Design and Implementation of a Worm Detection and Mitigation System

Hamad Binsalleeh

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

February 2008

Your file Votre référence
ISBN: 978-0-494-40903-9
Our file Notre référence
ISBN: 978-0-494-40903-9

Abstract

Design and Implementation of a Worm Detection and Mitigation System

Hamad Binsalleeh

Internet worms are self-replicating malware programs that use the Internet to replicate themselves and propagate to other vulnerable nodes without any user intervention. In addition to consuming the valuable network bandwidth, worms may also cause other harms to the infected nodes and networks. Currently, the economic damage of Internet worms' attacks has reached a level that made early detection and mitigation of Internet worms a top priority for security professionals within enterprise networks and service providers.

While the majority of legitimate Internet services rely on the Domain Name System (DNS) to provide the translation between the alphanumeric human memorizable host names and their corresponding IP addresses, scanning worms typically use numeric IP addresses to reach their target victims instead of domain names and hence eliminate the need for DNS queries before new connections are established by the worms. Similarly, modern mass-mailing worms employ their own SMTP engine to bypass local mail servers security measures. However, they still rely on the DNS servers for locating the respective mail servers of their intended victims. Creating host-based Mail eXchange (MX) requests is a violation of the typical communication pattern because these requests are supposed to only take place between mail servers and DNS servers. Several researchers have noted that the correlation of DNS queries with outgoing connections from the network can be utilized for

the detection zero-day scanning worms and mass-mailing worms.

In this work, we implement an integrated system for the detection and mitigation of zero-day scanning and mass-mailing worms. The detection engine of our system utilizes the above mentioned DNS anomalies of the worm traffic. Once a worm is detected, the firewall rules are automatically updated in order to isolate the infected host. An automatic alert is also sent to the user of the infected host. The system can be configured such that the user response to this alert is used to undo the firewall updates and hence helps reduce the interruption of service resulting from false alarms.

The developed system has been tested with real worms in a controlled network environment. The obtained experimental results confirm the soundness and effectiveness of the developed system.

Acknowledgement

I would like to express my deepest sense of gratitude to my supervisor, Dr. Amr Youssef, for his patient guidance, encouragement and excellent advice throughout this study.

I also would like to express my gratitude to my scholarship sponsor from Saudi Arabia: Imam Mohammed Bin Saud Islamic University.

I am thankful to Dr Lingyu Wang for his valuable suggestions, to my lab-mates: Esam Elsheh, Mohamed Raslan, Saad Inshi, Yazan Elhamwi and Najah Aboazom for helping me in collecting the experimental data during my research.

Special thanks also go to Farkhund Iqbal and Mohammed Ouhsain for their assistance in editing this thesis and giving me valuable feedback.

Finally, I take this opportunity to express my profound gratitude to my beloved parents, my wife and my little daughter for their moral support and patience during my studies at Concordia University.

Table of Contents

List of Tables

List of Figures

List of Acronyms

API	Application Programming Interface
ARP	Address Resolution Protocol
ASP	Active Server Pages
BSD	Berkeley Software Distribution
CBCRL	Credit-based Connection Rate Limiting
DBMS	DataBase Management System
DCU	DNS Correlation Unit
DDoS	Distributed Denial of Service attack
DNS	Domain Name System
DoS	Denial of Service attack
DSL	Digital Subscriber Loop
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GUI	Graphical User Interface

HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICSI	International Computer Science Institute
IDS	Intrusion Detection System
IIS	Internet Information Services
IM	Instant Message
IP	Internet Protocol
IRC	Internet Relay Chat
LAN	Local Area Network
LSASS	Local Security Authority Subsystem Service
MSN	Microsoft Network
MX	Mail eXchange record
NIDS	Network Intrusion Detection System
NSM	Network Security Monitoring
NTP	Network Time Protocol
PHP	Hypertext Preprocessor
PPU	Packet Processing Unit
RPC	Remote Procedure Call
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language

SSH	Secure Shell
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VoIP	Voice over IP

CHAPTER 1

Introduction

An Internet worm is a self-propagating malware program that automatically replicates itself to vulnerable systems and spreads across the Internet. Worms, which do not require human intervention, often called self-spreading worms, can propagate much faster than those, which are dependent on some user interaction. These worms propagate themselves by exploiting a security vulnerability in certain versions of service software to take control of the victim machine and copy themselves over to other vulnerable machines.

