%)	9,763 (0.24%)	4,142,722	60

To prepare the traces for a replay session, each dataset was divided into datagrams originating from a client host and those originating from a server host using a **`tcpreplay`** pre-processor known as **`tcpprep`** (Turner and Bing, 2005). The function of **`tcpprep`** is to split traffic in a `.pcap` file into two streams (traffic originating from client and server). The details of the two streams are then recorded in a **`.cache`** file. The resulting **`.cache`** file can then be used by **`tcpreplay`** to replay the traffic traces as client datagrams or server datagrams. The resulting client and server traffic was then replayed as client-server communication using a **`tcpreplay`** command between endpoints.

5.5.1 DARPA Dataset

The DARPA 1999 dataset comprises traces generated over five weeks of simulated network traffic. The physical network consists of an inside and outside component separated by a router. The network comprises five victim hosts, which are the targets of attacks and one sniffer. The inside component includes victim hosts of many types (e.g. Linux, Solaris, Sun OS) while the outside component includes two workstations which simulate gateways to a virtual outside Internet leading to 100 hosts. Data is collected from the inside victim running Solaris and from an outside sniffer. The evaluation programme reported in this thesis uses the traffic data collected by inside sniffers on week 1 and 3. Although the DARPA 1999 dataset is 17 years old at the time of this writing, the "inside" traces of week 1 and 3 of the dataset are attack free traces that contain payload information for a variety of protocols needed for the evaluation. Additionally, the traces include a wide range of collected traffic from 31 network hosts.

5.5.2 Cyber Defense Exercise (CDX) 2014 Dataset

The CDX is a computer security competition designed to promote awareness about the role of Information Assurance in protecting critical information systems. The CDX 2014 dataset comprises traffic from a cyber defense exercise that incorporated a set of hosts used for network attack and another set of hosts for benign traffic generation. The dataset recorded attack traffic generated using 30 hosts and white traffic generated using 20 hosts by manually interacting with web, email, DNS lookups, and other required services. The evaluation programme reported in this thesis uses the benign traffic generated during the CDX 2014 competition.

5.5.3 LAB Dataset

The LAB dataset was collected from a departmental computer laboratory that serves students and administrative staff members. The dataset comprises traffic with a variety of web (administrative sites and web portals for students and faculty), email services and other applications such as file sharing (SMB). The dataset comprises traffic collected from 60 hosts over a span of seven hours for five days using a network tap.

5.6 Pseudo-worms

The evaluation process used the Slammer worm and two contemporary pseudo-worms that were developed based on two recent wormable vulnerabilities, as test examples. The Slammer worm was chosen because it is among the fastest worm outbreaks experienced on the Internet that caused financial losses and disruption of services. The developed contemporary pseudo-worms were used to further evaluate the NEDAC scheme using new potential fast scanning worms and to attempt to characterise the threat posed by the recent vulnerabilities. The recent vulnerabilities were Microsoft RDP (CVE-2012-0002) and ShellShock (CVE-2014-6271).

To develop and use the pseudo-worms in experiments, some important metrics are required such as the vulnerable populations for the vulnerability, worm datagram size and

the likely scan rate. Moore et al. (2003a) reported that the Slammer worm had at least a vulnerable population of 75,000 hosts. They also noted that Slammer exhibited an average scan rate of 4,000 datagrams per infected host per second and had a datagram size of 404 bytes. Ahmad and Woodhead (2015) reported the vulnerable population and potential datagram size of the Microsoft RDP and ShellShock vulnerabilities as having values of 16.5M and 42.5k and 3,800 bytes and 2,000 bytes respectively.

5.6.1 Scan Rate

The bandwidth available for an infected host and the worm datagram size determine how fast a worm can send datagrams particularly in the case of UDP-based worms. The upload speed of Internet hosts was reported to be within the range of circa 3.2 Mbps to 5 Mbps in 2015 and 2016 (van der Vorst et al., 2014). van der Vorst et al. (2014) also reported an upload speed of 7.3 Mbps to 10 Mbps for network environments with high speed Internet connection, which can also be higher for data centre hosts (Zhuang et al., 2013). Additionally, the vast majority of server hosts have higher Internet bandwidth than client hosts in a network. Thus the bandwidth available to hosts to transmit worm datagrams depends on the network environment and the category of the hosts.

Based on the assumption that the Internet connected client and server hosts exhibit an average data transmission rate of 5 Mbps and 7.5 Mbps respectively, the scan rate S, available to a single worm instance, on an infected host, to transmit a datagram of size M (in bytes), over a C megabits Internet connection per second can be determined using equation 5.3.

$$S = \frac{C}{M} \tag{5.3}$$

Therefore, the possible scan rates for the RDP and ShellShock pseudo-worms are 165 and 313 datagrams per second for client hosts and 247 and 468 datagrams per second for server hosts respectively. The scan rates of the pseudo-worms were scaled down by a factor of 32, 1.3 and 2.5 for the Slammer, RDP and ShellShock pseudo-worms respectively for client hosts to avoid overloading V-Network server resources. The resulting

scan rates employed in the experiments are 125, 127 and 125 "infectious" datagrams per second for Slammer, RDP and ShellShock respectively.

5.6.2 Parameters

Having determined the vulnerable population values of the candidate worms along with the size of routable IPv4 address space ($3,673,309,759$ (Cotton and Vegoda, 2010), the number of vulnerable hosts per million Internet hosts for each pseudo-worm was determined using equation 5.4.

$$P_m = \frac{S_p}{R_{ip}} * 1,000,000 \tag{5.4}$$

where, P_m denotes the value of vulnerable hosts per million Internet hosts, S_p denotes the absolute number of vulnerable hosts and R_{ip} denotes the number of routable IPv4 addresses. The results were 21, 4454 and 12 vulnerable hosts per million Internet hosts for the Slammer, RDP and ShellShock pseudo-worms respectively. For experimentation, the maximum number of virtual hosts supported in the V-Network testbed is 1200. Therefore the number of hosts required for experimentation with each candidate worm depends on the value of its vulnerable hosts per million Internet hosts. Using IPv4 class A size network (2^{24}), the Slammer and ShellShock pseudo-worm experiments require $\left[2^{24} * 3 * \frac{21}{1000000}\right] = 1057$ and $\left[2^{24} * 5 * \frac{12}{1000000}\right] = 1007$ vulnerable hosts respectively. The RDP pseudo-worm has a relatively higher value of vulnerable hosts per million Internet hosts and therefore using class A size network requires a larger number of vulnerable hosts. Therefore, using four class B size networks (2^{16}), the RDP pseudo-worm experiment requires $\left[2^{16} * 4 * \frac{4454}{1000000}\right] = 1168$ vulnerable hosts. Thus, the experimentation used 1057 and 1007 and 1168 vulnerable hosts for Slammer, ShellShock and RDP pseudo-worm respectively, within the relevant network address spaces.

5.6.3 Worm Traffic

Initiating a worm outbreak involves creating the required number of virtual machines in V-Network from a base virtual machine that has been configured with the correct pseudo-worm daemon. For the Slammer pseudo-worm experiments, 1056 virtual machines were created across four enterprise networks from a configured base virtual machine, i.e., 264 virtual machines in each enterprise network. The same procedure was applied to RDP and ShellShock pseudo-worms, but the total number of virtual machines for the RDP pseudo-worm was 1168 across four enterprise networks (292 virtual machines per enterprise network) and 1007 virtual machines across five enterprise networks (201 virtual machines in three enterprise network and 202 in two enterprise networks) for the ShellShock pseudo-worm. For each experiment, the virtual machines were powered to automatically synchronise their time with the NTP server, and then wait for inbound datagrams. The worm infection event was then initiated by sending a UDP datagram to one of the vulnerable virtual machines in one of the virtualised enterprise networks. A UDP-based worm has been chosen due to its higher rate of propagation compared to a TCP-based counterpart. UDP-based worms require no acknowledgement and cannot be detected by mechanisms that rely on the number or state of failed connection attempts.

5.7 Procedure

The evaluation programme was conducted using the developed software prototypes of NEDAC, DSC and DNS-RL in sets of test experiments. During the evaluation, a prototype of a detection scheme was positioned on the gateways of each virtualised LAN in the V-Network enterprise networks, and for NEDAC, the containment system was positioned on the switches as depicted in Figure 5.1.

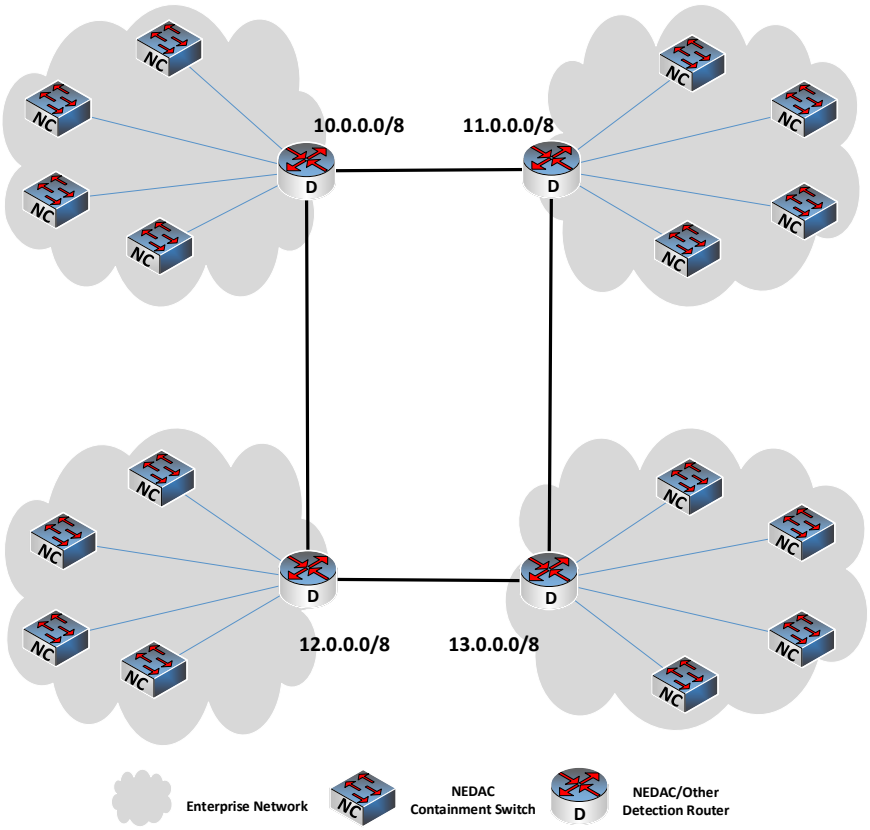


Figure 5.1: Prototype evaluation set up

5.7.1 Testing for Random Destination Contacts

The sets of experiments conducted for each detection scheme comprise a test for Slammer, RDP and ShellShock pseudo-worms using random and then hit-list scanning behaviours. The random scanning technique probes IPv4 addresses within the routable IP address space. The hit-list scanning technique infects a list of pre-compiled vulnerable hosts, which includes both client hosts and a server host, and then each infected host uses random scanning. Random scanning was chosen because it is the commonly used propagation method for most of the worm outbreaks experienced on the Internet such as the Code Red (Zou et al., 2002), Slammer (Moore et al., 2003a), Witty (Shannon and Moore, 2004) and Conficker (SANS, 2008). Additionally, hit-list scanning was also experienced during the outbreak of the Witty worm. Shannon and Moore (2004) noted that the Witty worm used a hit-list size of 110 to 160 hosts within the first 30 seconds

of the worm outbreak. This hit-list size was at least of 1% of Witty vulnerable hosts. Therefore, the evaluation used 1% of the vulnerable population values of 1057, 1168 and 1007 for Slammer, RDP and ShellShock pseudo-worms. The hit-list sizes were 11, 12 and 10 for Slammer, RDP, ShellShock pseudo-worms respectively.

5.7.2 Testing for Unused local IP Address Contacts

To evaluate the ARP-based detection component, a range of active IP addresses were defined in each subnet. Therefore during the evaluation, an ARP datagram transmitted outside the range of the active addresses of a subnet is considered suspicious and therefore flagged as a scanning event. The threshold used for ARP based worm scanning are, 1, 2 and 3 ARP request datagrams sent to inactive local IP addresses.

5.7.3 Threshold and Background Traffic Set-up

The experiments were conducted using threshold values of 100, 200, 300 and 400 worm datagrams sent by a host within timing windows of 10, 15 and 20 seconds. The threshold values were chosen because a fast scanning network worm can be detected before causing damage to networks even after transmitting 100 or 200 datagrams. Meanwhile, the values do not exert strong restriction for benign traffic that exhibit worm-like behaviour. The range of values was used to determine the possible effect of the thresholds and timing windows on the percentage of false positives. Each pseudo-worm experiment was conducted in the presence of background traffic using each of the three traffic datasets; DARPA, CDX and LAB. This was to test the detection systems using legitimate traffic with different characteristics and model heterogeneity in the experiments because the datasets comprises traffic from the most commonly used operating systems; Unix-based and Windows.

To test the effect of dynamic threshold scheme, the experiments were initiated in a defined peak period and then continue through a defined quiescent period. The peak period is characterised by a large volume of generated background traffic, i.e, the background traffic was replayed in large volume while worm attack datagrams are transmitted

by the pseudo-worms. Conversely, the quiescent period is characterised by low volume of generated background traffic along with worm attack datagrams. These were achieved by setting a number of hosts to replay background traffic in large volume while another set of hosts replay traffic in low volume during peak period. As the peak period elapses, the hosts that replay large volume of background traffic were halted and the set of hosts that replay low volume of background traffic continue. Citicaka (2014) noted that traffic volume reduces by 60% during non-business hours, therefore the evaluation examined the effect of reducing the threshold to 25%, 30% and 35%. The volume of background traffic replayed by each host during the peak period was the average number of datagrams transmitted per second for each traffic dataset, which was 76, 153 and 184 datagrams per second for the DARPA, CDX and LAB traffic traces respectively. However, the traffic volume was reduced by 25%, 30% and 35% during the quiescent period.

5.7.4 Experimentation Set-up

For each pseudo-worm experiment, a number of hosts (1057 for Slammer, 1168 for RDP and 1007 for ShellShock) were configured with the correct daemon for worm attack datagrams while other hosts were configured to replay the background traffic between endpoints across the V-Network internetwork as depicted in Figure 5.2. The worm attack and traffic replay events were executed concurrently in each experiment. The experiments were conducted without any countermeasures in place, then repeated with the countermeasures in the presence of background traffic. The performance metrics, TP, TN, FP and FN were determined for each countermeasure experiment. The metrics were then used to determine the precision and accuracy of each detection scheme.

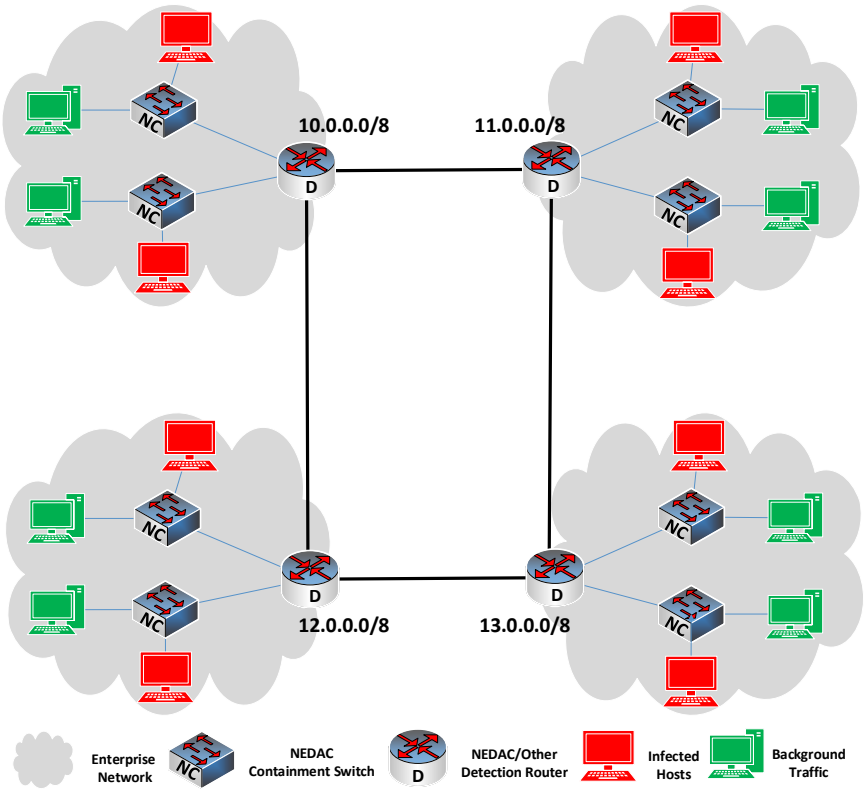


Figure 5.2: Pseudo-worm attack and background traffic generation events

Figure 5.3 summarises the evaluation experiments conducted for each of the three detection schemes. Each of the 18 experiments presented in Figure 5.3 was conducted 12 times using a combination of different threshold values and timing windows as presented in Table 5.2, and then repeated twice.

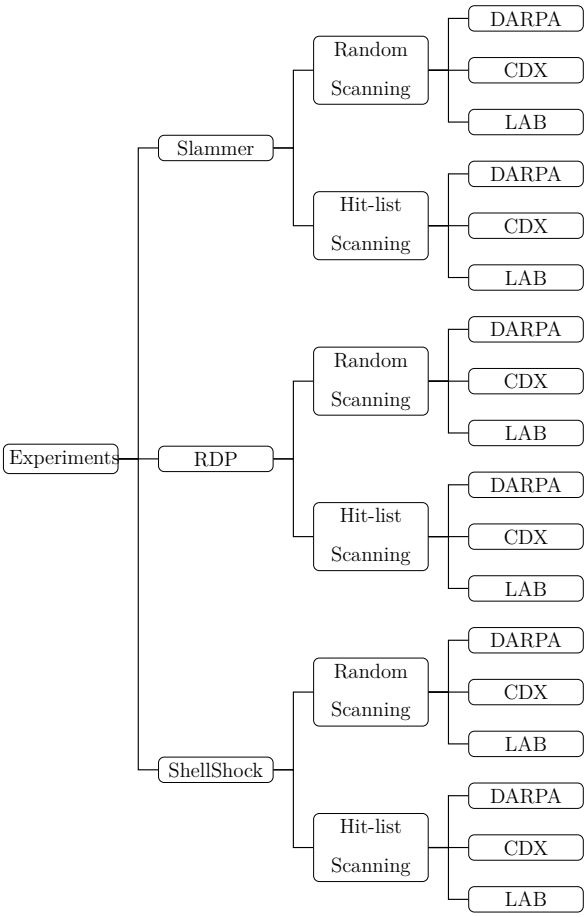


Figure 5.3: Evaluation experiments

Table 5.2: Sets of Experiments

Experiments	Threshold value				Timing Window		
	100	200	300	400	10	15	20
1	✓	✗	✗	✗	✓	✗	✗
2	✗	✓	✗	✗	✓	✗	✗
3	✗	✗	✓	✗	✓	✗	✗
4	✗	✗	✗	✓	✓	✗	✗
5	✓	✗	✗	✗	✗	✓	✗
6	✗	✓	✗	✗	✗	✓	✗
7	✗	✗	✓	✗	✗	✓	✗
8	✗	✗	✗	✓	✗	✓	✗
9	✓	✗	✗	✗	✗	✗	✓
10	✗	✓	✗	✗	✗	✗	✓
11	✗	✗	✓	✗	✗	✗	✓
12	✗	✗	✗	✓	✗	✗	✓

5.8 Summary

This chapter has presented an evaluation programme that has been developed to test NEDAC along with two earlier detection methods for comparative purposes. The programme uses collected traces together with worm attack datagrams to evaluate the detection systems. The collected traces, from the DARPA, CDX and LAB datasets, were used as background traffic during the evaluation. The worm attack events employed a Slammer-based pseudo-worm and two contemporary pseudo-worms to facilitate worm propagation experiments. The worm propagation experiments used random and hit-list worm scanning strategies for the testing. Finally, the number and range of experiments needed to evaluate the detection schemes have been set out.

Chapter 6

RESULTS OF THE NEDAC SCHEME EVALUATION

6.1 Introduction

This chapter presents the results of the evaluation experiments used to test the performance of NEDAC in comparison with the DSC and DNS-RL schemes. The objective of this chapter is to empirically establish whether NEDAC meets the research objective of developing a more faster, effective and accurate worm detection and containment mechanism in comparison with the existing schemes. Section 6.2 presents the worm experimentation set up used and Sections 6.3 and 6.4 present the worm detection and containment results respectively. The false positive results for the DSC, DNS-RL and NEDAC schemes are presented in Section 6.5. Section 6.6 presents analysis of the performance of the detection and containment schemes and Section 6.7 summarises the chapter.

6.2 Worm Propagation Experiments

To evaluate the capability of NEDAC together with the DSC and DNS-RL schemes, a set of worm outbreak experiments were conducted in the presence of each scheme as set out in Table 5.2. Each experiment was repeated twice and an average result of three experiments was computed. The three worm daemons used during the evaluation were configured as discussed in Sections 6.2.1 through 6.2.3.

6.2.1 Slammer

As presented in Section 5.6.2, the Slammer pseudo-worm experiments were conducted using 1057 vulnerable hosts (1051 as client hosts and 6 as server hosts) across three class A size networks as depicted in Figure 6.1.

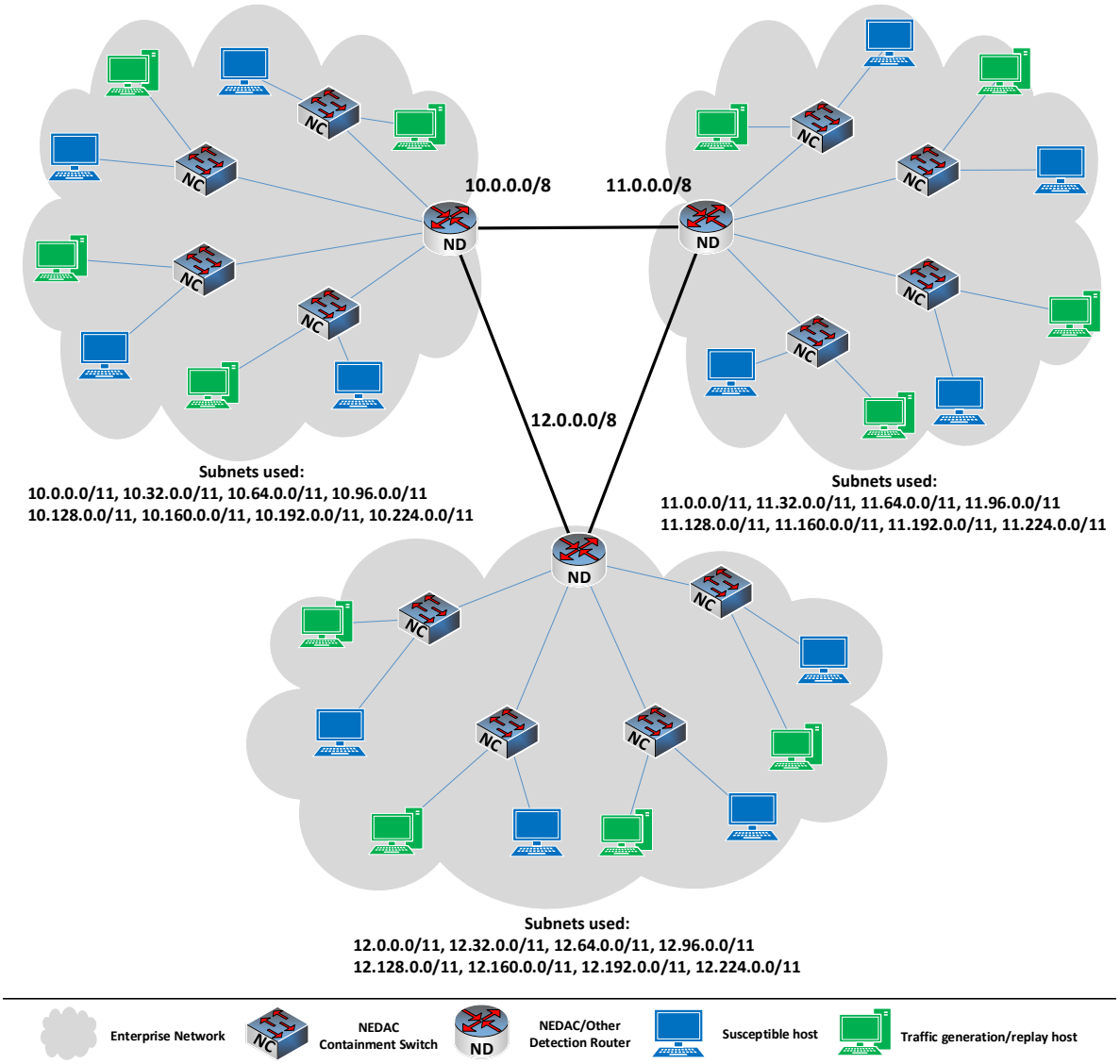


Figure 6.1: Slammer pseudo-worm experimental set-up

The Slammer pseudo-worm daemon was configured to listen on UDP port 1434 and then transmit UDP datagrams to port 1434 at a scan rate of 125 “infectious” datagrams per second, once “infected”.

6.2.2 RDP

Also as presented in Section 5.6.2, the RDP pseudo-worm experiments were conducted using 1168 vulnerable hosts (1160 as client hosts and 8 as server hosts) across four class B size networks as depicted in Figure 6.2.

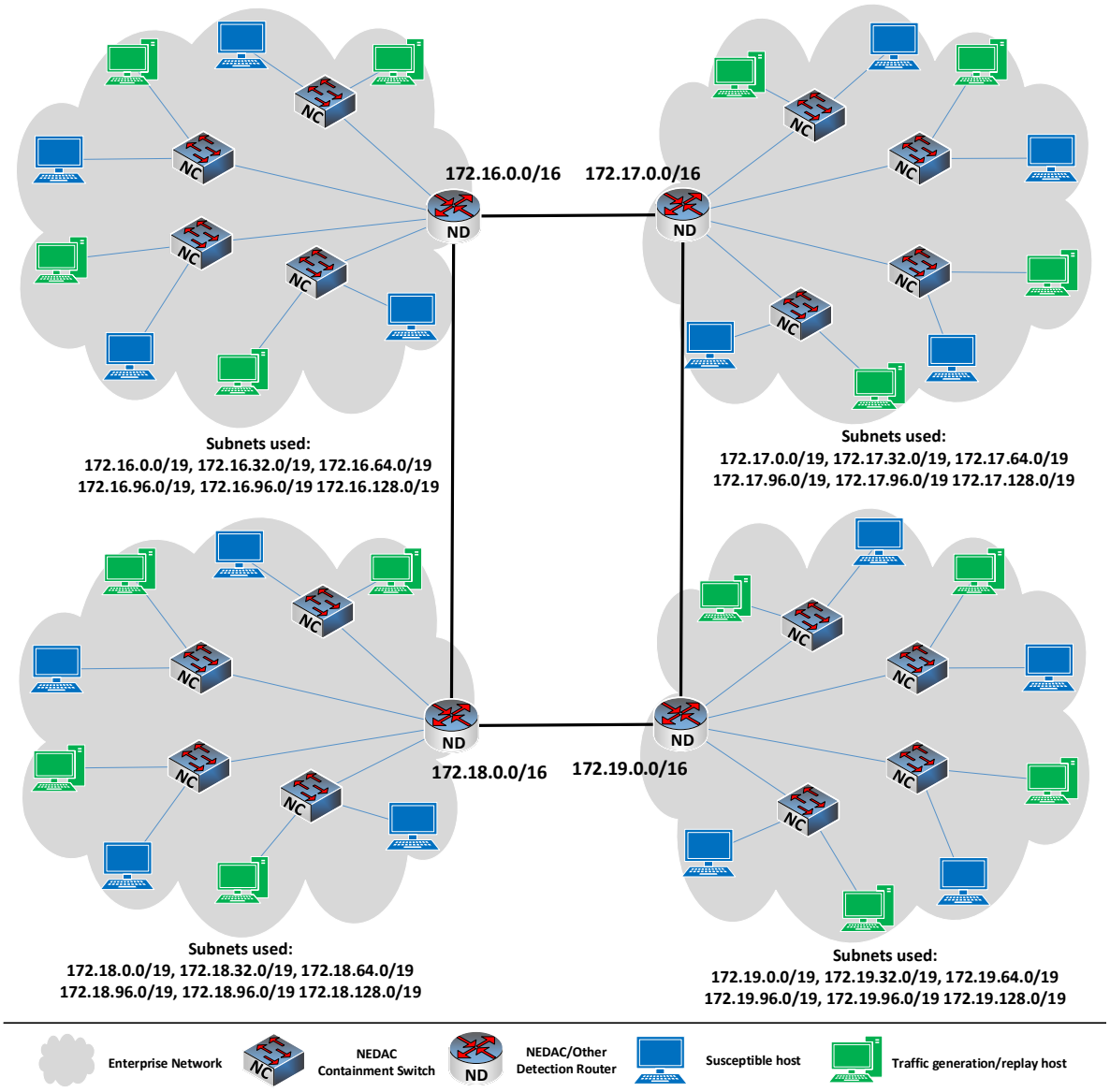


Figure 6.2: RDP pseudo-worm experimental set-up

The RDP pseudo-worm daemon was configured to listen on UDP port 3389 and then transmit UDP datagrams to port 3389 at a scan rate of 127 “infectious” datagrams per second, once “infected”.

6.2.3 ShellShock

Finally, the ShellShock pseudo-worm experiments were conducted using 1006 vulnerable hosts, as derived in Section 5.6.2 (996 as client hosts and 10 as server hosts) across five class A size networks as depicted in Figure 6.3.

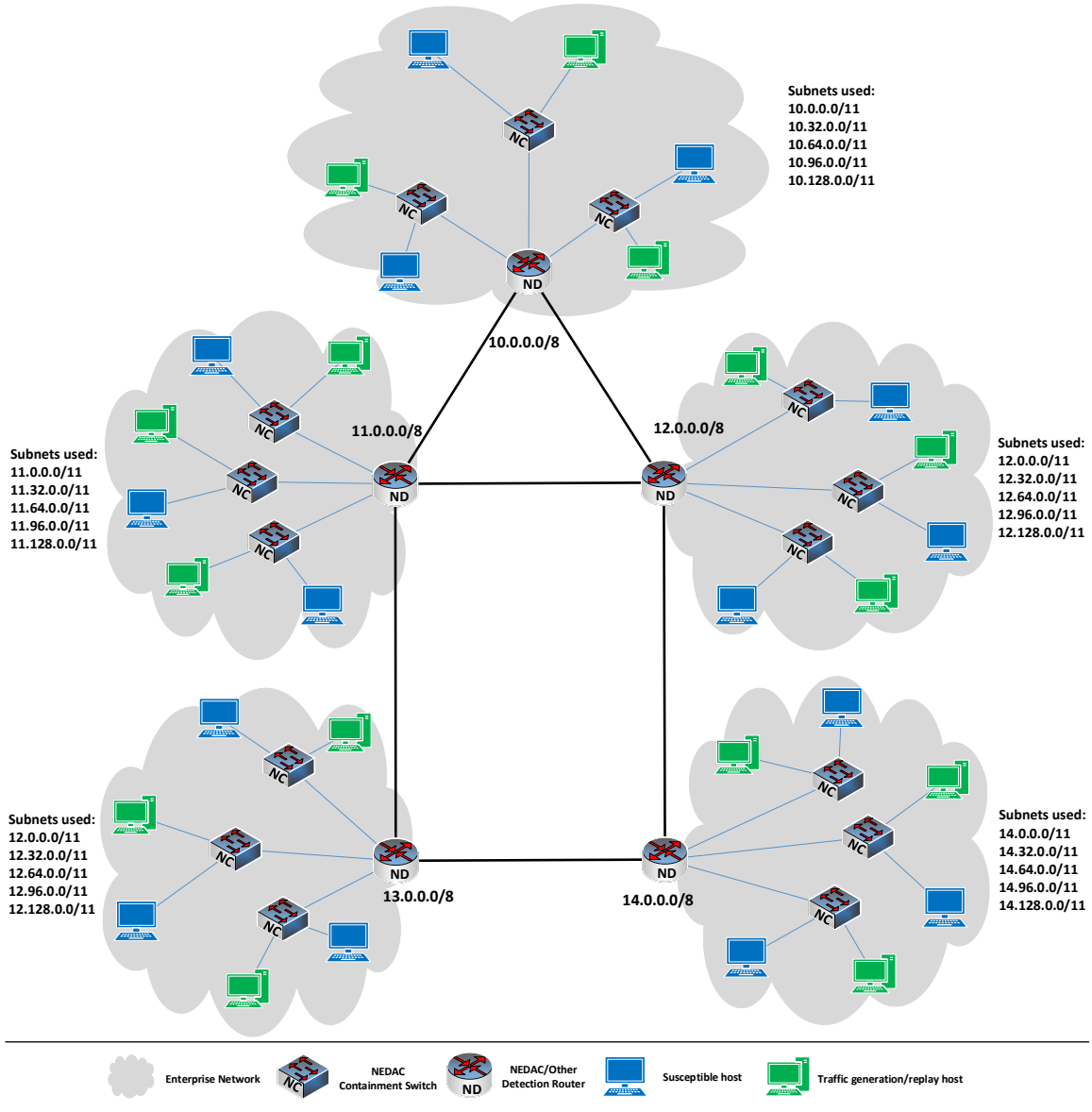


Figure 6.3: ShellShock pseudo-worm experimental set-up

The ShellShock pseudo-worm daemon was configured to listen on UDP port 8080 and then transmit UDP datagrams to port 8080 at a scan rate of 125 “infectious” datagrams

per second, once "infected".

6.3 Worm Detection Results

The Slammer, RDP and ShellShock worm experiments were conducted using random and hit-list scanning techniques in the presence of background traffic. The thresholds used during the experiments were 100, 200, 300 and 400 anomalous datagrams sent within the timing windows of 10, 15 and 20 seconds as discussed in Section 5.7.3. The detection performance of each scheme was determined during the experiments and then an average result of three experiments was computed for each scheme. The results are then presented based on the type of background traffic used, i.e. DARPA, CDX and LAB network traffic.

The metrics used to assess the detection performance of each scheme were *Accuracy* and *Precision* as discussed in Section 5.4. The main goal of a detection system is to accurately identify a worm infection. Precision shows how a detection scheme correctly classifies worm instances as intrusions, while accuracy measures how satisfactorily the worm instances are classified as intrusions by a detection scheme (En-Najjary and Urvoy-Keller, 2010). The overall detection performance of a scheme is then assessed based on the extent of its accuracy and precision.

6.3.1 Precision

This section presents the results of the precision of worm detection for the DSC, DNS-RL and NEDAC schemes. Figure 6.4 shows the precisions of the three detection schemes with the DARPA network traffic. The precisions of the DSC, DNS-RL and NEDAC, with a timing window of 10 seconds and a threshold value of 100, are 88.4%, 86.7% and 97.3% respectively. Thus the schemes are 88.4%, 86.7% and 97.3% likely to correctly classify worm instances as intrusions with a threshold value of 100 and a timing window of 10 seconds respectively. The results also show that the precisions increased with increasing threshold values (200, 300 and 400) and a timing window of 10 seconds; 3.5%, 3.6%

and 1.4%, on average, for the DSC, DNS-RL and NEDAC schemes respectively. The precisions also increased, on average, by 2.6%, 3.4% and 1.3% with increasing threshold values and a timing window of 15 seconds for the DSC, DNS-RL and NEDAC schemes respectively. It should be noted that the precisions of the three detection schemes also increased with the same threshold value but longer timing window. For instance, the precision of DSC with threshold values of 100 and 200 and a timing window of 10 seconds increased by 3.8% and 4.1% respectively when the timing window was increased to 15 seconds.

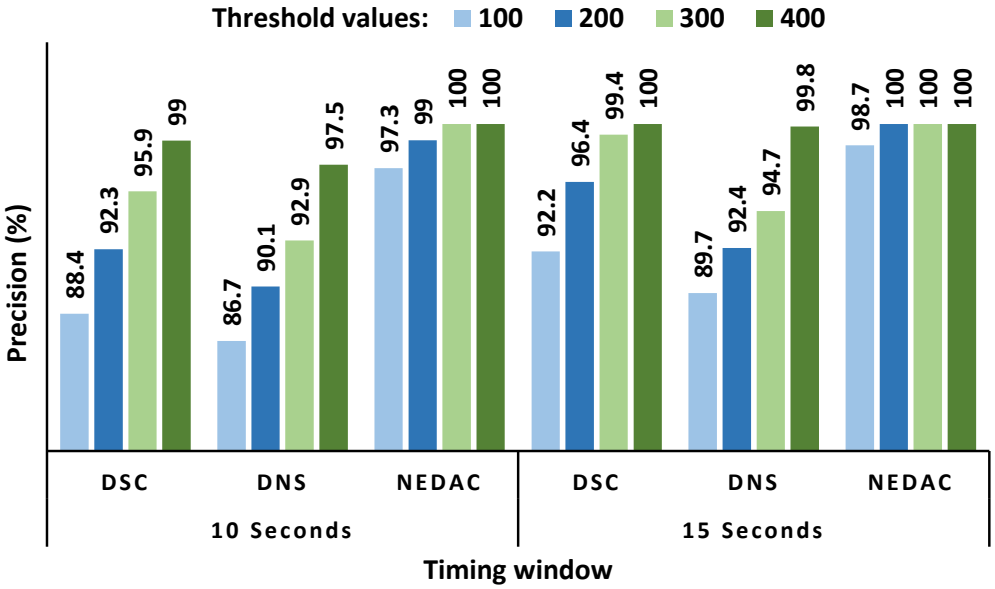


Figure 6.4: Precision of detection with the DARPA network traffic

Figure 6.5 shows the precision values of the three detection schemes with the CDX network traffic. The precision increased with increasing threshold values and longer timing window as with the DARPA network traffic. The precision increased, on average, across increasing thresholds and a timing window of 10 seconds by 3.3%, 2.8% and 1% for the DSC, DNS-RL and NEDAC schemes respectively. The precision also increased by 3.1%, 2.5% and 1.6% for DSC, DNS-RL and NEDAC with a threshold value of 100 and a timing window of 15 seconds.

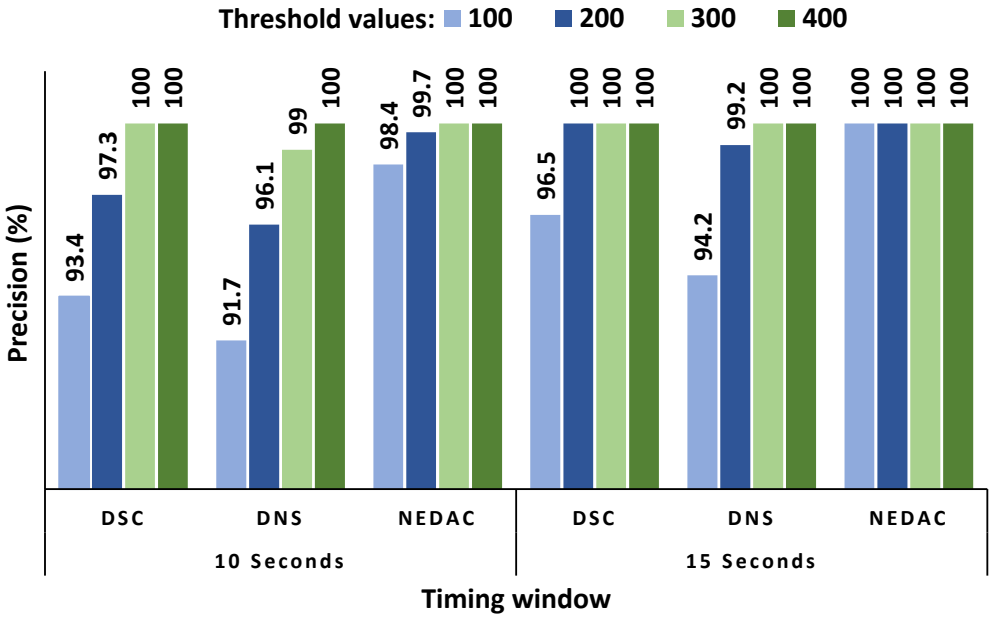


Figure 6.5: Precision of detection with the CDX network traffic

Figure 6.6 shows the precision values of the three detection schemes with the LAB network traffic. With a timing window of 10 seconds, the precision increased, on average, by 1.2% and 1.7% for the DSC and DNS-RL schemes respectively across the increasing thresholds.

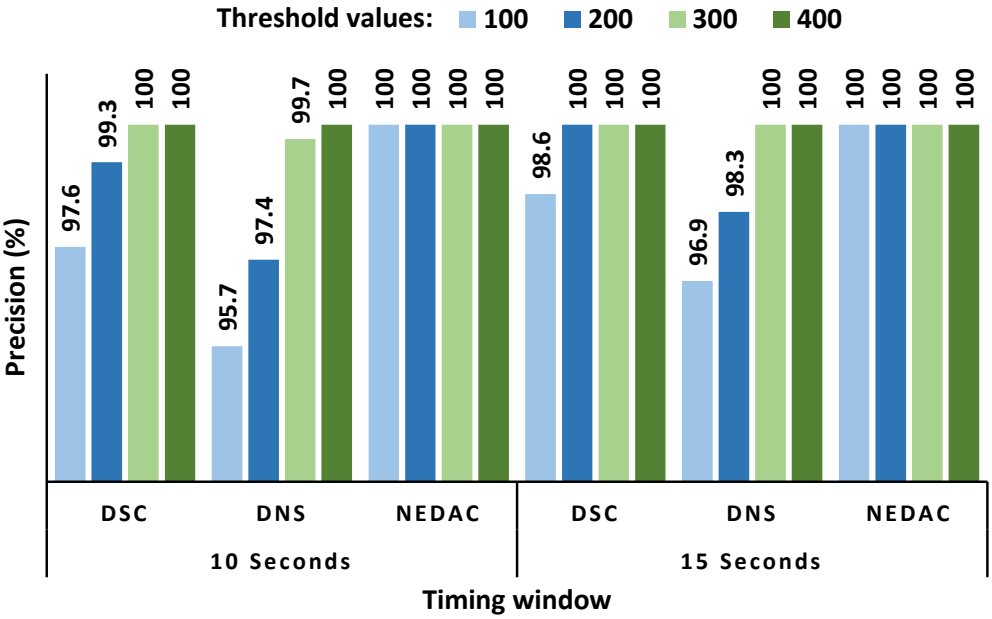


Figure 6.6: Precision of detection with the LAB network traffic

The precision also increased by 1% and 1.2% for DSC and DNS-RL with a threshold

value of 100 and a timing window of 15 seconds. The precision of NEDAC remained 100% across the thresholds and timing windows used with the LAB network traffic.