Unlike computer viruses that typically spread from one computer to another by attaching themselves to either data files or executable applications (and hence their spread is limited by the speed by which these infected files can be transmitted from one system to another), worms, in contrast, are capable of autonomous migration from one system to another via the network without the assistance of external software. Typically, a worm-infected host scans the Internet for vulnerable systems. It chooses an IP address, attempts a connection to a service port, and if successful, lunches the attack. The above process repeats with different random addresses. The more machines are compromised, the more copies of the worm that can work together to reproduce themselves. Since the hosts that are vulnerable

to a worm typically account for a small portion of the IP address space, worms rely on high-volume random scan to find victims. Thus, the scan traffic from tens of thousands of compromised machines can congest networks and an explosive epidemic can therefore be developed across the Internet causing a Denial-of-Service (DoS). Although most known worms did not cause severe damage to the compromised nodes, they could have altered data, removed files, stolen information if they had chosen to do so.

In the next section, we present a brief history of Internet worms.

1.1 History and Examples of Internet Worms

The term worm was casted by John Brunner in 1975 in a scientific fiction novel entitled: *The Shockwave Rider.* Later on, two researchers, J. Shock and J. Hupp, of Xerox PARC chose the name in an ACM paper published in 1982 [1].

The first implementation of a worm was by these same two researchers at Xerox PARC in 1978. Shoch and Hupp originally designed the worm to find idle processors on the network and assign them tasks, sharing the processing load, and so improving the CPU cycle use efficiency across an entire network. These benign worms were self-limited so that they would spread no farther than intended.

In 1988, the Morris worm showed the Internet community for the first time that a worm could bring the Internet down in few hours [2]. Morris originally intended the program to be a benign proof of concept. However, it had a massive effect due to a bug in the code. When it reinfected a machine, there was a fixed chance that the new infection wouldn't quit, causing the number of running worms on a machine to build up, thereby causing a heavy

load on many systems. Even on a modern machine, such bugs would have a similar effect of overwhelming the system. This caused the worm to be quickly noticed and caused significant disruption. Most subsequent worms have mechanisms to prevent this from happening [3].

Since then, new worm outbreaks have occurred periodically even though their mechanism of spreading was long well understood. On July 19th, 2001, the code-red worm (version 2) infected more than 250,000 hosts in just 9 hours [4]. In 2002 another worm targeting the Microsoft server appeared. This worm, dubbed *SQL Snake* targeted poorly configured Microsoft SQL servers [5] [6]. While the worm did not spread very far, it did infect several thousand machines and also demonstrated the advances in worm techniques authors are making. Soon after, the *Nimda* worm raged on the Internet [7]. On January 25th, 2003, a new worm called *SQLSlammer* [8] reportedly shut down networks across Asia, Europe and the Americas, which showed how ill-prepared the current Internet infrastructure.

Internet worm attacks are not only limited to taking advantage of system vulnerabilities, but can also be spread through electronic mail, instant messages, and file sharing systems. On March 1999, an electronic-mail virus, called Melissa, exploited a bug in the email system. Melissa virus first hit the Asia- Pacific region, which includes Hong Kong, Singapore, and Australia, and then spread throughout the globe [9]. On August 19th, 2003, Sobig.F was discovered as an email worm, which spreads as attachment of emails with varying subject, message body and attachment filename [10]. On January 26th, 2004, the *MyDoomemail* worm first appeared. This worm contained its own mail transfer agent application and replicated itself by sending copies of itself as an email attachment [11].

The wide spread of instant messages (IM) systems attracted worm authors to target

these systems. In 2002, a new worm targeted the Microsoft Messenger (MSN) by attracting the victims to a malicious website that uses JavaScript language to gain control over the system.

On March 6th, 2005 another worm called *Kelvir* was discovered. *Kelvir* changes the normal strategy of *IM* worms for propagation and introduces the usage of peer to peer architecture [11]. With the increasing migration toward a network-centric computing model, threats to all computers grow in severity. Table 1.1 shows a summary and classification of some famous Internet worms and some of their characteristics, which will be explained throughout the next chapter.