6.3.2 Accuracy

This section presents the results of the accuracy of worm detection for each of the DSC, DNS-RL and NEDAC schemes. Figure 6.7 shows the accuracy of worm detection for the three schemes with the DARPA network traffic. The values of detection accuracy for DSC, DNS-RL and NEDAC with a threshold value of 100 and a timing window of 10 seconds are 93.4%, 92.3% and 98.4% respectively. Thus the DSC, DNS-RL and NEDAC schemes are 93.4%, 92.3% and 98.4% likely to satisfactorily classify worm instances as intrusions. As with the precision, the detection accuracy increased with increasing threshold values and longer timing window. Figures 6.7 and 6.9 shows the accuracy of worm detection for each of the three schemes with the CDX and LAB network traffic respectively, which exhibit similar trend of increasing accuracy across rising threshold values and longer timing window.

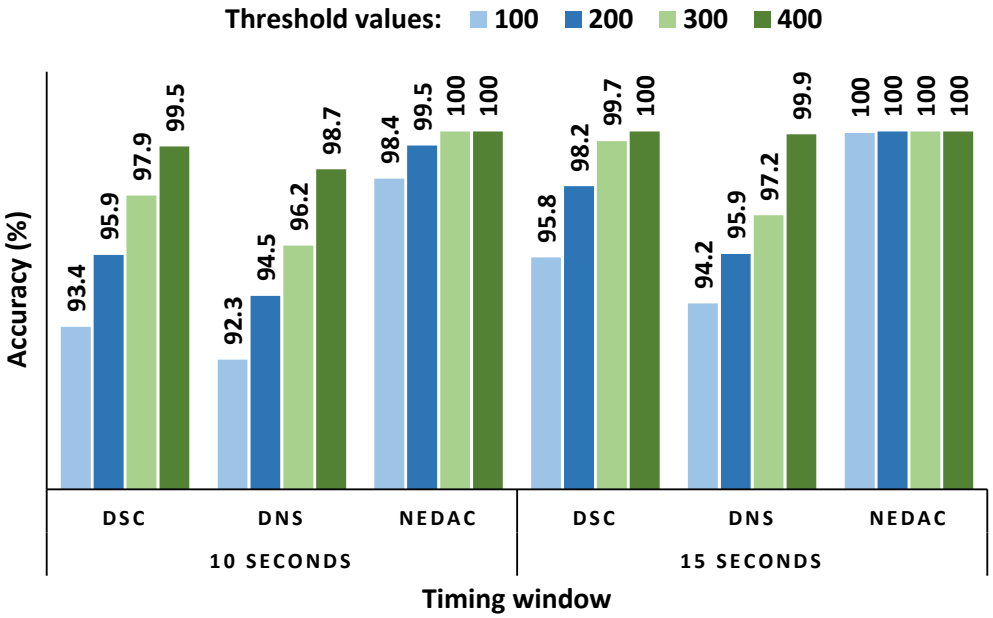


Figure 6.7: Accuracy of detection with the DARPA network traffic

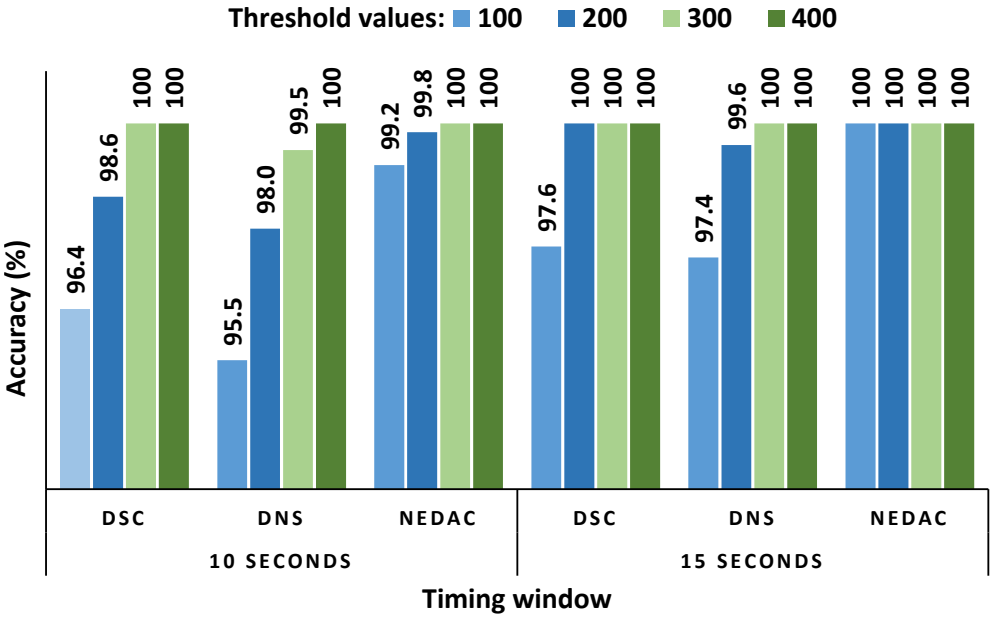


Figure 6.8: Accuracy of detection with the CDX network traffic

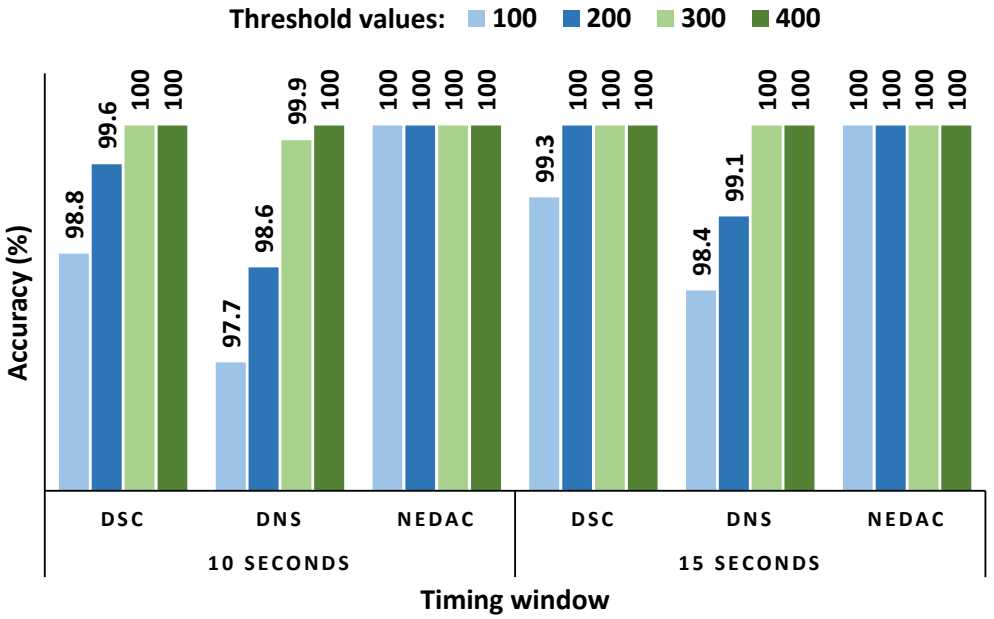


Figure 6.9: accuracy of detection with the LAB network traffic

6.3.3 Performance

The general performance of the detection schemes shows that the trend is for the precision and accuracy to increase with higher threshold values and a longer timing window, i.e., increasing the timing window and/or threshold value provide an additional level

of precision and accuracy of worm detection. However, increasing the timing window provides more precision and accuracy with fewer malware traffic propagation than increasing the threshold value. The detection schemes also performed better with the CDX and LAB datasets due to fewer false positives compared to the DARPA dataset.

Figure 6.10 presents the average detection performance of DSC, DNS-RL and NEDAC based on the threshold values used across all the experiments conducted with the DARPA, CDX and LAB datasets. Generally, the NEDAC scheme demonstrated a better detection performance than the DSC and DNS-RL with minimum average values of 99.1% and 99.6% for precision and accuracy respectively.

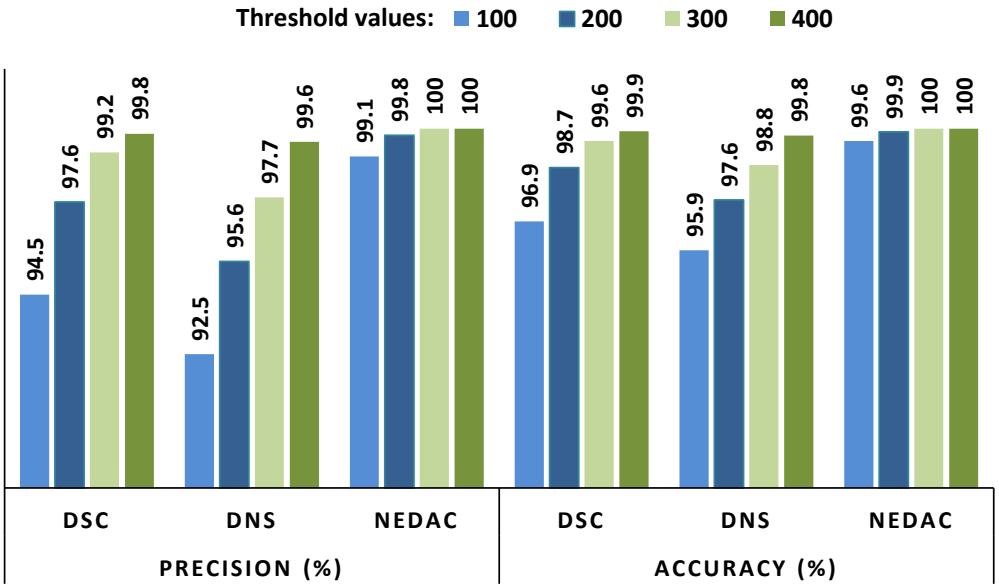


Figure 6.10: Average detection performance for the three schemes

6.4 Worm Containment Results

The worm containment performance of the DSC, DNS-RL and NEDAC schemes was evaluated using random and hit-list scanning infections as discussed in Section 5.7.1. The experiments were initially conducted without any countermeasure solution in place to determine the time taken to attain full infection. The experiments were then repeated in the presence of each detection scheme to evaluate their countermeasure capabilities.

The results of the experiments have been scaled up by factors of 32, 1.3 and 2.5 for the Slammer, RDP and ShellShock pseudo-worms respectively for client hosts in order to conform to the actual scan rate of the worm daemons. This is because the scan rates of worm daemons were scaled down before the experiments as explained in Section 5.6.1. The results of the worm containment performance using a threshold value of 100 are presented in Sections 6.4.1 through 6.4.3.

6.4.1 Results for the Containment of Random Scanning Worm

This section presents the results of the containment of random scanning infection using the timing windows of 10, 15, and 20 seconds.

6.4.1.1 Results for the Timing Window of 10 Seconds

This section presents the results of random infection behaviour using a timing window of 10 seconds. Figures 6.11, 6.12 and 6.13 show the number of infected hosts per second using the Slammer, RDP and ShellShock pseudo-worms respectively. The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 120 seconds with no detection scheme and infected 104 (10%) and 184 (17%) of its vulnerable hosts in the same period of time with the DSC and DNS-RL schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 120 seconds with no detection scheme and infected 208 (18%) and 235 (20%) of its vulnerable hosts in the same period of time with the DSC and DNS-RL schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 2000 seconds with no detection scheme and infected 200 (20%) and 322 (32%) of its vulnerable population in 2000 seconds with the DSC and DNS-RL schemes respectively. The Slammer, RDP and ShellShock infection rates remained zero with the NEDAC scheme.

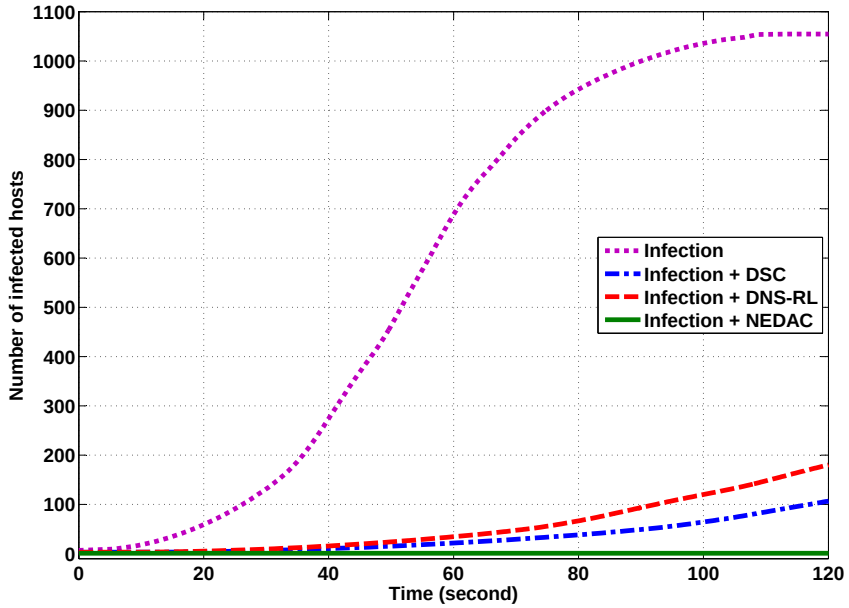


Figure 6.11: Slammer Random infection behaviour with 10 seconds timing window

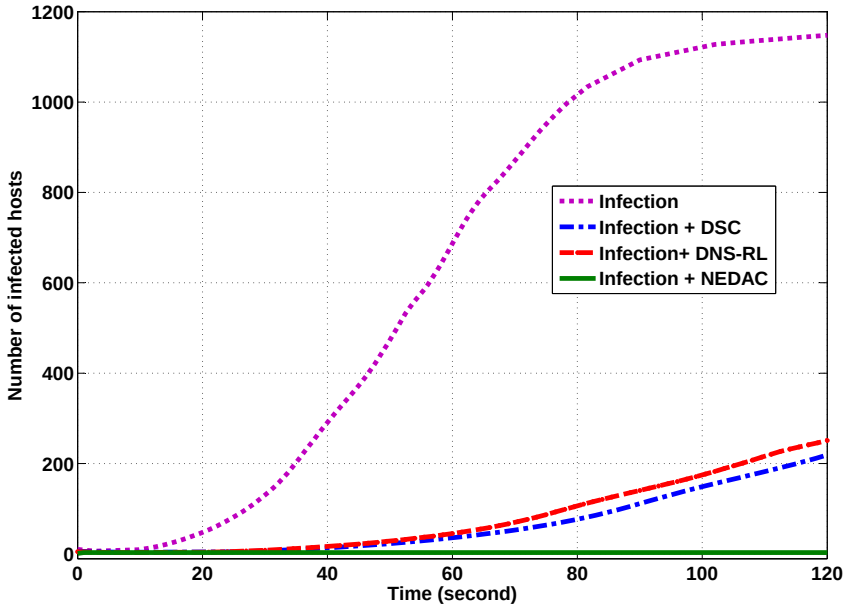


Figure 6.12: RDP Random infection behaviour with 10 second timing window

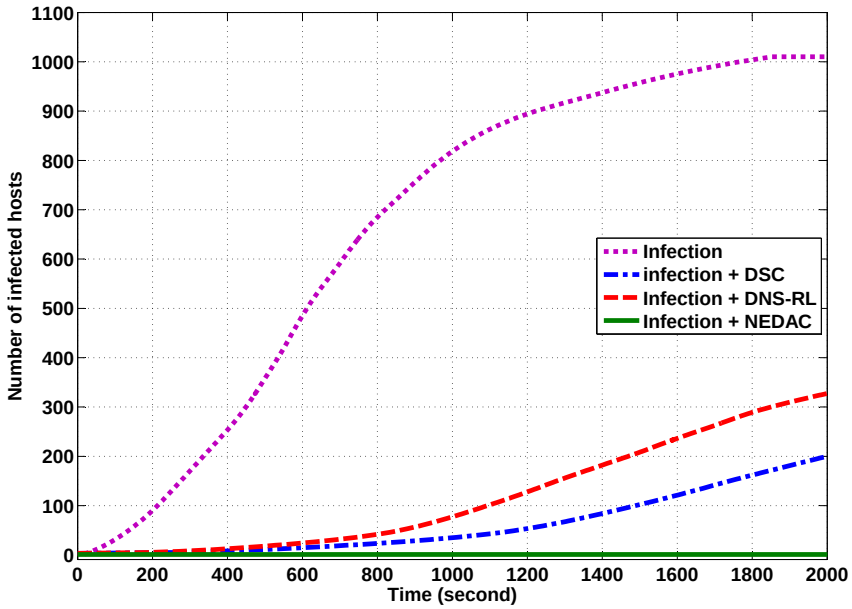


Figure 6.13: ShellShock Random infection behaviour with 10 second timing window

This is because the initially infected host, for each pseudo-worm experiment, was detected and then blocked from sending out datagrams at the data link layer. Additionally, inbound worm datagrams were also blocked at the network layer, which stopped the infection completely for each of the three worm outbreak scenarios. Generally, when the detection schemes were applied, the infections were significantly suppressed to at most 20% and 32% of the vulnerable populations by the DSC and DNS-RL schemes respectively and blocked completely by the NEDAC scheme. Although the worm infections were detected by the DSC and DNS-RL schemes and countermeasure solutions were applied, the worm spread because the initially infected host continued sending infectious datagrams and therefore the internal network hosts were left open to the worm attack.

6.4.1.2 Results for the Timing Window of 15 Seconds

This section presents the results of random infection behaviour using a timing window of 15 seconds. Figures 6.14, 6.15 and 6.16 show the number of infected hosts per second using the Slammer, RDP and ShellShock pseudo-worms respectively.

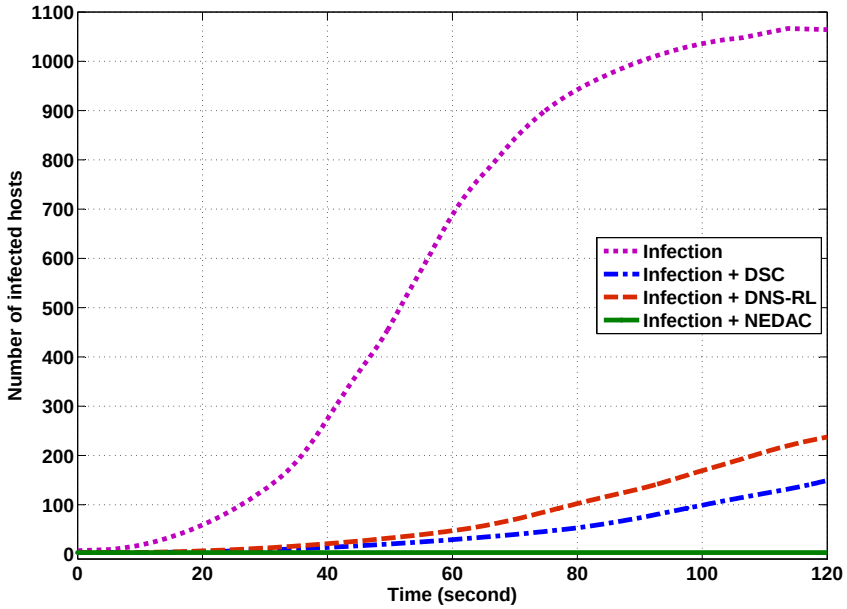


Figure 6.14: Slammer Random infection behaviour with 15 seconds timing window

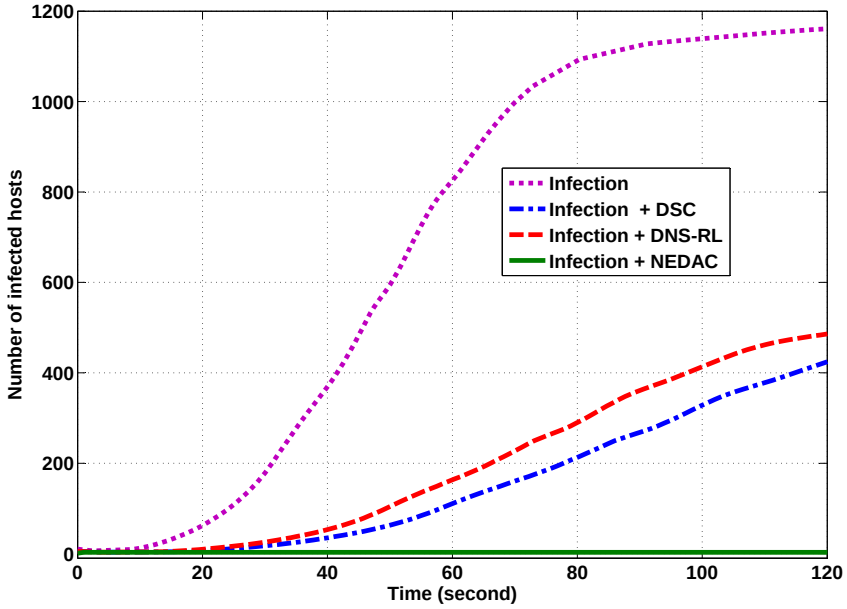


Figure 6.15: RDP Random infection behaviour with 15 second timing window

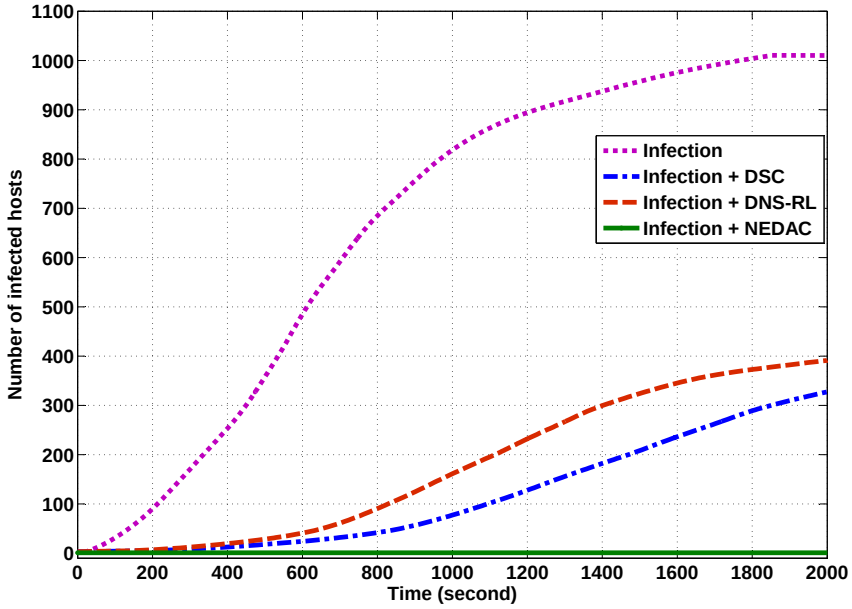


Figure 6.16: ShellShock Random infection behaviour with 15 second timing window

The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 120 seconds with no detection scheme and infected 153 (14%), 247 (23%) and three of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 120 seconds with no detection scheme and infected 438 (38%), 492 (42%) and two of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 2000 seconds with no detection scheme and infected 326 (32%), 398 (40%) and one of its vulnerable population in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively.

In contrast to the results of 10 seconds timing window, the infection rates of the pseudo-worms increased, on average, by 12%, 11% and 0.2% for the DSC, DNS-RL and NEDAC schemes respectively. The 50% increase in the timing window enabled the worms to transmit more infectious datagrams before exceeding the assigned threshold. However, the infections were suppressed to at most 38%, 42% and three of the vulnerable

populations by the DSC, DNS-RL and NEDAC schemes respectively.

6.4.1.3 Results for the Timing Window of 20 Seconds

This section presents the results of random infection behaviour using a timing window of 20 seconds. Figures 6.17, 6.18 and 6.19 show the number of infected hosts per second using the Slammer, RDP and ShellShock pseudo-worms respectively. The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 120 seconds with no detection scheme and infected 198 (20%), 286 (28%) and five of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 120 seconds with no detection scheme and infected 495 (42%), 516 (44%) and two of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 2000 seconds with no detection scheme and infected 419 (42%), 500 (50%) and one of its vulnerable population in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively.

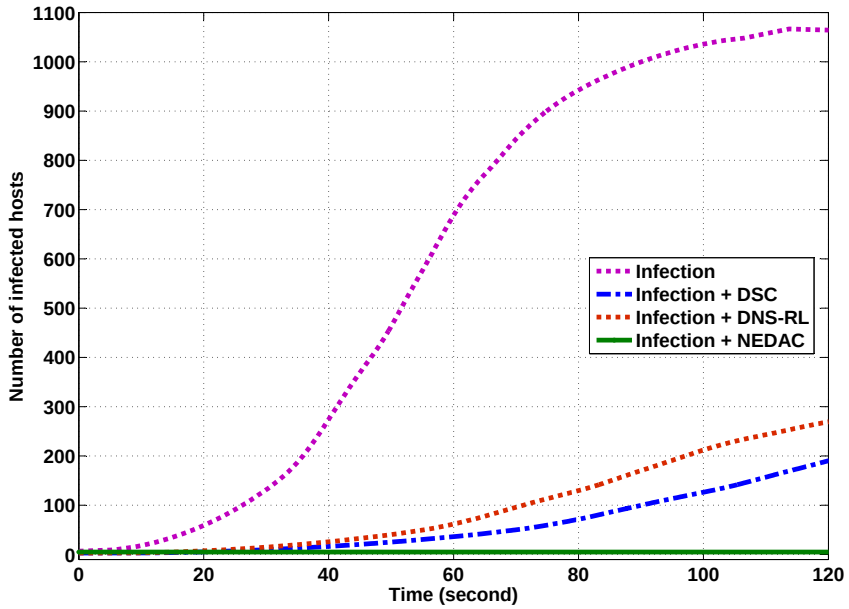


Figure 6.17: Slammer Random infection behaviour with 20 seconds timing window

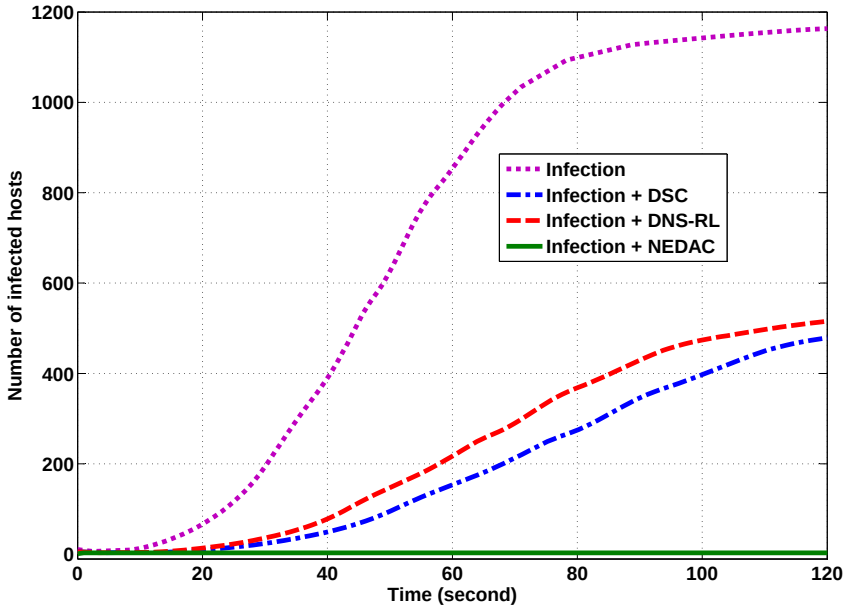


Figure 6.18: RDP Random infection behaviour with 20 second timing window

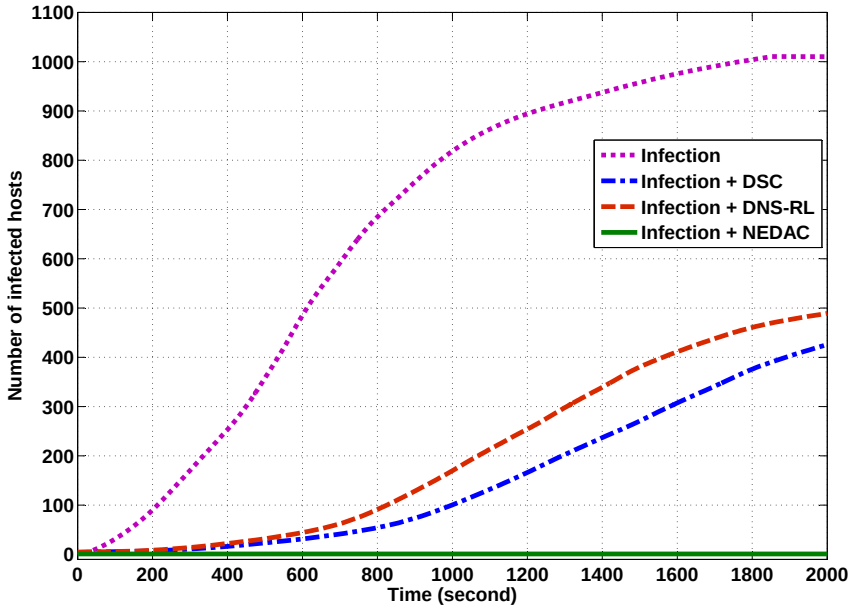


Figure 6.19: ShellShock Random infection behaviour with 20 second timing window

The infection rates of the pseudo-worms increased, on average, by 5% for the DSC and DNS-RL schemes and 0.1% for the NEDAC scheme in comparison with the results for

the 15 seconds timing window.

6.4.1.4 Performance

Table 6.1 summarises the number of hosts infected across the experiments conducted using random scanning within the relevant infection time of each pseudo-worm. Generally, the trend is that the infection increases with an increasing timing window.

Table 6.1: Number of Infected Hosts with the three Schemes using Random Scanning

Timing Window	Scheme	Slammer	RDP	ShellShock	Average (%)
10 Seconds	DSC	104 (10%)	208 (18%)	200 (20%)	16
	DNS-RL	184 (17%)	235 (20%)	322 (32%)	23
	NEDAC	1	1	1	1
15 Seconds	DSC	153 (14%)	438 (38%)	326 (32%)	28
	DNS-RL	247 (23%)	492 (42%)	398 (40%)	34
	NEDAC	4 (0.4%)	3 (0.3%)	2 (0.2%)	0.3
20 Seconds	DSC	198 (20%)	495 (42%)	419 (42%)	33
	DNS-RL	286 (28%)	516 (44%)	500 (50%)	39
	NEDAC	6 (0.6%)	3 (0.3%)	2 (0.2%)	0.4

6.4.2 Results for the Containment of Hit-list Scanning Worm

This section presents the results of the containment of hit-list scanning infection using the timing windows of 10, 15, and 20 seconds.

6.4.2.1 Results for the Timing Window of 10 Seconds

This section presents the results of hit-list infection using a timing window of 10 seconds and pre-compiled lists of 11, 12 and 10 hosts for the Slammer, RDP and ShellShock pseudo-worms respectively as discussed in Section 5.7.1. Figures 6.20, 6.21 and 6.22 show the number of infected hosts per second with the Slammer, RDP and ShellShock pseudo-worms respectively.

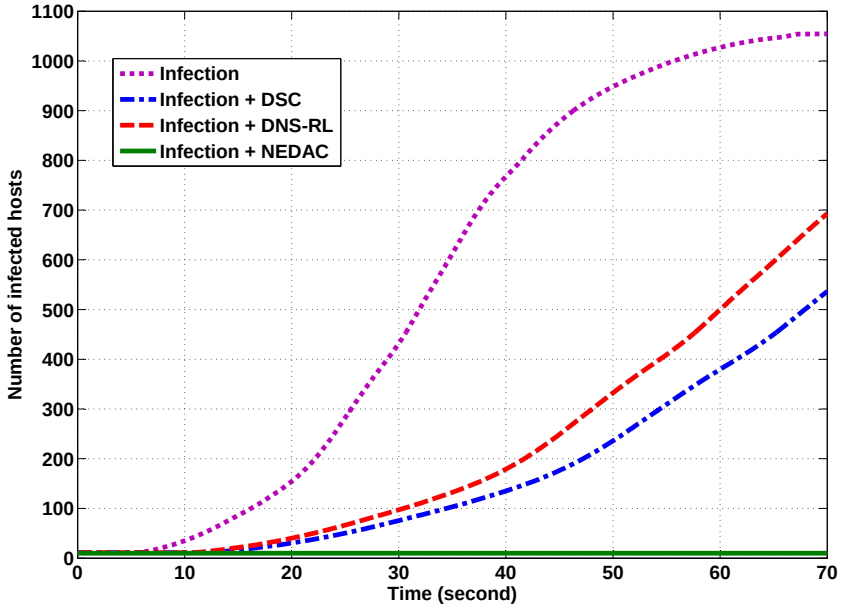


Figure 6.20: Slammer with a timing window of 10 seconds and a hit-list of 11 hosts

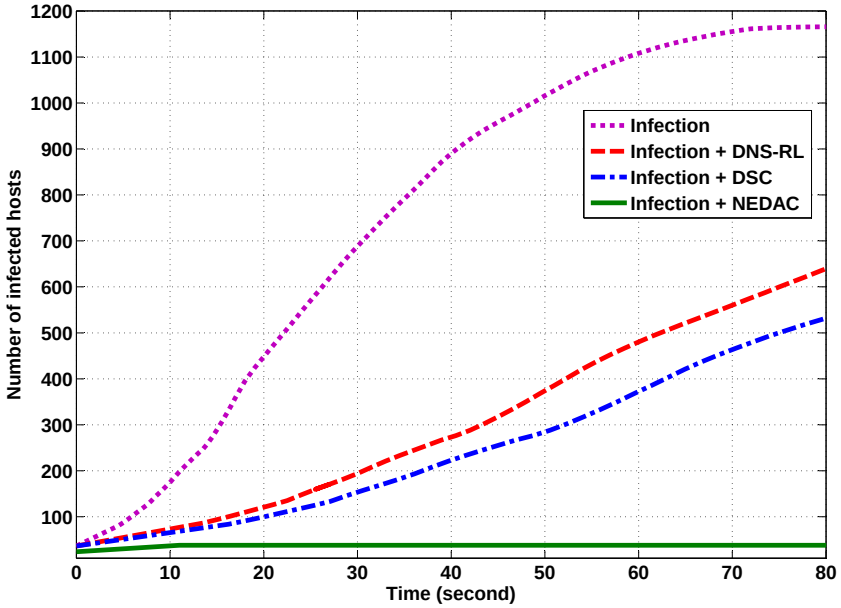


Figure 6.21: RDP with a timing window of 10 seconds and a hit-list of 12 hosts

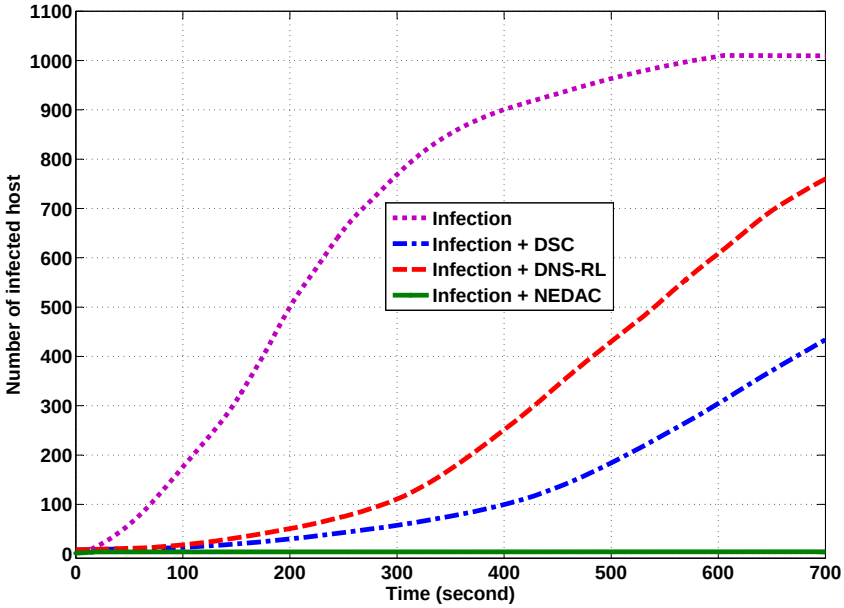


Figure 6.22: ShellShock with a timing window of 10 seconds and a hit-list of 10 hosts

The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 70 seconds with no detection scheme and infected 541 (51%), 700 (66%) and 18 (2%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 80 seconds with no detection scheme and infected 532 (46%), 644 (55%) and 38 (3.5%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 700 seconds with no detection scheme and infected 447 (44%), 781 (78%) and 13 (1.3%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively.

6.4.2.2 Results for the Timing Window of 15 Seconds

This section presents the results of hit-list infection using a timing window of 15 seconds and pre-compiled lists of 11, 12 and 10 hosts for the Slammer, RDP and ShellShock pseudo-worms respectively. Figures 6.23, 6.24 and 6.25 show the number of infected

hosts per second with the Slammer, RDP and ShellShock pseudo-worms respectively.

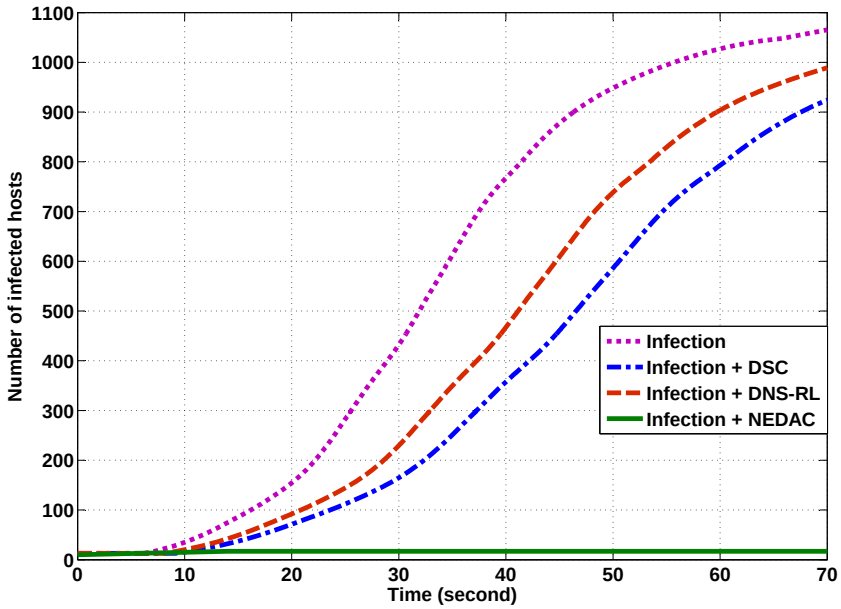


Figure 6.23: Slammer with a timing window of 15 seconds and a hit-list of 11 hosts

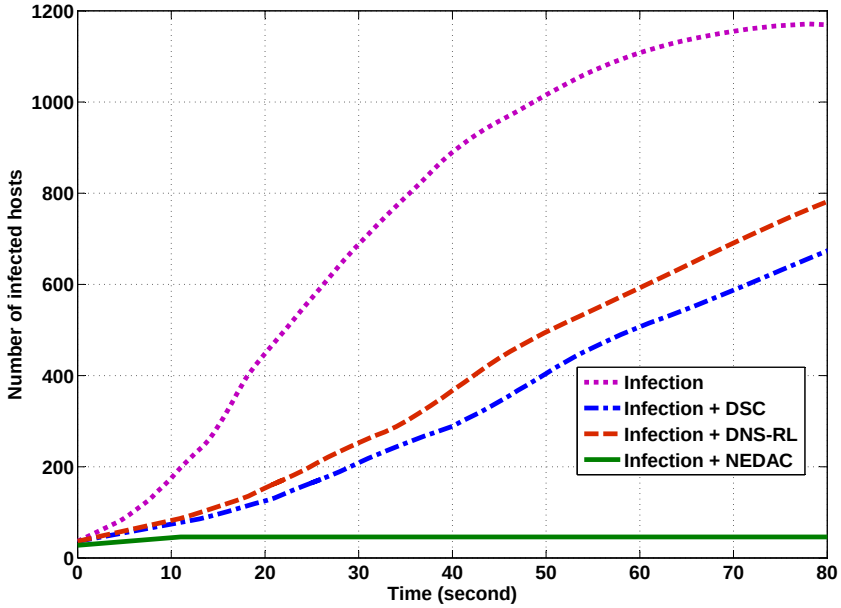


Figure 6.24: RDP with a timing window of 15 seconds and a hit-list of 12 hosts

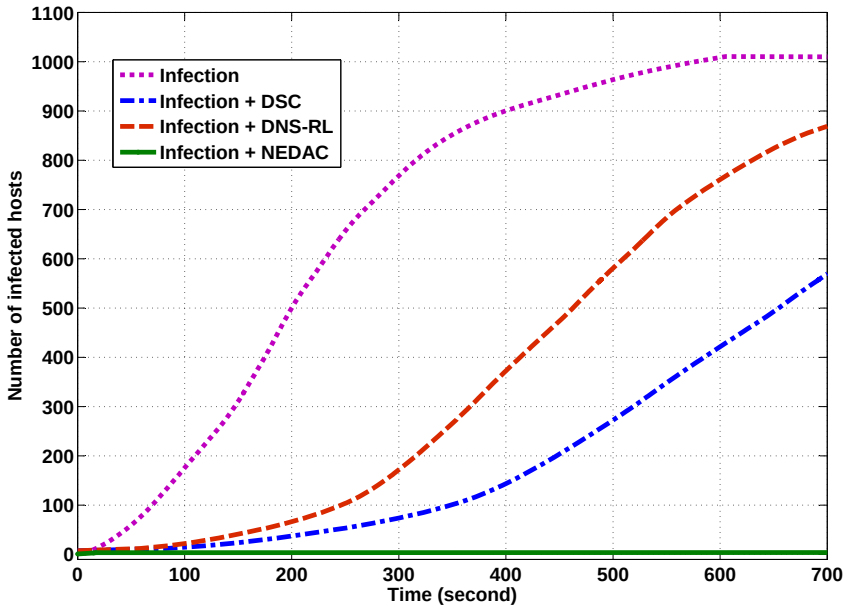


Figure 6.25: ShellShock with a timing window of 15 seconds and a hit-list of 10 hosts

The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 70 seconds with no detection scheme and infected 926 (88%), 998 (94%) and 20 (2%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 80 seconds with no detection scheme and infected 693 (59%), 797 (68%) and 46 (4%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 700 seconds with no detection scheme and infected 591 (59%), 887 (88%) and 18 (2%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively.

6.4.2.3 Results for the Timing Window of 20 Seconds

This section presents the results of hit-list infection using a timing window of 20 seconds and pre-compiled lists of 11, 12 and 10 hosts for the Slammer, RDP and ShellShock pseudo-worms respectively. Figures 6.26, 6.27 and 6.28 show the number of infected

hosts per second with the Slammer, RDP and ShellShock pseudo-worms respectively.