1.2 Motivation

Currently, the economic damage of Internet worms' attacks has reached a level that made early detection and mitigation of Internet worms a top priority for security professionals within enterprise networks and service providers.

Early attempts of worm detection systems were mainly signature based systems, which look for particular explicit indications of attacks such as the pattern of malicious traffic payload (called attack signatures). The attack signatures have to be manually identified by human experts through careful analysis of the byte sequence from captured attack traffic. A good signature should be one that consistently shows up in attack traffic but rarely appears in normal traffic. Automated signature generation for new attacks is extremely difficult due to several reasons. First, in order to create an attack signature, we must identify and isolate attack traffic from legitimate traffic. Second, the signature generation must be general

Name	Worm Type	Infection Propagator	Size [Byte]	Protocol and Port	Outbreak Date	Infected Hosts
W32.Mancsyn [11]	Vulnerability	Child request	23552	MS-DS 445 TCP	03.23.2007	> 1000
W32.Sagevo [11]	Vulnerability	Child request	Varies	TCP port 2967 and download backdoor on TCP port 21211	12.13.2006	> 1000
W32.Wallz [11]	Vulnerability	Child request	6578	MS-DS 445 TCP	02.7.2005	> 1000
W32.Sasser.B [11]	Vulnerability	Child request	15872	MS-DS 445 FTP 5554 listening on 9996	01.05.2004	500000 - 1000000
W32.Welchia.A [11]	Vulnerability	Child request	10240	RPC DCOM 135 Web-Dav 80	18.08.2003	> 1000
W32.Blaster.A [11]	Vulnerability	Child request	6176	RPC DCOM 135 TFTP 69 listening on 4444	11.08.2003	200000 - 8000000
SQL Slammer Worm [11]	Vulnerability	Direct propagation	376	MS-SQL 1434 (UDP)	26.01.2003	> 75000
Code Red Worm [11]	Vulnerability	Direct propagation	3569	HTTP 80	19.07.2001	> 350000
W32.Nimda.A [11]	Vulnerability Mass-mailing	SMTP engine /Child request	57344	DNS 53 (UDP) SMTP 25, TFTP 69 HTTP 80, MS-DS 445 NET-BIOS 137-139	18.09.2001	> 1000
W32.Mabezat.A [11]	Mass-mailing	SMTP engine	29366	DNS 53 (UDP) SMTP 25	11.12.2007	> 1000
W32.Areses.A [11]	Mass-mailing	SMTP engine	-	DNS 53 (UDP) SMTP 25	01.04.2006	> 1000
W32.Aprilcone.A [11]	Mass-mailing	JMail open source software	585728	DNS 53 SMTP 25	04.07.2005	> 1000
W32.Netsky.D [11]	Mass-mailing	SMTP engine	17424	DNS 53 (UDP) SMTP 25	01.04.2004	> 1000
W32.Mydoom.A [11]	Mass-mailing	SMTP engine	22528	DNS 53 (UDP) SMTP 25 listening 3127-3198	26.01.2004	> 1000
W32.Beagle.A [11]	Mass-mailing	SMTP engine	15872	DNS 53 (UDP) SMTP 25 listening on 6777	19.01.2004	> 1000
W32.Sobig.F [11]	Mass-mailing	SMTP engine	72000	NTP 123 (UDP) UDP 8998	19.08.2003	> 1000
W32.Bugbear [11]	Mass-mailing	SMTP engine	50688	DNS 53 (UDP) SMTP 25 listening on 36794	30.09.2002	> 1000
W32.Gibe [11]	Mass-mailing	SMTP engine	122880	DNS 53 (UDP) SMTP 25	04.05.2002	> 1000
W32.Sircam [11]	Mass-mailing	SMTP engine	134000	DNS 53 (UDP) SMTP 25	17.07.2001	> 1000
W32.Klez.E [11]	Mass-mailing	SMTP engine	60000	DNS 53 (UDP) SMTP 25	17.01.2001	> 1000
W97.Melissa.A [11]	Mass-mailing	MAPI commands to send emails	134000	DNS 53 (UDP) SMTP 25	17.07.2001	> 1000

Table 1.1: A summary of some famous Internet worms [12]

enough to capture all attack traffic of a certain type while at the same time specific enough to avoid overlapping with the content of normal traffic in order to reduce false-positives. This problem has so far been handled in an ad-hoc way based on human judgment. Third, the defense system must be flexible enough to deal with the polymorphism [13, 14] in the attack traffic. Otherwise, worms may be programmed to deliberately modify themselves each time they replicate and thus fool the defense system.