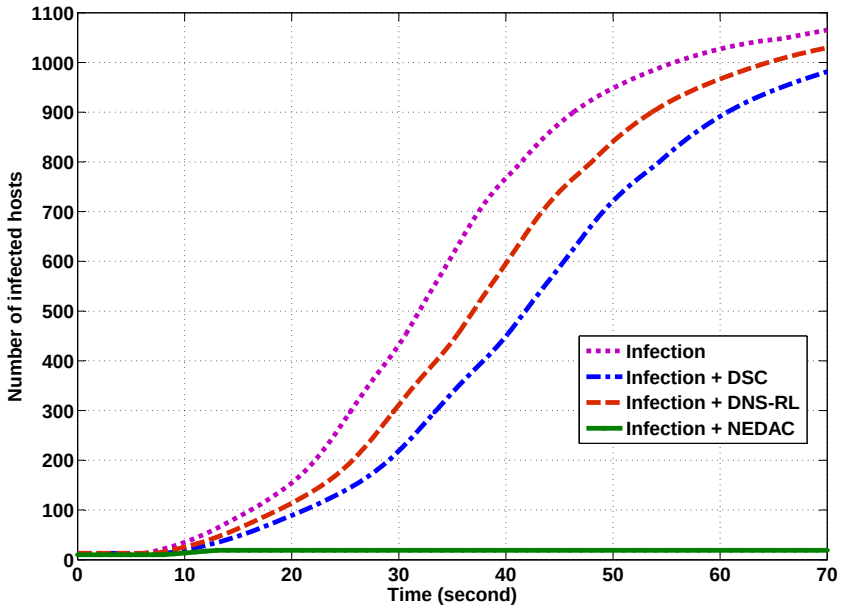


Figure 6.26: Slammer with a timing window of 20 seconds and a hit-list of 11 hosts

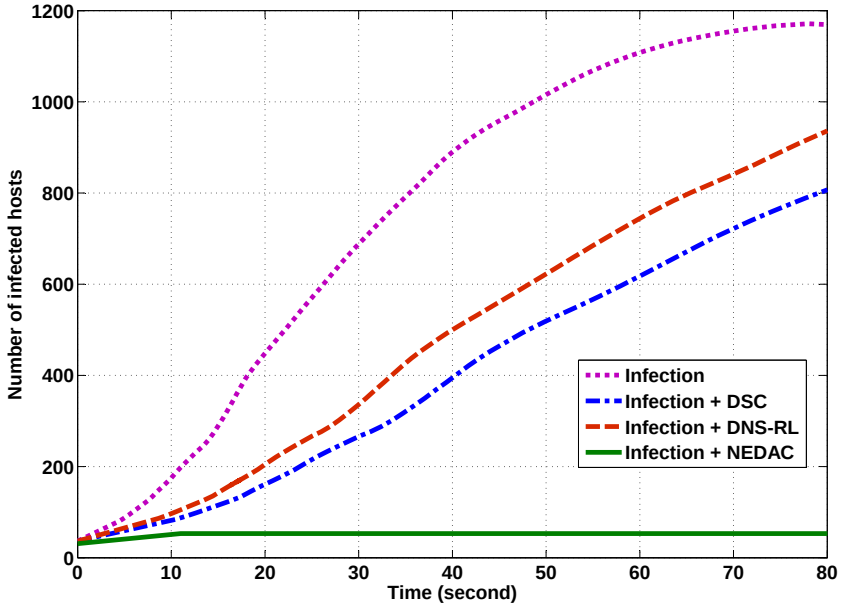


Figure 6.27: RDP with a timing window of 20 seconds and a hit-list of 12 hosts

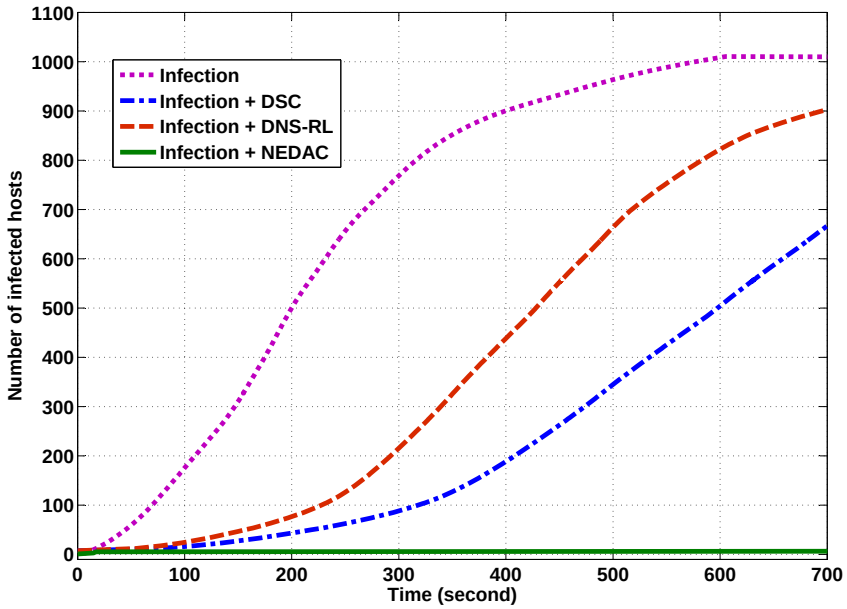


Figure 6.28: ShellShock with a timing window of 20 seconds and a hit-list of 10 hosts

The Slammer pseudo-worm attained full infection (1057 vulnerable hosts) in 70 seconds with no detection scheme and infected 996 (94%), 1027 (97%) and 24 (3%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The RDP pseudo-worm attained full infection (1168 vulnerable hosts) in 80 seconds with no detection scheme and infected 802 (69%), 931 (80%) and 53 (5%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively. The ShellShock pseudo-worm attained full infection (1006 vulnerable hosts) in 700 seconds with no detection scheme and infected 680 (68%), 900 (89%) and 21 (2%) of its vulnerable hosts in the same period of time with the DSC, DNS-RL and NEDAC schemes respectively.

6.4.2.4 Performance

Table 6.2 summarises the number of hosts infected across the experiments conducted using hit-list scanning within the relevant infection time of each pseudo-worm. The general trend is also for the infection to increase with an increasing timing window.

Table 6.2: Number of Infected Hosts with Hit-list Scanning

Timing Window	Scheme	Pseudo-worm (hit-list size)			
		Slammer (11)	RDP (12)	ShellShock (10)	Average (%)
10 Seconds	DSC	541 (51%)	532 (46%)	447 (44%)	47
	DNS-RL	700 (66%)	644 (55%)	781 (78%)	66
	NEDAC	18 (2)	38 (3.3%)	13 (1.3)	2
15 Seconds	DSC	926 (88%)	693 (59%)	591 (59%)	69
	DNS-RL	998 (94%)	797 (68%)	887 (88%)	83
	NEDAC	20 (2%)	46 (4%)	18 (2%)	3
20 Seconds	DSC	996 (94%)	802 (69%)	680 (68%)	77
	DNS-RL	1027 (97%)	931 (80%)	900 (89%)	89
	NEDAC	24 (3%)	53 (5%)	21 (2%)	3

6.4.3 Infected Hosts Contained by NEDAC

The RDP pseudo-worm experiments conducted using pre-compiled lists of 12 initially infected hosts spread the worm infection to a number of client and server hosts, which were all detected and contained completely by the NEDAC scheme. Table 6.3 summarises the number of infected client and server hosts that were contained by NEDAC due to either exceeding the threshold assigned for probing IPv4 routable address as set out in Section 5.7.1 or the threshold assigned for contacting inactive local IP address as discussed in Section 5.7.3.

Table 6.3: Infected Hosts Contained by NEDAC during RDP Hit-list Experiment

Hit-list Size	Infections		Host Type		Number of hosts Contained for Probing	
	Additional	Total	Client	Server	IPv4 Addresses	Inactive Local IP Addresses
12	26	38	35	3	37	1

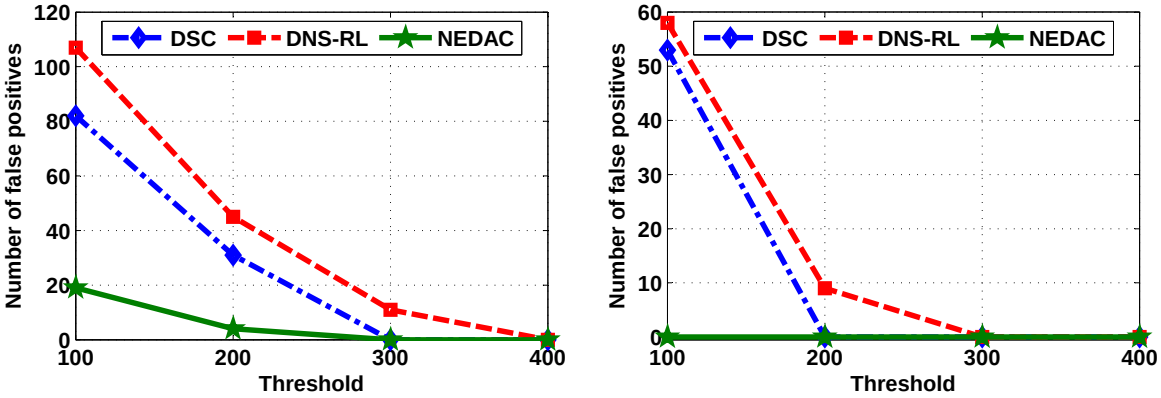
An additional number of 26 hosts were infected during the experiment with a pre-compiled list of 12 initially infected hosts. Three out of the 38 infected hosts were

servers that were contained for probing random IPv4 routable addresses. Among the 35 infected client hosts, a host was contained for probing an inactive local IP address. All other infected hosts in the set of experiments conducted were contained due to probing random IPv4 routable addresses. Across the sets of experiments conducted, most of the hosts were contained due to probing IPv4 routable addresses, because during worm scanning, the frequency of generating an IP address within the inactive local IP address space is lower compared to generating an IP address within the entire IPv4 routable address space.

6.5 False Positive Results

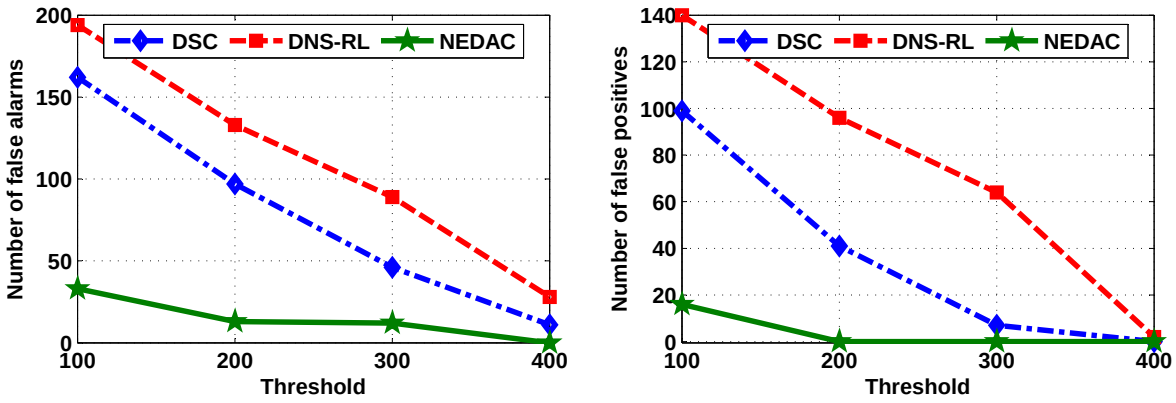
A false positive occurs when an alarm is raised, which has been caused by benign traffic. An alarm was considered to be an alert generated by a detection scheme for datagram(s) sent by a host on a distinct destination port, i.e., alerts raised by a detection scheme for datagrams destined for port X from host Y are considered to be a single alarm raised by the detection scheme for host Y. This is then recorded as an alarm from host Y until another alert is generated for datagrams sent by host Y to a different destination port.

All real pseudo-worm datagrams were detected by the schemes in all the experiments conducted and therefore the TP and FN rates are 100% and zero respectively. However, a number of false positives were raised by the schemes when employing timing windows of 10 and 15 seconds. The false positives dissipated with the timing window of 20 seconds. Figures 6.29, 6.30 and 6.31 present the number of false positives raised by the three schemes with the CDX, DARPA and LAB network traffic respectively.



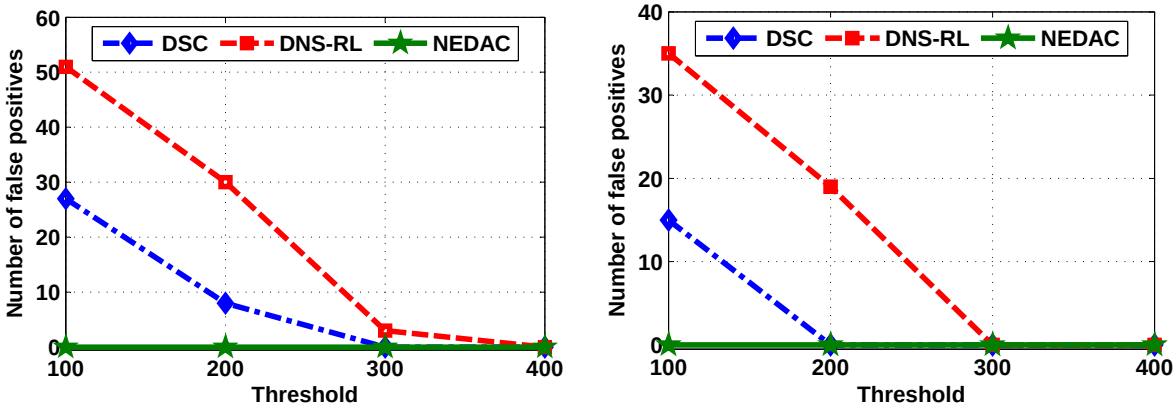
(a) Timing window of 10 seconds (b) Timing window of 15 seconds

Figure 6.29: False positives raised with CDX network traffic



(a) Timing window of 10 seconds (b) Timing window of 15 seconds

Figure 6.30: False positives raised with DARPA network traffic



(a) Timing window of 10 seconds (b) Timing window of 15 seconds

Figure 6.31: False positives raised with LAB network traffic

The schemes raised higher number of false positives with the DARPA network traffic than the CDX and LAB network traffic. These variations indicate the impact of network environment on the detection performance of the schemes. The schemes also have a higher number of false positives with low threshold values and timing window, which generally diminished with rising threshold values and longer timing window. For the former, the reduction was due to increasing the number of allowed benign datagrams that exhibit worm scanning-like behaviour. For the latter, the reduction was due to an increase in the amount of time for benign datagrams that exhibit worm scanning-like behaviour, which occur frequently within a short period of time. Section 6.5.1 through 6.5.3 present discussions of the network protocols in the DARPA, CDX and LAB network traffic that caused false positives for the DSC, DNS-RL and NEDAC schemes.

6.5.1 False Positive Analysis for the DARPA Traffic

The false positives raised by the DSC, DNS-RL and NEDAC schemes, with DARPA network traffic, were mainly datagrams destined for TCP port 23 (Telnet), UDP ports 53 (DNS), 123 (NTP), 137 (NetBIOS name resolution), 138 (NetBIOS datagram services) and 520 (routing information protocol) and other network ports (see Tables B.3 and B.4 in Appendix B for details). Figure 6.32 summarises the TCP and UDP ports that caused false positives for the three schemes. The common cause of false positives was a multicast UDP datagram destined for UDP port 520.

The UDP multicast datagram was an RIP update datagram between the internal gateway and an external router in the MIT Lincoln laboratory. RIP update datagrams appeared in the trace because the packet sniffer used to collect the traces was positioned between the gateway of the "outside" network and an external router that connects the "inside" and "outside" network segments. Ideally, the worm detection schemes, particularly NEDAC, were designed to work at the gateway interface of a protected network and monitor traffic to and from internal network hosts, where it is less likely for the schemes to encounter RIP update datagrams.

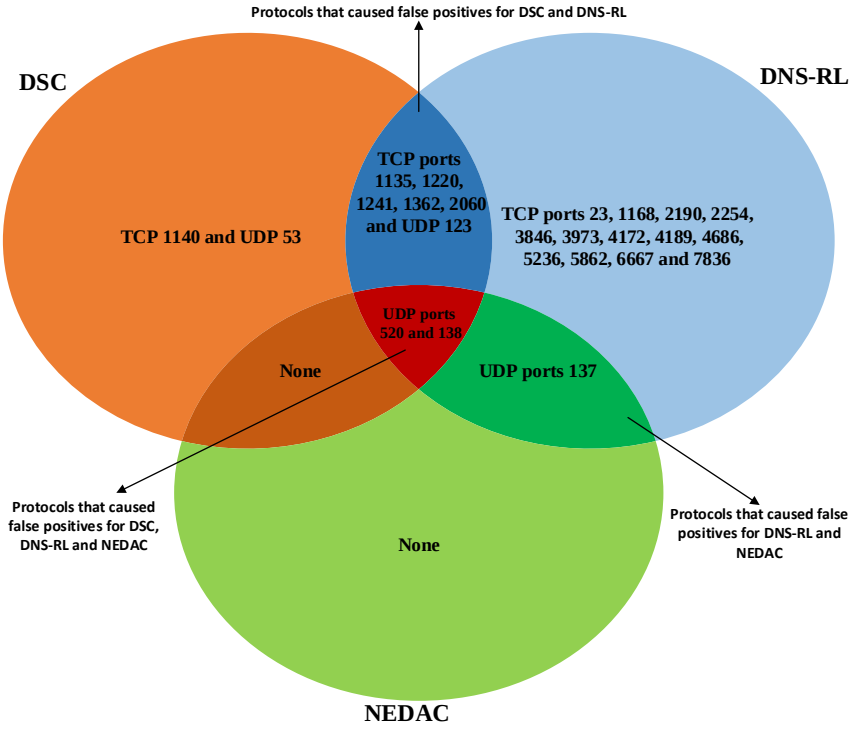


Figure 6.32: Protocols that causes false positives in the DARPA network traffic

However, the rate at which RIP sends updates to neighbouring routers is not similar to fast scanning network worm behaviour, because RIP routers exchange complete updates every 30 seconds by default (Malkin, 1998). Nevertheless, the RIP port can be added into the exempt list in NEDAC to avoid false positives, which are less likely to occur when the mechanism is deployed in a network.

In addition to the RIP update datagrams, the schemes also raised common false positives for NetBIOS service datagrams, i.e. datagrams destined for UDP ports 137 (broadcast NetBIOS name resolutions) and 138 (datagram services). The NetBIOS name service manages name resolution for NetBIOS hosts on a local network, while the datagram service is used to send messages to unique or multiple NetBIOS hosts (Appanasamy, 2013). A NetBIOS host resolves a name using a cache, a name server or an IP subnet broadcast (Davies, 2008). An IP subnet broadcast sent by a host is received by all machines on the same network. Similarly, a NetBIOS host uses the datagram service to send messages to a group of hosts, which also uses subnet broadcast for name resolution.

These behaviours are similar to worm scanning because a host is sending datagrams to different IP addresses on the same destination ports without a prior DNS resolution. Additionally, the response sent by a host upon receiving a NetBIOS service datagram is triggered by an inbound datagram received on the same port, which also resembles worm scanning for the DSC scheme and the NEDAC server anomaly detection component. However, the rate at which the datagrams are transmitted by name resolution and datagram services are not similar to fast scanning network worms.

Furthermore, the DSC and DNS-RL schemes raised common false positives for datagrams destined for TCP ports 1135, 1220, 1241, 1362 and 2060 and then UDP port 123. The DSC scheme also raised false positives for datagrams destined for UDP port 53 due to the occurrence of client host DNS resolution datagrams, which were triggered by DNS response datagrams received on the same port. The DNS-RL scheme raised false positives for datagrams destined for TCP port 23 and other network ports in the DARPA network traffic.

6.5.2 False Positives Analysis for the CDX Traffic

The false positives raised by the DSC, DNS-RL and NEDAC schemes, with the CDX network traffic, were datagrams destined for TCP ports 21 (FTP), 25 (SMTP), 80 (HTTP), 443 (HTTPS) and 445 (Microsoft SMB protocol), and also UDP ports 53, 67 (DHCP server), 68 (DHCP client), 123 (NTP), 137 (NetBIOS name resolution) and 138 (NetBIOS datagram services). Figure 6.33 summarises the TCP and UDP ports that caused false positives for the three schemes (see Tables B.1 and B.2 in Appendix B for details).

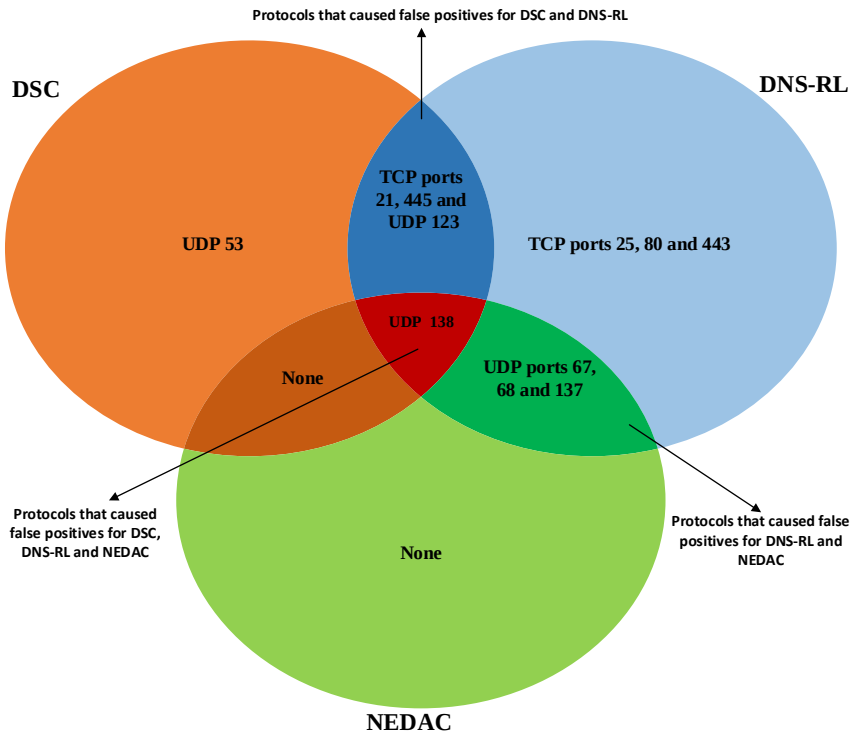


Figure 6.33: Protocols that causes false positives in the CDX network traffic

As with the DARPA network traffic, the common cause of false positives for the three detection schemes were NetBIOS name resolution (for DNS-RL and NEDAC only) and datagram services. The DNS-RL and NEDAC schemes raised common false positives for DHCP client requests and server reply datagrams destined for UDP ports 67 and 68 respectively. The DHCP datagrams destined for UDP 67 were IP address renewals/requests sent by hosts during system start up using numeric broadcast or DHCP server IP addresses. Then the DHCP server responds to the requests by sending reply datagrams destined for UDP port 68 in order to assign a new IP address or renew an existing lease. Both DHCP client requests and server responses exhibit worm scanning-like behaviour due to multiple contacts to different destinations on the same port, particularly when a number of network hosts are booted within a short period of time. However, the rate at which the DHCP requests and responses occur is not similar to fast scanning network worm behaviour. Additionally, most IP address request/renew datagrams do not go through a network router/gateway unless where such a router/gateway is also the local DHCP server or a relay agent. Nevertheless, the false positives can be resolved with

the NEDAC scheme by white-listing the IP address of the router and the DHCP port number combination when the mechanism is deployed in a network.

The DSC and DNS-RL raised common false positives for datagrams destined for TCP ports 21 and 445 and UDP port 123. The Microsoft SMB protocol (TCP port 445) is used for file and printer sharing directly over TCP unlike the previous version of the protocol that requires NetBIOS layer using UDP ports 137 and 138 and TCP port 139 (Martin, 2003). This is to simplify the SMB traffic by replacing the NetBIOS broadcast name resolution with DNS resolution (MacDonald and Barkley, 2000). As with the DARPA network traffic, the DSC scheme raised false positives for datagrams destined to UDP port 53, while the DNS-RL scheme raised false positives for datagrams destined for TCP ports 25, 80 and 443.

6.5.3 False Positives Analysis for the LAB Traffic

The false positives raised by the DSC and DNS-RL schemes, with the LAB network traffic, were datagrams destined for TCP ports 25, 80, 443, 445 and also UDP ports 123, 137, 138. Figure 6.34 summarises the TCP and UDP ports that caused false positives for either or both of the DSC and DNS-RL schemes (see Tables B.5 and B.6 in Appendix B for details). The most common cause of false positives were datagrams addressed to TCP port 445 and UDP ports 123 and 138 as with the DARPA and CDX network traffic. The DNS-RL scheme raised false positives for datagrams destined for TCP ports 25, 80, 443 and 1935 (Adobe Systems Macromedia Flash Real Time Messaging Protocol) and then UDP port 5355 (LLMNR). The LLMNR protocol enables name resolution within a subnet or link-local scoped network, especially when DNS name resolution fails (Aboba et al., 2007; Jeong et al., 2003).

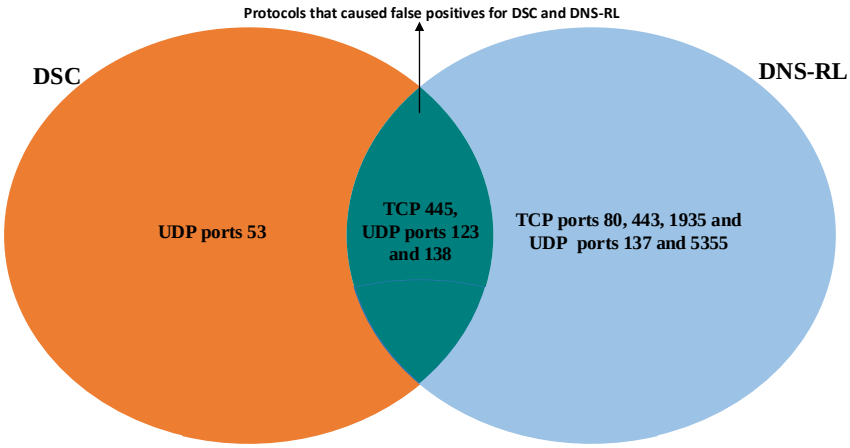


Figure 6.34: Protocols that causes false positives in the Lab network traffic

An LLMNR host resolves a name by sending an LLMNR query to multicast addresses, which is destined for UDP port 5355. Then an LLMNR host that is authoritative to the name responds to the query by sending a unicast UDP response to the sender on the same port (Aboba et al., 2007). The multicast LLMNR query resembles a worm scanning activity because a multicast query sent to different addresses with the same destination port will cause an alarm by the DNS-RL scheme. However, LLMNR queries and responses are not routable, i.e. they do not pass through a router, and the rates at which the queries and responses occur are not similar to those of a fast scanning network worm.

6.5.4 NEDAC False Positive Detection for the Peak and Quiescent Periods

The NEDAC detection system used a dynamic scheme that assigns threshold based on peak and quiescent periods as explained in Section 3.3. The dynamic threshold scheme was tested using different volumes of background traffic as explained in Section 5.7.3. During the NEDAC evaluation, the effect of reducing the peak period threshold to 25%, 30% and 35% during the quiescent period was tested. Table 6.4 summarises the false positives raised by NEDAC with CDX and DARPA network traffic during the peak and quiescent periods.

Table 6.4: False positives raised by NEDAC during peak and quiescent periods

Network Traffic	Threshold	Total No. of FPs	Peak Period	Quiescent Period			
				25%	30%	35%	40%
10 seconds timing window							
CDX	100	19	16	3	3	3	0
	200	4	4	0	0	0	0
DARPA	100	31	27	4	2	2	2
	200	13	11	2	2	0	0
	300	8	8	0	0	0	0
15 seconds timing window							
DARPA	100	14	2	2	0	0	0

Generally, the NEDAC detection system raised a higher number of false positives per unit time during the peak period due to higher volume of background traffic. The NEDAC detection system raised 16 false positives using a threshold of 100 with the CDX network traffic during the peak period. The number of false positives then reduced to three during the quiescent period when the threshold was reduced by 25%, 30% and 35% and then dissipated when the threshold was reduced by 40%. Generally, the false positives raised by NEDAC with both CDX and DARPA network traffic dissipated with increasing threshold in peak or quiescent periods and longer timing window. NEDAC raised no false positives with the LAB network traffic.

6.6 Analysis

This section presents a brief analysis of the worm detection and containment performance of the schemes tested. An analysis of the detection performance is presented in Section 6.6.1, while Section 6.6.2 presents an analysis of the containment performance of the schemes.

6.6.1 Detection Performance

The false positives raised by the three detection schemes with the DARPA, CDX and LAB network traffic occurred due to the following:

- True false positives: These are legitimate and routable TCP and UDP datagrams that were marked as worm datagrams by the detection schemes.
- Non-routable datagrams: These are legitimate and non-routable datagrams that were marked as worm datagrams by the detection schemes. These are datagrams meant to be used within a subnet or local area network such as NetBIOS name resolution (UDP port 137) and datagrams services (UDP port 138), DHCP client request (UDP port 67) and server reply (UDP port 68) datagrams and LLMNR datagrams. With proper network configuration, these datagrams are less likely to pass through a router/gateway. However, where NetBIOS and DHCP protocols where configured to communicate across a router due to network requirements, the ports and/or participating hosts can be exempted in NEDAC when the mechanism is deployed in a network. This is to resolve the possibility of false positives although the rate at which the datagrams are transmitted is not similar to that of a fast scanning network worm.
- RIP update datagrams: These are route advertisement datagrams that are sent and received by neighbouring routers for route update information. These datagrams are also less likely to encounter with the detection schemes.

Thus, the true false positives raised by the detection schemes are those that were not caused by non-routable and RIP update datagrams. Tables C.1 through C.6 in Appendix C present the details of true false positives raised by the detection schemes. For example, the DSC and DNS-RL schemes raised 66 and 28 true false positives out of a total of 82 and 31 false positives with the CDX traffic using a threshold of 100 and a timing window of 10 seconds. Similarly, the reduction holds for the two schemes across all the experiments conducted with the three datasets. However, the NEDAC scheme raised no

true false positive across all the experiments conducted with the three datasets. Figure 6.35 presents the overall average detection performance of each scheme across all the conducted experiments.

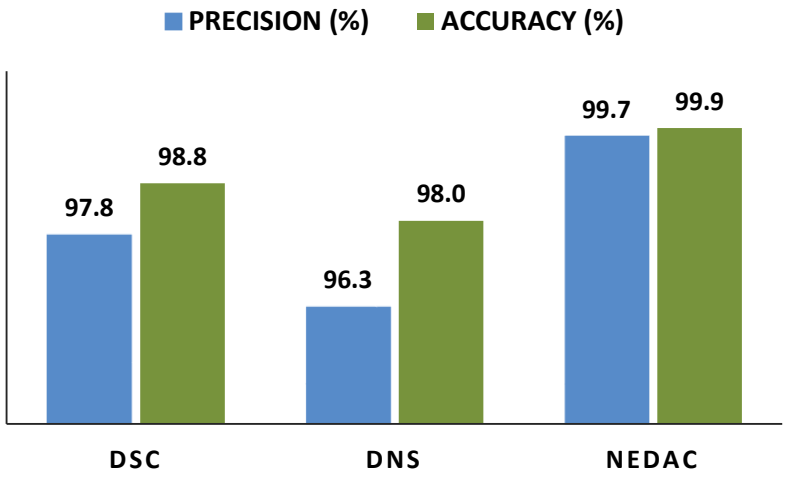


Figure 6.35: Overall average detection performance for the three schemes

Figures 6.36 and 6.37 present the ROC curve of each scheme with the DARPA and CDX network traffic. As noted in Section 6.5.4, NEDAC raised no false positive with the LAB network traffic.

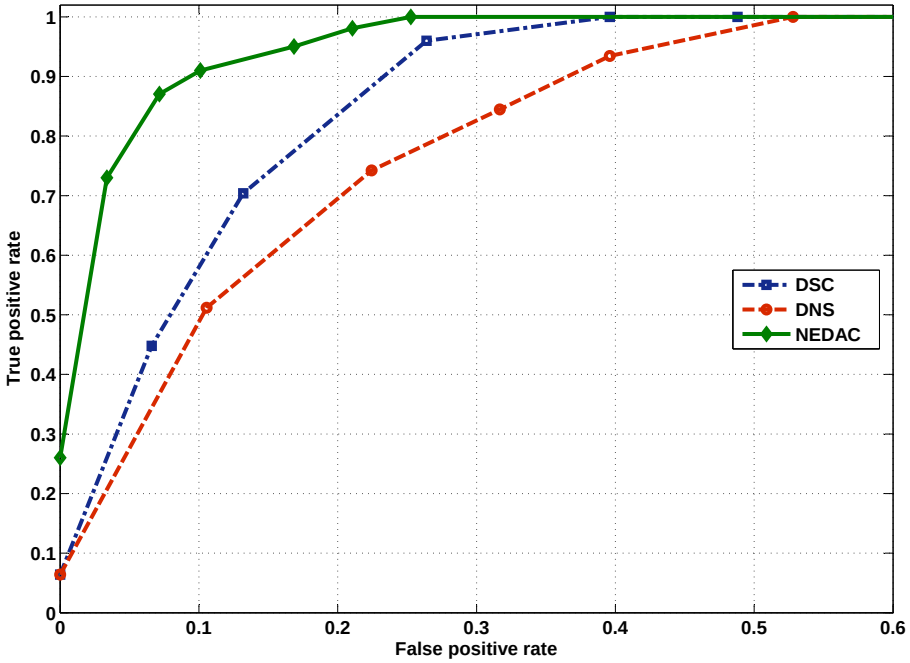


Figure 6.36: ROC curve for the three detection scheme with the DARPA traffic

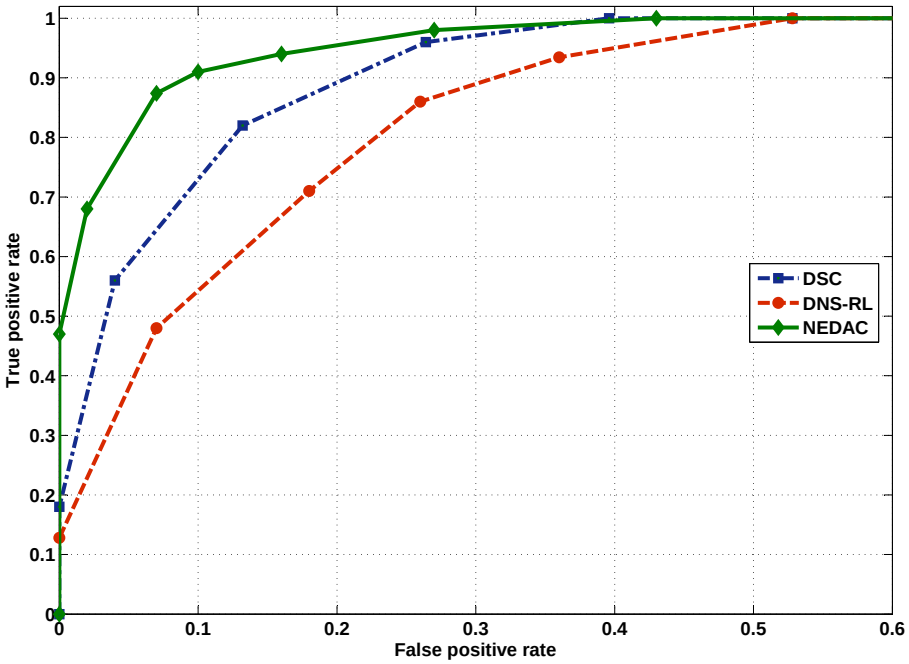


Figure 6.37: ROC curve for the three detection scheme with the DARPA traffic

Generally, the NEDAC scheme has demonstrated a better performance in differentiating worm traffic from benign traffic than the DSC and DNS-RL schemes, with an overall precision of 99.7% and an accuracy of 99.9%. Furthermore, the DSC and DNS schemes also showed a good sensitivity in detecting worm infection from their ROC curves for both DARPA CDX network traffic. NEDAC demonstrated a better sensitivity than the two detection schemes in both scenarios. These were achieved because the NEDAC scheme monitored and assigned a counter for each distinct destination port contacted by a host. This enables NEDAC to accurately identify fast scanning network worm traffic because the counter associated to the destination port used by a worm increased rapidly unlike other legitimate traffic.

6.6.2 Containment Performance

Worm containment involves blocking the propagation of a worm from an infected host to vulnerable hosts. The DSC and DNS-RL schemes used network level access control that blocked outbound worm traffic from infecting remote vulnerable hosts. This allowed the worm infection to continue to spread within the internal network and from external

networks as depicted in Figure 6.38. However, the NEDAC scheme used data link layer blocking to isolate an infected host, which prevented the worm infection from spreading locally and to remote hosts, and then blocked identified inbound worm traffic using network layer blocking as depicted in Figure 6.39.

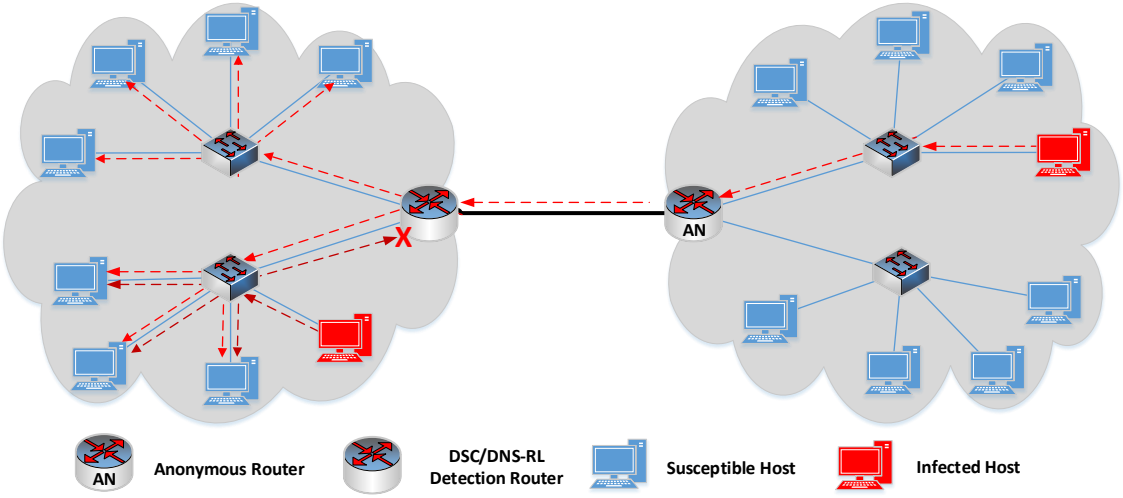


Figure 6.38: Worm containment scenario for the DSC and DNS-RL schemes

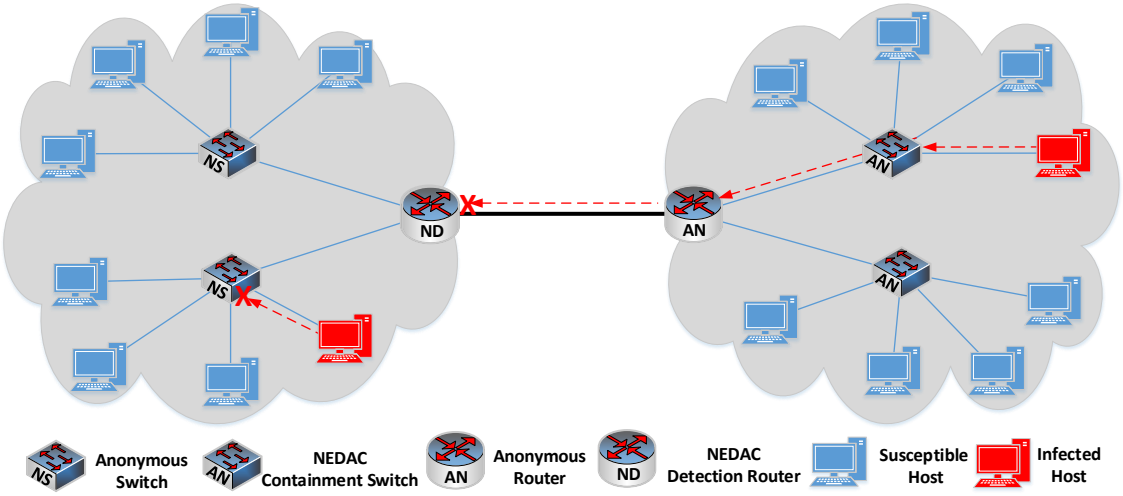


Figure 6.39: Worm containment scenario for the NEDAC schemes

Figure 6.40 presents the average percentage values to which the DSC, DNS-RL and NEDAC schemes suppressed the propagation of the pseudo-worms used in the evaluation experiments.

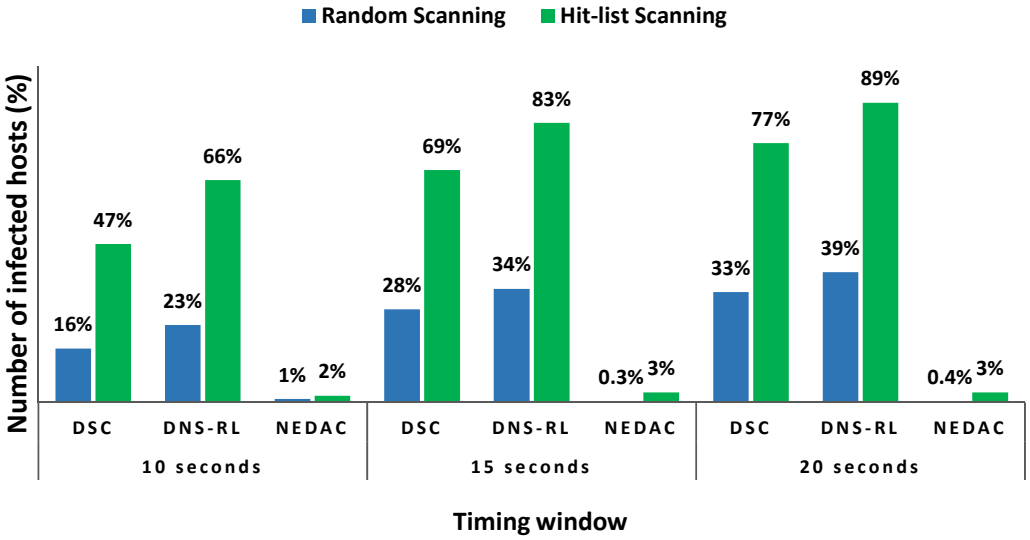


Figure 6.40: Average number of infected hosts with DSC, DNS-RL and NEDAC

The DSC and DNS-RL schemes suppressed the worm propagation to 33% and 39% of the vulnerable population with random scanning technique, while the NEDAC scheme suppressed the worm propagation to 1% of the vulnerable population. The suppression of hit-list worm propagation by the DSC and DNS-RL was less effective than the worm propagation with random scanning as the infection increased beyond 70% of vulnerable hosts. This is due to the large scale of continued scanning activities by the infected hosts after countermeasures were applied by the DSC and DNS-RL schemes. The NEDAC scheme suppressed the hit-list infection to 3% of the vulnerable population.