Because of their simplicity, the signature-based systems can operate in real time and can detect all known worms. On the other hand, such systems are usually helpless against zero-day worms for which the attack signature have not been identified at the time of the attack.

Second generation of worm detection systems employ an additional anomaly based component [15–17] that profiles the statistical features of the network traffic. Any deviation from the normal profile will be treated as suspicious. Although these systems can detect previously unknown attacks, they may have high false positives when the normal activities are diverse and unpredictable. Moreover, while the majority of most known worms have very aggressive behaviors because they attempt to infect the Internet in a short period of time, and hence are easier to be detected because their aggressiveness stands out from the background traffic, future worms may be modified to circumvent the rate-based defense systems and purposely slow down the propagation rate in order to compromise a vast number of systems over the long run without being detected [18]. Thus the detection of zero-day worms with very low scanning rates requires fundamentally new ideas.

Whyte *et al.* [19] have noted that while the majority of legitimate Internet services rely

on the Domain Name System (DNS) to provide the translation between the alphanumeric human memorizable host names and their corresponding IP addresses, worms typically use numeric IP addresses to reach their target victims instead of domain names and hence eliminate the need for DNS queries before new connections are established by worms. Hence, the correlation of DNS queries with outgoing connections from the network can be utilized for the detection of zero-day worms even if they have a very low scanning rate [19].

Similarly, modern mass-mailing worms employ their own SMTP engine to bypass local mail servers security measures. However, they still rely on the DNS servers for locating the respective mail servers of their intended victims. Creating host-based Mail eXchange (MX) requests is a violation of the typical communication pattern because these requests are supposed to only take place between mail servers and DNS servers [20].

In this work, we implement an integrated system for the detection and mitigation of zero-day scanning and mass-mailing worms. The detection engine of our system utilizes the above mentioned DNS anomalies of the worm traffic. Once detected, the firewall rules are automatically updated in order to isolate the infected host. An automatic alert is also sent to the user of the infected machine. The system can be configured such that the user response to this alert is used to undo the firewall updates and hence helps reduce the effect of false alarms. The developed system has been tested with real worms in a controlled network environment. The obtained experimental results confirm the soundness and effectiveness of the developed system.

1.3 Thesis Overview

The rest of this thesis is organized as follows:

- In Chapter 2, we review the generic structure of advanced Internet worms and the common strategies they use to acquire new victims. We also discuss the main characteristics of Internet worms and present some worm detection strategies. The basic features of Internet worm, which can be used by detection algorithms are also discussed.

- In Chapter 3, we present the design of the developed worm detection system and discuss some implementation details.

- In Chapter 4, we describe our experimental setup and provide some analysis for our experimental results

- In Chapter 5, conclusions as well as some future research directions are given.

CHAPTER 2

Background of Internet Worms

In this chapter, we review the generic structure of advanced Internet worms and the common strategies they use to acquire new victims. We also discuss the main characteristics of Internet worms and present some worm detection strategies. The basic features of Internet worm, which can be used by detection algorithms are also discussed. For further details about Internet worms, the reader is referred to [3, 21].

2.1 Introduction

Over the past years, security researchers have done very extensive analysis to better understand the behavior of Internet worms. Each Internet worm has its own characteristic and tactic to propagate and infect systems.

In the next section, we give a brief description of the structure of Internet worms. In Section 2.3, we present a classification of common Internet worms. In Section 2.4, we provide some characteristics of Internet worms, which explain the process of finding new victims as well as the worm propagation process. In Section 2.5, we review some of the Internet worm features that are used by worm detection systems for early detection of worm attacks. In

Section 2.6, we give a classification of different worm detection systems. Finally, in Section 2.7, we discuss some of the related detection systems with short description for each one of them.