The NEDAC containment approach is more effective because the worm infection cannot spread within the internal network and there is no contribution from other infected hosts from external networks. Although the comparison is uneven because NEDAC used a different containment technique compared with that of the DSC and DNS-RL schemes, the empirical tests have demonstrated that using a data link layer containment solution is more effective in blocking outbound worm traffic from infecting internal and remote network hosts, although it consumes more switch processing power especially when multiple infected hosts (at least 20) in a network segment are scanning the Internet. Additionally, the network layer containment used by NEDAC to block inbound worm infection further enabled the scheme to protect internal vulnerable hosts against worm

infection coming from external networks. These enabled the NEDAC scheme to have a better containment performance than the DSC and DNS-RL schemes.

6.7 Summary

This chapter has presented the worm experiments conducted to test the NEDAC scheme together with two previously reported worm detection and containment schemes. The chapter has initially presented the experimentation set-up and network designs used for the candidate worms during the experiments. The chapter then presented the worm detection and containment results for the NEDAC scheme and the two previously reported schemes. A brief analysis of the overall results has been presented, which showed that NEDAC has a better worm detection and containment performance in comparison with the two previously reported detection schemes tested.

Chapter 7

CONCLUSIONS

7.1 Introduction

The early chapters of this thesis have presented an overview of fast scanning network worms, the vulnerabilities they exploit, their infection and propagation techniques and the threat they pose to the security of enterprise networks and the Internet. The thesis then presents a cross-layer mechanism, termed NEDAC, that uses a network level detection system and a data-link level containment system to provide a novel countermeasure solution for fast scanning network worms. Chapters four through six then present the testing and evaluation of NEDAC, demonstrating that it provides improved performance in comparison to previous methods. This chapter concludes the thesis by summarising the suggested original contributions, providing an overview of two developed research tools, and setting out some recommendations for further work.

7.2 Research Contributions

The suggested contributions of this research work are:

- *A better worm detection system*: The NEDAC detection system uses two separate and specific anomaly detection techniques for network client and server hosts. The detection system also uses an ARP-based detection technique for all network hosts in order to strengthen the detection capabilities of the scheme in identifying worm scans targeted towards inactive local IP addresses in a network as discussed in

Section 3.4. The advances provided by NEDAC enabled the scheme to demonstrate a substantial performance improvement over the worm detection schemes reported by Gu et al. (2004), Whyte et al. (2005) and Whyte et al. (2005) as presented in Section 6.3. This is because the NEDAC scheme has average precision of 99.7% and accuracy of 99.9%, while the DSC and DNS-RL have average precisions of 97.8% and 96.3%, and average accuracies of 98.8% and 98% respectively as presented in Section 6.6.1. The NEDAC detection system requires no training session and also uses TTL for traffic information recorded in caches as discussed in Section 3.4. These are considered to have resolved the issue of training-driven delays and provides resource efficiency as set out in Section 2.10.

- *A better worm containment system*: The NEDAC containment techniques used at the network and data-link levels enabled the scheme to demonstrate an improved containment capability. The containment system does not only isolate an identified infected host in a protected network, but also blocks inbound worm traffic from entering the network. Therefore the network and data-link containment techniques provide a combined containment solution that was capable of protecting vulnerable hosts from internal and external worm infection at an early stage as presented in Section 6.4. The use of the data link containment technique also enabled NEDAC to protect the network from local-to-local worm infections. The network and data link containment capabilities enabled NEDAC to suppress worm infection to, at most, 3% of its vulnerable population, unlike DSC and DNS-RL that allowed the worm to infect more than 70% of the vulnerable population as presented in Section 6.6.2.

These contributions are considered to have addressed the identified research gaps as set out in Section 2.10. However, the NEDAC scheme is limited to providing a worm countermeasure solution when deployed across the distribution and access layers of an enterprise network. The scheme cannot contain a worm infection when deployed in the core layer of a network unless where the core and distribution layers are merged together. The NEDAC performance evaluation is limited to a maximum scale of five enterprise

networks, with a network having a maximum number of eight subnets.

7.3 Research Tools

The thesis presented some research tools that have been developed to support the establishment of the research contributions presented in Section 7.2. These research tools are summarised as follows;

- *The use of pseudo-worms incorporating contemporary malware characteristics for evaluation*: The evaluation programme used to test the NEDAC scheme employed two contemporary pseudo-worms that have been developed based on two contemporary wormable vulnerabilities namely Microsoft RDP (CVE-2012-0002) of 2012 (CVE, 2012) and ShellShock (CVE-2014-6271) (CVE, 2014a). The pseudo-worms were used to evaluate the NEDAC scheme using new potential fast scanning network worms and to characterise the threat posed by the recent vulnerabilities. It is suggested that this is a step forward in testing the capabilities of worm detection systems.

- *An improved and suitable testbed for worm experimentation and testing countermeasure systems*: The research work developed the V-Network testbed with improved features such as a larger scale of experimentation, with 1200 virtual hosts and support for physical network integration, which is larger than the scale of the ViSe and VMT tesbeds that supported 10 and 300 hosts respectively. The V-Network testbed also has facilities for managing worm experiments and support for multiple operating systems, background traffic generation and replay. These are improvements over the vGround testbed (Jiang et al., 2006), the ViSe testbed (Årnes et al., 2006) and the VMT testbed (Shahzad et al., 2013).

It is suggested that these research tools addressed the objective of having an improved and suitable testing environment and evaluation mechanism as discussed in Section 2.11.

7.4 Recommendation For Further Work

It is recommended that future researchers in this field consider the following possible future work:

- *Threshold scheme*: The NEDAC detection system used a dynamic threshold scheme that assigned limits for anomalous behaviour based on the time of day, i.e, peak and quiescent periods. It is possible that the scheme can be further improved to assign a threshold that is relative to the network activity of a host or group of hosts at any given time.

- *Address and packet spoofing*: The NEDAC detection system used IP and ARP datagrams to detect anomalous behaviour from a network host. The NEDAC scheme has no ability to detect a worm that may spoof the IP address of a network host or employs an ARP poisoning attack. Similarly, the data-link layer containment system also has limited ability to detect a spoofed MAC address. It may be appropriate to explore ways in which NEDAC can be enhanced to detect spoofed addresses.

- *Testing the impact of different environment*: Network environments have different network traffic due to differences in the use of network protocols and applications. The performance of an anomaly-based detection scheme depends on the protocols and applications used in a network environment. The NEDAC scheme was tested using three different network traffic datasets, which represent a fraction of the different network environments connected to the Internet. Thus the behaviour of the NEDAC scheme can be further evaluated using different network traffic in order to examine how the scheme will adapt to changes in different network environments. The scheme can also be evaluated using larger number of enterprise networks and subnets.

7.5 Summary

This chapter has presented an overview of the contributions provided by this research work. The NEDAC worm countermeasure scheme presented in this thesis was comparatively evaluated with two previously reported worm detection schemes and empirically proven to provide a faster countermeasure solution with almost no false alarms. The NEDAC scheme therefore has the potentials to provide a countermeasure solution for the propagation of fast scanning network worms before causing any damage when deployed in networks. Thus, implementing the NEDAC scheme on routers and switches in networks will provide a countermeasure solution against fast scanning network worms.

REFERENCES

3Com (2016). *3Com OfficeConnect*. [Online]. Accessed 2nd June 2016. Available: http://www.mtmnet.com/PDFFILES/3CDSG10PWR.pdf.

Aboba, B., Esibov, L., and Thaler, D. (2007). Link-local multicast name resolution (llmnr).

Ahmad, A. and Dey, L. (2007). A k-mean clustering algorithm for mixed numeric and categorical data. *Data and Knowledge Engineering*, 63(2):503–527.

Ahmad, M. A. and Woodhead, S. (2015). Containment of fast scanning computer network worms. In *Internet and Distributed Computing Systems*, volume 9258 of *Lecture Notes in Computer Science*, pages 235–247. Springer International Publishing.

AlEroud, A. and Karabatis, G. (2013). Toward zero-day attack identification using linear data transformation techniques. In *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on*, pages 159–168. IEEE.

Aljawarneh, S. A., Moftah, R. A., and Maatuk, A. M. (2016). Investigations of automatic methods for detecting the polymorphic worms signatures. *Future Generation Computer Systems*, 60:67–77.

Appanasamy, P. (2013). Server message block and netbios.

Årnes, A., Haas, P., Vigna, G., and Kemmerer, R. A. (2006). Digital forensic reconstruction and the virtual security testbed vise. In *Detection of Intrusions and Malware and Vulnerability Assessment*, pages 144–163. Springer.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177.

Beigh, B. M. and Peer, M. (2014). Performance evaluation of different intrusion detection system: An empirical approach. In *Computer Communication and Informatics (ICCCI), 2014 International Conference on*, pages 1–7. IEEE.

Belhaouane, M., Garcia-Alfaro, J., and Debar, H. (2015). On the isofunctionality of network access control lists. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, pages 168–173. IEEE.

Bencsáth, B., Pék, G., Buttyán, L., and Felegyhazi, M. (2012). The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971–1003.

Berghel, H. (2001). The code red worm. *Communications of the ACM*, 44(12):15–19.

Bierman, A., Huang, L., and Clemm, A. (2013). Yang data model for stateless packet filter configuration.

Calheiros, R. N., Buyya, R., and De Rose, C. A. (2010). Building an automated and self-configurable emulation testbed for grid applications. *Software: Practice and Experience*, 40(5):405–429.

CERT, A. (2001). *Nimda Worm*. Available: http://www.cert.org/advisories/.

Chang, X. (1999). Network simulations with opnet. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 307–314. ACM.

Chate, A. and Chirchi, V. (2015). Access control list provides security in network. *International Journal of Computer Applications*, 121(22).

Cheema, F. M., Akram, A., and Iqbal, Z. (2009). Comparative evaluation of header vs. payload based network anomaly detectors. In *Proceedings of the World congress on Engineering*, volume 1, pages 1–5.

Chen, S., Liu, L., Wang, X., Zhang, X., and Zhang, Z. (2014). A host-based approach for unknown fast-spreading worm detection and containment. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):21.

Chen, S. and Ranka, S. (2004). An internet-worm early warning system. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 4, pages 2261–2265. IEEE.

Chen, T. M. and Abu-Nimeh, S. (2011). Lessons from stuxnet. *Computer*, 44(4):91–93.

Chen, Z., Gao, L., and Kwiaty, K. (2003). Modeling the spread of active worms. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1890–1900. IEEE.

Chien, E., OMurchu, L., and Falliere, N. (2012). W32. duqu: the precursor to the next stuxnet. In *Proc. of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*.

Chiueh, S. N. T.-c. and Brook, S. (2005). A survey on virtualization technologies. *RPE Report*, pages 1–42.

Chung, S. P. and Mok, A. K. (2006). Allergy attack against automatic signature generation. In *Recent Advances in Intrusion Detection*, pages 61–80. Springer.

Cisco (2016). *Cisco Catalyst 4500 Series Switches*. [Online]. Accessed 2nd June 2016. Available: http://www.cisco.com/c/en/us/products/switches/catalyst4500seriesswitches/index.html.

Citicaka (2014). *24-Hour Examination: Average Mobile and Desktop Usage Rates*. [Online]. Accessed on 12th November 2014. http://cdn2.hubspot.net/hub/239330/file-30132818-pdf/ChitikaInsights-Hour-By-Hour.pdf.

Comar, P. M., Liu, L., Saha, S., Tan, P.-N., and Nucci, A. (2013). Combining supervised and unsupervised learning for zero-day malware detection. In *INFOCOM, 2013 Proceedings IEEE*, pages 2022–2030. IEEE.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., and Barham, P. (2005). Vigilante: End-to-end containment of internet worms. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 133–147. ACM.

Cotton, M. and Vegoda, L. (2010). Special use ipv4 addresses. Technical report, BCP 153, RFC 5735, January.

Cowie, J., Ogielski, A., Premore, B., and Yuan, Y. (2001). Global routing instabilities during code red ii and nimda worm propagation.

CVE (2012). *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0002.

CVE (2014a). *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

CVE (2014b). *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3704.

CVE (2014c). *Common Vulnerabilities and Exposures*. [Online]. Accessed on 19th October 2014. Avalable: https://cve.mitre.org/.

Davies, J. (2008). Tcp/ip fundamentals for microsoft windows. *Microsoft Corporation*.

Dekens, L. and Renouf, A. (2011). *VMware vSphere powerCLI reference: automating vSphere administration*. John Wiley and Sons.

DLINK (2016). *Layer 2 Gigabit Managed Switch*. [Online]. Accessed 2nd June 2016. Available: http://ftp.dlink.ru/pub/Switch/DGS3000

Dolak, J. C. (2001). Security essentials version 1.2 e the code red worm.

eEye Digital Security (2001). *MS IIS Remote buffer overflow (SYSTEM Level Access).* [Online]. Accessed on 12th May 2016. Avalable: http://www.eeye.com/html/Research/Advisories/AD20010618.html.

eEye Digital Security (2004). *Internet Security Systems PAM ICQ Server Response Processing Vulnerability.* [Online]. Accessed on 12th May 2016. Avalable: http://www.securityfocus.com/archive/1/357916/30/0/threaded.

Elz, R., Bush, R., et al. (1997). Clarifications to the dns specification.

En-Najjary, T. and Urvoy-Keller, G. (2010). A first look at traffic classification in enterprise networks. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 764–768. ACM.

Falliere, N., Murchu, L. O., and Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5.

Fei, S., Zhaowen, L., and Yan, M. (2009). A survey of internet worm propagation models. In *Broadband Network and Multimedia Technology, 2009. IC-BNMT'09. 2nd IEEE International Conference on*, pages 453–457. IEEE.

Fink, G. A., Chappell, B., Turner, T., and O'Donoghue, K. (2002). A metrics-based approach to intrusion detection system evaluation for distributed real-time systems. Technical report, DTIC Document.

Floyd, S. and Paxson, V. (2001). Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403.

Fosnock, C. (2005). Computer worms: past, present, and future. *East Carolina University*, 8.

Ganger, G. R., Economou, G., and Bielski, S. M. (2002). Self-securing network interfaces: What, why and how? Technical report, DTIC Document.

Goldman, D. (2014). Shodan: The scariest search engine on the internet. *Webseite, Stand*, pages 01–21.

Goyal, R., Sharma, S., Bevinakoppa, S., and Watters, P. (2012). Obfuscation of stuxnet and flame malware. *Latest Trends in Applied Informatics and Computing*.

Gu, G., Sharif, M., Qin, X., Dagon, D., Lee, W., and Riley, G. (2004). Worm detection, early warning and response based on local victim information. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 136–145. IEEE.

Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. (2008). Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14.

Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., and Lepreau, J. (2008). Large scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference*, pages 113–128.

Hwang, J., Zeng, S., Wu, F. Y., and Wood, T. (2013). A component based performance comparison of four hypervisors. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium*, pages 269 – 276.

Ishiguro, K., Takada, T., Ohara, Y., Zinin, A., Natapov, G., and Mizutani, A. (2007). Quagga routing suite.

Jamil, N. and Chen, T. M. (2006). Effectiveness of rate control in slowing down worm epidemics. In *GLOBECOM*.

Jeong, J., Park, J., and Kim, H. (2003). Service discovery based on multicast dns in ipv6 mobile ad-hoc networks. In *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, volume 3, pages 1763–1767. IEEE.

Jiang, X., Xu, D., Wang, H., and Spafford, E. (2006). Virtual playgrounds for worm behavior investigation. In *Recent Advances in Intrusion Detection*, pages 1–21. Springer.

Joukov, N. and Chiueh, T.-c. (2003). Internet worms as internet-wide threat. *Experimental Computer Systems Lab, Tech. Rep. TR-143, September*.

Jung, J., Paxson, V., Berger, A. W., and Balakrishnan, H. (2004). Fast portscan detection using sequential hypothesis testing. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 211–225. IEEE.

Jyothsna, V., Prasad, V. R., and Prasad, K. M. (2011). A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35.

Kaur, R. and Singh, M. (2014). Efficient hybrid technique for detecting zero-day polymorphic worms. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 95–100. IEEE.

Khamphakdee, N., Benjamas, N., and Saiyod, S. (2014). Improving intrusion detection system based on snort rules for network probe attack detection. In *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pages 69–74. IEEE.

Kim, S.-i. and Nwanze, N. (2008). Noise-resistant payload anomaly detection for network intrusion detection systems. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 517–523. IEEE.

Kim, S.-i., Nwanze, N., Edmonds, W., Johnson, B., and Field, P. (2012). On network intrusion detection for deployment in the wild. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 253–260. IEEE.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. (2007). kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230.

Kumar, A., Kamtam, A., and Patkar, U. (2016). Self-defending approach of a network.

Kumar, V. and Sangwan, O. P. (2012). Signature based intrusion detection system using snort. *International Journal of Computer Applications and Information Technology*, 1(3):35–41.

Li, J. and Shad, S. (2014). Detecting smart, self-propagating internet worms. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 193–201. IEEE.

Li, P., Salour, M., and Su, X. (2008). A survey of internet worm detection and containment. *Communications Surveys and Tutorials, IEEE*, 10(1):20–35.

Liang, Z. and Sekar, R. (2005). Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 213–222. ACM.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Lippmann, R., Fried, D., Piwowarski, K., and Streilein, W. (2003). Passive operating system identification from tcp/ip packet headers. In *Workshop on Data Mining for Computer Security*, page 40. Citeseer.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., and Das, K. (2000). The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595.

Lowe, S. (2011). *Mastering VMware vSphere 5*. John Wiley and Sons.

MacDonald, D. and Barkley, W. (2000). Microsoft windows 2000 tcp/ip implementation details. *white paper, http://secinf.net/info/nt/2000ip/tcpipimp.html*.

Mackie, A., Roculan, J., Russel, R., and Velzen, M. (2001). Nimda worm analysis. *Incident Analysis Report, Version*, 2.

Mahoney, M. V. and Chan, P. K. (2001). Phad: Packet header anomaly detection for identifying hostile network traffic.

Malkin, G. S. (1998). Rip version 2.

Martin, O. (2003). Taking smb lure to the next level. *IBM Global Services, UK*.

Mell, P., Hu, V., Lippmann, R., Haines, J., and Zissman, M. (2003). An overview of issues in testing intrusion detection systems.

Mirkovic, J., Benzel, T. V., Faber, T., Braden, R., Wroclawski, J. T., and Schwab, S. (2010). The deter project.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., and Weaver, N. (2003a). Inside the slammer worm. *IEEE Security and Privacy*, (4):33–39.

Moore, D., Shannon, C., Voelker, G. M., and Savage, S. (2003b). Internet quarantine: Requirements for containing self-propagating code. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1901–1910. IEEE.

Murillo, A. and Duarte, O. C. M. B. (2013). Virtual networks for cyber security testing of future internet proposals.

Nazario, J., Ptacek, T., and Song, D. (2004). Wormability: A description for vulnerabilities. *Arbor Networks (October 2004)*.

Newsome, J., Karp, B., and Song, D. (2006). Paragraph: Thwarting signature learning by training maliciously. In *Recent advances in intrusion detection*, pages 81–105. Springer.

NVD (2014). *National Vulnerability Database*. http://nvd.nist.gov/.

Paxson, V. (2005). Lbnl/icsi enterprise tracing project.

Perdisci, R., Dagon, D., Lee, W., Fogla, P., and Sharif, M. (2006). Misleading worm signature generators using deliberate noise injection. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE.

Perera, G., Miller, N., Mela, J., McGarry, M. P., and Acosta, J. C. (2013). Emulating internet topology snapshots in deterlab. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 165–168. ACM.

Perumalla, K. S. and Sundaragopalan, S. (2004). High-fidelity modeling of computer network worms. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 126–135. IEEE.

Punithan, X. J., Kim, J.-D., Kim, D., and Choi, Y.-H. (2016). A game theoretic model for dynamic configuration of large-scale intrusion detection signatures. *Multimedia Tools and Applications*, 75(23):15461–15477.

Qin, X., Dagon, D., Gu, G., and Lee, W. (2004). Worm detection using local networks.

Rasheed, M. M., Norwawi, N. M., Ghazali, O., and Kadhum, M. M. (2009). Intelligent failure connection algorithm for detecting internet worms. *IJCSNS*, 9(5):280.

Riley, G. F. (2003). The georgia tech network simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12. ACM.

Sahoo, J., Mohapatra, S., and Lath, R. (2010). Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE.

SANS (2008). *The Conficker Worm*. Available: http://www.sans.org/security-resources/malwarefaq/conficker-worm.php.

Shahzad, K. and Woodhead, S. (2014a). A pseudo-worm daemon (pwd) for empirical analysis of zero-day network worms and countermeasure testing. In *Computing, Com-*

munication and Networking Technologies (ICCCNT), 2014 International Conference on, pages 1–6. IEEE.

Shahzad, K. and Woodhead, S. (2014b). Towards automated distributed containment of zero-day network worms. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–7. IEEE.

Shahzad, K., Woodhead, S., and Bakalis, P. (2013). A virtualized network testbed for zero-day worm analysis and countermeasure testing. In *Advances in Security of Information and Communication Networks*, pages 54–64. Springer.

Shannon, C. and Moore, D. (2004). The spread of the witty worm. *Security and Privacy, IEEE*, 2(4):46–50.

Sharma, A. and Sharma, M. (2015). Analysis and implementation of bro ids using signature script. In *Soft Computing Techniques and Implementations (ICSCTI), 2015 International Conference on*, pages 57–60. IEEE.

Sherman (2014). *Common considerations when selecting your hypervisor*. [Online]. Accessed 19th October 2014 Available: http://www.sudops.com/wpcontent/uploads/2014/03/.

Smith, C., Matrawy, A., Chow, S., and Abdelaziz, B. (2009). Computer worms: Architectures, evasion strategies, and detection mechanisms. *Journal of Information Assurance and Security*, 4:69–83.