2.2 Worm Structure

Internet worms are composed of several components, which collaborate together to achieve the goal of the worm designer. Every worm contains two main components: the target locator module and infection propagator module. Some other optional components such as remote control, update interface, life-cycle manager, and payload routines can be used according to the worm function during its life cycle. In what follows, we briefly describe each one of these components.

2.2.1 Target Locator

Any Internet worm has to choose which platform to attack and target its associated vulnerabilities. It also has to decide how to attack the victim machines and how to develop the malicious network of the worm. The target locator component is responsible for spreading the worm to other nodes and networks. Scanning worms find their victims by developing techniques to scan the networks for nodes that might be vulnerable using fingerprinting techniques. Mass-mailing worms search inside victim system for next hop by collecting email addresses and send themselves to these addresses.

2.2.2 Infection Propagator

Transferring the worm from one point to another is the second main component in every Internet worm. Using this component, worms gain entry to remote systems. After identifying the target system, the worm has to follow a specific propagation strategy to spread itself to its next victim. Emails, buffer overflow, string formatting, misconfigured systems, file shares and network scanning are examples of worm propagation methods.

2.2.3 Communications and Control

Advanced worms can communicate with their malicious network by specific protocol to strengthen and control both the spreading and infection mechanisms. These communication channels can benefit the worms by updating the infection strategy during the worm life cycle, retrieving some information from the victim machine, or receiving specific control messages such as starting Distributed Denial of Service (DDoS) against several targets.

2.2.4 Life-Cycle Manager

Depending on the goals of the worm author, worms can operate and propagate in specific periods of time. Many worms exploit the world with different variants in different periods.

2.2.5 Payload

Some worms are designed to achieve specific goals after compromising the system. One of the famous goals is to utilize the compromised system network to cause DDoS for a known target. The payload can also be a simple SMTP engine that is used for propagation purposes in case of mass-mailing worms.

2.2.6 Self-Tracking

This module is typically used by worm writers to gather more information about the damages that have been caused during the worm life cycle. Sometimes, it is also used to track the speed of propagation and number of victims.

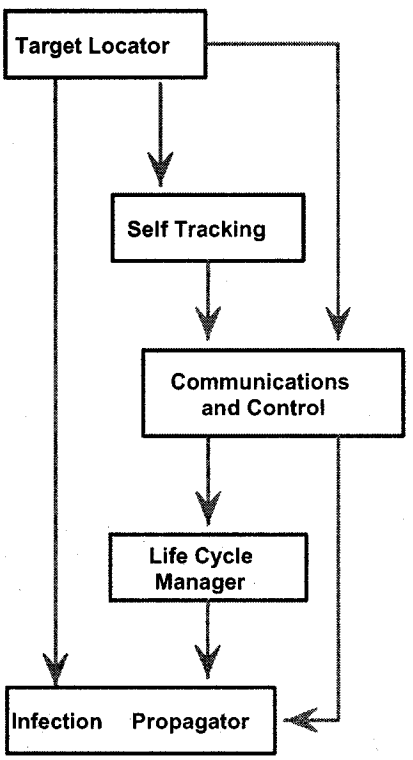

Figure 2.1: Structure of a typical Internet worm [3]

Fig. 2.1 shows the structure of a typical Internet worm. The attack component sends information to the infection propagator component about where to start the attack. It also sends this information to self-tracking and possibly using the communication and control component. This communication component is also used as an interface to the command

component, calling for an attack or the use of the other capabilities against a target.

2.3 Worm Classifications

Internet worms can be classified into various categories depending on different criteria [3,21].