Soltani, S., Seno, S. A. H., Nezhadkamali, M., and Budiarto, R. (2014). A survey on real world botnets and detection mechanisms. *International Journal of Information and Network Security*, 3(2):116.

Srivastava, A. and Giffin, J. (2010). Automatic discovery of parasitic malware. In *Recent Advances in Intrusion Detection*, pages 97–117. Springer.

Stafford, S. and Li, J. (2010). Behavior-based worm detectors compared. In *Recent Advances in Intrusion Detection*, pages 38–57. Springer.

Staniford, S., Paxson, V., Weaver, N., et al. (2002). How to own the internet in your spare time. In *USENIX Security Symposium*, pages 149–167.

Sun, W., Katta, V., Krishna, K., and Sekar, R. (2008). V-netlab: An approach for realizing logically isolated networks for security experiments. *CSET*, 8:1–6.

Symantec, C. (2015). *Vulnerabilities*. [Online]. Accessed on 12th November 2014. Available: http://www.securityfocus.com/.

Tidy, L., Shahzad, K., Aminu, A. M., and Steve, W. (2014). An assessment of the contemporary threat posed by network worm malware. In *The Ninth Internation Conference on Systems and Networks Communications (ICSNC 2014)*.

Tidy, L., Woodhead, S., and Wetherall, J. (2015). Simulation of zero-day worm epidemiology in the dynamic, heterogeneous internet. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 12(2):123–138.

TPLINK (2016). *TPLINK Managed Switches*. [Online]. Accessed 2nd June 2016. Available: http://www.tplink.com/en/products/bizlist39.html.

Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., and Song, D. (2007). Sweeper: A lightweight end-to-end system for defending against fast worms. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 115–128. ACM.

Turner, A. and Bing, M. (2005). Tcpreplay: Pcap editing and replay tools for* nix. *online], http://tcpreplay. sourceforge. net*.

USMA (2016). *CDX Data*. [Online]. Accessed on 20th October 2015. http://www.usma.edu/acc/SitePages/CDX.aspx.

van der Vorst, T., Brennenraedts, R., van Kerkhof, D., and Bekkers, R. (2014). Fast forward: How the speed of the internet will develop between now and 2020.

van Heerden, R., Pieterse, H., Burke, I., and Irwin, B. (2013). Developing a virtualised testbed environment in preparation for testing of network based attacks. In *Adaptive Science and Technology (ICAST), 2013 International Conference on*, pages 1–8. IEEE.

Velte, A. and Velte, T. (2009). *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc.

WAND (2001). *WAND WITS: Auckland-IV trace data*. [Online]. Accessed 19th October 2014. Available: http://wand.net.nz/.

Wang, K. and Stolfo, S. J. (2004). Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer.

Wang, Y., Wen, S., Xiang, Y., and Zhou, W. (2014). Modeling the propagation of worms in networks: A survey. *Communications Surveys and Tutorials, IEEE*, 16(2):942–960.

Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. (2003). A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 11–18. ACM.

Weaver, N., Staniford, S., and Paxson, V. (2004). Very fast containment of scanning worms. In *USENIX Security Symposium*, volume 2, pages 16–85.

Wei, S., Mirkovic, J., and Swany, M. (2005). Distributed worm simulation with a realistic internet model. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 71–79. IEEE Computer Society.

White, J. and Pilbeam, A. (2010). A survey of virtualization technologies with performance testing. *arXiv preprint arXiv:1010.3233*.

Whyte, D., Kranakis, E., and van Oorschot, P. C. (2005). Dns-based detection of scanning worms in an enterprise network. In *NDSS*.

Williamson, M. M. (2002). Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 61–68. IEEE.

Wong, C., Bielski, S., Studer, A., and Wang, C. (2006). Empirical analysis of rate limiting mechanisms. In *Recent Advances in Intrusion Detection*, pages 22–42. Springer.

Yang, L.-X., Yang, X., Zhu, Q., and Wen, L. (2013). A computer virus model with graded cure rates. *Nonlinear Analysis: Real World Applications*, 14(1):414–422.

Yoon, S. and Kim, Y. B. (2009). A design of network simulation environment using ssfnet. In *In Advances in System Simulation, SIMUL'09. First International Conference on IEEE*, pages 73 – 78.

Zhuang, Y., Cappos, J., Rappaport, T., and McGeer, R. (2013). Future internet bandwidth trends: An investigation on current and future disruptive technologies. Technical report, Technical Report TR-CSE-2013-0411/01/2013, Polytechnic Institute of NYU, Department of Computer Science and Engineering, WEB: http://www. cs. ubc. ca/˜ yyzh/tr-cse-2013-04. pdf (available: 22 February 2014).

Zou, C. C., Gong, W., and Towsley, D. (2002). Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 138–147. ACM.

Zou, C. C., Towsley, D., and Gong, W. (2006). On the performance of internet worm scanning strategies. *Performance Evaluation*, 63(7):700–723.

PUBLICATIONS

This thesis comprises contents that have been coauthored in the following publications;

1. **Muhammad Aminu Ahmad** and Steve Woodhead and Diane Gan, "**A Safeguard Against Fast Self-propagating Malware.**" *The 6th International Conference on Communication and Network Security (ICCNS 2016)*, November 26-29, 2016, Singapore

2. **Muhammad Aminu Ahmad** and Steve Woodhead and Diane Gan, **"Early Containment of Fast Network Worm Malware.**" *The NAFOSTED Conference on Information and Computer Science (NICS) (NICS 2016),* September 14-16, 2016, Danang, Vietnam.

3. **Muhammad Aminu Ahmad** and Steve Woodhead and Diane Gan, **"A Countermeasure Mechanism for Fast Scanning Malware".** *International Conference On Cyber Incident Response, Coordination, Containment and Control (Cyber Incident 2016).* June 13-14, 2016, London, UK.

4. **Muhammad Aminu Ahmad** and Steve Woodhead and Diane Gan. **"The V-Network Testbed for Malware Analysis."** *International Conference on Advanced Communication Control and Computing Technologies (ICACCCT' 16).* May 25-26, India.

5. **Muhammad Aminu Ahmad** and Steve Woodhead, **"Containment of Fast Scanning Computer Network Worms."** *International Conference on Internet and Distributed Computing Systems.* Springer International Publishing, 2015.

6. Tidy, Luc, Shahzad, Khurram, **Muhammad Aminu Ahmad** and Woodhead, Steve, **"An assessment of the contemporary threat posed by network worm malware."** *The Ninth International Conference on Systems and Networks Communications (ICSNC 2014).* October 2014, Nice, France.

Appendix A

NEDAC Pseudocode

```
1: Begin
2: /* Initialize Caches */
3: initializeCache(IOCache)
4: initializeCache(resolutionCache)
5: initializeCache(noResolutionCache)
6: initializeTable(exemptTable)
7: /* Initialize Thresholds */
8: Threshold1 = setThresholdARP()
9: Threshold2 = setThresholdIP()
10: /* Set timer */
11: T = SetTimerSignal()
12: /* open interface */
13: openInterface(interface)
14: /* do in parallel */ process 1
15: while (there are datagrams to process) do
16:     getARPandIPDatagram()
17:     uniqueMAC = getMACAddress(datagram)
18:     headerInfo = getHeaderInfo()
```

```
19:  if datagram is ARP request to unused IP address then
20:      dportcounter = updateHostCount()
21:      if determineThreshold(dportcounter) >= Threshold1 then
22:          containHost(MACAddress)
23:      end if
24:  end if
25:  if datagram is inbound IP datagram then
26:      if datagram is DNS reply then
27:          updateResolutionCache()
28:      else
29:          updateIOCache(dstIP)
30:          updateIOCache(dPort)
31:      end if
32:  else
33:      if headerInfo is not found in exemptTable[] then
34:          if source host not a server then
35:              if IP addresses not in resolutionCache then
36:                  updateNoResolutionCache(headerInfo)
37:                  if determineThreshold(dportcounter) >= Threshold2 then
38:                      containHost(MACAddress)
39:                      if dPort in IOCache then
40:                          blockInbound(dPort)
41:                      end if
42:                  end if
43:              end if
44:          else
45:              correlate(headerInfo)
46:              if dport in IOCache then
47:                  updateIOCache(headerInfo)
48:                  if determineThreshold(dportcounter) >= Threshold2 then
```

```
49:                         containHost(MACAddress)
50:                         if dPort in IOCache then
51:                             blockInbound(dPort)
52:                         end if
53:                     end if
54:                 end if
55:             end if
56:         end if
57:     end if
58:     getNextDatagram()
59: end while
60: /* do in parallel */ process 2
61: while true do
62:     if (T generates timeout signal) then
63:         for entries in resolutionCache do
64:             if (TTL >= 86400) then
65:                 deleteEntry()
66:             end if
67:         end for
68:         for entries in noResolutionTable and inboundTable do
69:             if (TTL >= 60) then
70:                 deleteEntry()
71:             end if
72:             halveThresholds()
73:         end for
74:     end if
75: end while
76: End
```

Appendix B

False Positives

The false positives raised by the DSC, DNS-RL and NEDAC during the sets of experiments conducted with the DARPA, CDX and LAB datasets are presented in Tables B.1 through B.6. The Port/Host column indicates the number of hosts that caused the false positives on a particular port. For example, during the DSC experiment with CDX dataset using a threshold value of 100, 12 hosts raised a false positive on TCP port 12.

Table B.1: False positives raised with CDX dataset x10 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	82	TCP 21/12, TCP 445/17	UDP 53/19, UDP 123/18 UDP 138/16
	200	31	TCP 21/7, TCP 445/11	UDP 53/10, UDP 123/1, UDP 138/3
DNS-RL	100	107	TCP 21/9, TCP 25/9, TCP 80/13, TCP 443/11, TCP 445/8	UDP 67/10, UDP 68/2, UDP 123/13, UDP 137/18, UDP 138/14
	200	45	TCP 80/2, TCP 443/5	UDP 67/12, UDP 68/1, UDP 123/7, UDP 137/10, UDP 138/8
	300	11	UDP 67/5, UDP 123/1, UDP 137/3, UDP 138/2	
NEDAC	100	19	UDP 67/5, UDP 68/1, UDP 137/7, UDP 138/6	
	200	4	UDP 137/3, UDP 138/1	

Table B.2: False positives raised with CDX dataset x15 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	53	TCP 21/8, TCP 445/9	UDP 53/17, UDP 123/12, UDP 138/7
DNS-RL	100	58	TCP 21/2, TCP 80/7, TCP 443/9	UDP 67/14, UDP 68/1, UDP 123/6, UDP 137/11, UDP 138/9
	200	9	TCP 443/1	UDP 67/2, UDP 137/4, UDP 138/2

Table B.3: False positives raised with DARPA dataset x10 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	162	TCP 1135/18, TCP 1140/19, TCP 1220/16, TCP 1241/14, TCP 1362/12, TCP 2060/11, TCP 2254/7, TCP 3973/16	UDP 53/28, UDP 123/19, UDP 520/2
	200	97	TCP 1135/12, TCP 1140/15, TCP 1220/14, TCP 1241/13, TCP 1362/12, TCP 2060/8,	UDP 53/24, UDP 520/2, UDP 123/3
	300	46	TCP 1220/6, 1241/5, TCP 1135/7, TCP 1362/7	UDP 53/19, UDP 520/2
	400	11	TCP 1135/3	UDP 53/6, UDP 520/2
DNS-RL	100	194	TCP 23/20, TCP 1135/9, TCP 1168/9, TCP 1220/19, TCP 1241/14, TCP 1362/13, TCP 2060/11, TCP 2190/12, 2254/7, TCP 3846/7, TCP 3973/8, TCP 4172/8, TCP 4189/7, TCP 4686/6, TCP 5236/4, TCP 5862/10, TCP 6667/4, 7836/5	UDP 123/4, UDP 137/10, UDP 138/5, UDP 520/2
	200	133	TCP 23/17, TCP 1135/7, TCP 1168/8, TCP 1220/19, TCP 1241/14, TCP 1362/13, TCP 2060/10, TCP 2190/12, TCP 7836/5, TCP 5236/4, TCP 5862/9	UDP 520/2, UDP 137/8, UDP 138/5
	300	89	TCP 23/8, TCP 1135/7, TCP 1168/8, TCP 1220/15, TCP 1241/6, TCP 1362/9, TCP 2060/9, TCP 2190/7, TCP 7836/4, TCP 5236/3	UDP 520/2, UDP 137/6, UDP 138/5
	400	28	TCP 23/9, TCP 1220/8, TCP 1168/5, TCP 7836/2, TCP 5236/2	UDP 520/2
NEDAC	100	31	-	UDP 137/16, UDP 138/13, UDP 520/2
	200	13	-	UDP 137/7, UDP 138/4, UDP 520/2
	300	12	-	UDP 137/6, UDP 138/4, UDP 520/2

Table B.4: False positives raised with DARPA dataset x15 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	99	TCP 1135/10, TCP 1140/3, TCP 1220/11, TCP 1241/10, TCP 1362/9, TCP 2060/11, TCP 2190/7, TCP 2254/1, TCP 3846/4, TCP 3973/2, TCP 4154/2, TCP 4686/2, TCP 7716/2, TCP 7836/4, TCP 9681/4	UDP 53/15, UDP 520/2, UDP 123/2
	200	41	TCP 1135/5, TCP 1140/1, TCP 1220/5, TCP 1241/5, TCP 1362/5, TCP 2060/4, TCP 2190/2, TCP 2254/1, TCP 7836/3	UDP 53/8, UDP 520/2
	300	7	TCP 23/2, TCP 1220/1, TCP 1135/1, TCP 1362/1, TCP 1241/1, TCP 2060/1	
DNS-RL	100	140	TCP 23/12, TCP 1135/9, TCP 1168/7, TCP 1220/12, TCP 1241/10, TCP 1362/11, TCP 2060/9, TCP 2190/8, TCP 2254/6, TCP 3846/6, TCP 3973/5, TCP 4172/7, TCP 4189/5, TCP 4686/4, TCP 5236/4, TCP 5862/2, TCP 7836/5	UDP 137/10, UDP 138/5, UDP 123/2, UDP 520/2
	200	96	TCP 23/8, TCP 1135/6, TCP 1168/6, TCP 1220/8, TCP 1241/9, TCP 1362/7, TCP 2060/8, TCP 2190/5, TCP 2254/4, TCP 7716/4, TCP 5862/1, TCP 3846/6, TCP 3973/5, TCP 4172/4, TCP 4686/3, TCP 4189/5	UDP 137/7, UDP 138/4, UDP 123/2, UDP 520/2
	300	64	TCP 23/9, TCP 1135/3, TCP 1168/3, TCP 1220/5, TCP 1241/2, TCP 1362/4, TCP 2060/3, TCP 2190/2, TCP 2254/4, TCP 7716/3, TCP 7836/2, TCP 3846/6, TCP 3973/5, TCP 4172/4, TCP 4686/3, TCP 4189/5, TCP 7836/1	
	400	2	TCP 23/2	
NEDAC	100	16		UDP 137/9, UDP 138/5, UDP 520/2

Table B.5: False positives raised with LAB dataset x10 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	27	TCP 445/10	UDP 53/4, UDP 123/8, UDP 138/5
	200	8	TCP 445/6	UDP 123/2
DNS-RL	100	51	TCP 80/5, TCP 443/16, TCP 445/7, TCP 1935/3	UDP 123/2, UDP 137/5, UDP 138/7, UDP 5355/6
	200	30	TCP 443/12, TCP 1935/1, TCP 445/4, TCP 80/1	UDP 123/1, UDP 137/2, UDP 138/4, UDP 5355/5
	300	3	TCP 443/1, TCP 445/2	-

Table B.6: False positives raised with LAB dataset x15 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	15	TCP 445/8	UDP 53/2, UDP 123/2, UDP 138/3
DNS-RL	100	35	TCP 80/3, TCP 443/11, TCP 445/4, TCP 1935/1	UDP 123/3, UDP 137/2, UDP 138/6, UDP 5355/5
	200	19	TCP 443/8, TCP 445/4	UDP 137/1, UDP 138/3, UDP 5355/3

Appendix C

True False Positives

The true false positives raised by the DSC and DNS-RL schemes during the sets of experiments conducted with the DARPA, CDX and LAB datasets are presented in Tables C.1 through C.6. These are false positives raised by the schemes for legitimate and routable datagrams transmitted by hosts.

Table C.1: True false positives raised with CDX dataset x10 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	66	TCP 21/12, TCP 445/17	UDP 53/19, UDP 123/18
	200	28	TCP 21/7, TCP 445/11	UDP 53/10, UDP 123/1
DNS-RL	100	63	TCP 21/9, TCP 25/9, TCP 80/13, TCP 443/11, TCP 445/8	UDP 123/13
	200	14	TCP 80/2, TCP 443/5	UDP 123/7
	300	1	UDP 123/1	

Table C.2: True false positives raised with CDX dataset x15 seconds

Scheme	**Threshold**	**Total False Positives**	**TCP Port/Hosts**	**UDP Port/Hosts**
DSC	100	46	TCP 21/8, TCP 445/9	UDP 53/17, UDP 123/12
DNS-RL	100	23	TCP 21/2, TCP 80/7, TCP 443/9	UDP 123/6
	200	1	TCP 443/1	-

Table C.3: True false positives raised with DARPA dataset x10 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	160	TCP 1135/18, TCP 1140/19, TCP 1220/16, TCP 1241/14, TCP 1362/12, TCP 2060/11, TCP 2254/7, TCP 3973/16	UDP 53/28, UDP 123/19
	200	95	TCP 1135/12, TCP 1140/15, TCP 1220/14, TCP 1241/13, TCP 1362/12, TCP 2060/8,	UDP 53/24, UDP 123/3
	300	44	TCP 1220/6, 1241/5, TCP 1135/7, TCP 1362/7	UDP 53/19
	400	9	TCP 1135/3	UDP 53/6
DNS-RL	100	177	TCP 23/20, TCP 1135/9, TCP 1168/9, TCP 1220/19, TCP 1241/14, TCP 1362/13, TCP 2060/11, TCP 2190/12, 2254/7, TCP 3846/7, TCP 3973/8, TCP 4172/8, TCP 4189/7, TCP 4686/6, TCP 5236/4, TCP 5862/10, TCP 6667/4, 7836/5	UDP 123/4
	200	118	TCP 23/17, TCP 1135/7, TCP 1168/8, TCP 1220/19, TCP 1241/14, TCP 1362/13, TCP 2060/10, TCP 2190/12, TCP 7836/5, TCP 5236/4, TCP 5862/9	-
	300	76	TCP 23/8, TCP 1135/7, TCP 1168/8, TCP 1220/15, TCP 1241/6, TCP 1362/9, TCP 2060/9, TCP 2190/7, TCP 7836/4, TCP 5236/3	-
	400	26	TCP 23/9, TCP 1220/8, TCP 1168/5, TCP 7836/2, TCP 5236/2	-

Table C.4: True false positives raised with DARPA dataset x15 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	97	TCP 1135/10, TCP 1140/3, TCP 1220/11, TCP 1241/10, TCP 1362/9, TCP 2060/11, TCP 2190/7, TCP 2254/1, TCP 3846/4, TCP 3973/2, TCP 4154/2, TCP 4686/2, TCP 7716/2, TCP 7836/4, TCP 9681/4	UDP 53/15, UDP 123/2
	200	39	TCP 1135/5, TCP 1140/1, TCP 1220/5, TCP 1241/5, TCP 1362/5, TCP 2060/4, TCP 2190/2, TCP 2254/1, TCP 7836/3	UDP 53/8
	300	7	TCP 23/2, TCP 1220/1, TCP 1135/1, TCP 1362/1, TCP 1241/1, TCP 2060/1	
DNS-RL	100	123	TCP 23/12, TCP 1135/9, TCP 1168/7, TCP 1220/12, TCP 1241/10, TCP 1362/11, TCP 2060/9, TCP 2190/8, TCP 2254/6, TCP 3846/6, TCP 3973/5, TCP 4172/7, TCP 4189/5, TCP 4686/4, TCP 5236/4, TCP 5862/2, TCP 7836/5	UDP 123/2
	200	83	TCP 23/8, TCP 1135/6, TCP 1168/6, TCP 1220/8, TCP 1241/9, TCP 1362/7, TCP 2060/8, TCP 2190/5, TCP 2254/4, TCP 7716/4, TCP 5862/1, TCP 3846/6, TCP 3973/5, TCP 4172/4, TCP 4686/3, TCP 4189/5	UDP 123/2
	300	64	TCP 23/9, TCP 1135/3, TCP 1168/3, TCP 1220/5, TCP 1241/2, TCP 1362/4, TCP 2060/3, TCP 2190/2, TCP 2254/4, TCP 7716/3, TCP 7836/2, TCP 3846/6, TCP 3973/5, TCP 4172/4, TCP 4686/3, TCP 4189/5, TCP 7836/1	
	400	2	TCP 23/2	

Table C.5: True false positives raised with LAB dataset x10 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	22	TCP 445/10	UDP 53/4, UDP 123/8
	200	8	TCP 445/6	UDP 123/2
DNS-RL	100	33	TCP 80/5, TCP 443/16, TCP 445/7, TCP 1935/3	UDP 123/2
	200	19	TCP 443/12, TCP 1935/1, TCP 445/4, TCP 80/1	UDP 123/1
	300	3	TCP 443/1, TCP 445/2	-

Table C.6: False positives raised with LAB dataset x15 seconds

Scheme	Threshold	Total False Positives	TCP Port/Hosts	UDP Port/Hosts
DSC	100	12	TCP 445/8	UDP 53/2, UDP 123/2
DNS-RL	100	22	TCP 80/3, TCP 443/11, TCP 445/4, TCP 1935/1	UDP 123/3
	200	12	TCP 443/8, TCP 445/4	-