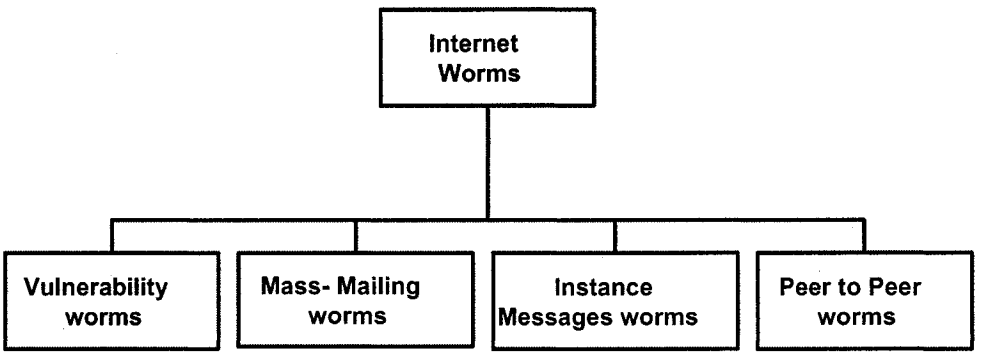

Figure 2.2: Worm classifications

2.3.1 Vulnerability Worms

Vulnerability worms are considered to be the traditional way of Internet worms to exploit the networks. Worms can take advantage of security weaknesses through known vulnerabilities to gain control over vulnerable machines. Vulnerable systems have to be allocated by worms using different scanning mechanisms. In this category, the target locator component relies on the nature of the vulnerability and the scanning technique to choose the victims. After acquiring the target machine for a specific vulnerability, worms start propagating themselves and, at the same time, compromising targeted systems to use them for future attacks.

2.3.2 Mass-mailing Worms

Because of their popularity and daily usage by people and businesses, emails have become perfect carriers for the Internet worms. Mass-mailing worms send themselves to email addresses, which can be collected very easily from different sources in the public Internet. This strategy adds more accuracy in the target locator component by using existing target addresses, which can be found by different mechanisms. In the early stages of these kinds of worms, email client side applications are necessary for the worm propagation through the internet. Nowadays, mass-mailing worms are becoming stronger and more capable to spread without the help of any host programs. In particular, these worms can spread very fast and overcome the security patches of email client side applications by developing a tiny Simple Mail Transfer Protocol (SMTP) engine inside the payload of the worm. After that, the infected machines, communicate with other mail servers in order to send messages to targeted email addresses. Mass-mailing worms may also use social engineering techniques to mislead the victims to open their attachments in order to get infected. Then, they go to the next victims according to their propagation strategy.

2.3.3 Instant Messaging Worms

The wide spread use of Instant Message (IM) networks have attracted the attention of worm authors. IM worms utilized these environments to exploit victims in several ways and worm authors have developed many strategies to get control of such systems. Text messaging, voice chatting and file transferring are attractive applications that can facilitate the propagation of internet worms in a very short period of time. These applications

work with different kinds of protocols depending on the service providers such as Yahoo, Microsoft, and AOL. IM worms use many methods to exploit the systems. They can use some disclosed Application Program Interfaces (APIs) from the vendors of these applications to allow worms gain access to any resources inside these applications including the list of email addresses and send any messages on behalf of the owners of the compromised systems. Furthermore, IM worms can attract the victims by sending URL addresses that link to an existing compromised system, which contains the payload of the worm in order to infect the targeted machine. Moreover, IM worms can attach themselves to any messages that either target victims over the network or infect any files during file transfer.

2.3.4 Peer to Peer Worms

Peer to Peer (P2P) networks are used to share files between network nodes using specific communication protocols depending on the network design. Every node in the P2P network can search and retrieve any file from different sources according to the protocol of that network. P2P worms abuse the flexible P2P network infrastructure to accelerate speed of propagation and infection. Such worms obtain information about target machines directly from a list of nodes inside any sharing network. Within these networks, worms can infect any files inside the shared folders that are accessible by other nodes. P2P worms have become more efficient than any other kinds of worms because of the popularity of P2P networks and their lack of protection. In addition, file exchanging makes this type of network an ideal environment for worm propagation.

2.4 Worm Characteristics

There are several characteristics that define Internet worms. These characteristics are related to worm propagation, targeting victims, and spreading over the internet.

2.4.1 Scanning Mechanisms

The speed of propagation of Internet worms depends on the strength of its scanning algorithm. In this section, we briefly discuss some of these techniques.

Random Scanning

The simplest way to find victims is to use random network scanning. When using this approach, Internet worms start generating all the IP address bytes randomly using pseudo-random number generator algorithms. Then, they probe each IP address to find out whether it is vulnerable or not. After that, they start attacking the remote victim machine by their infection component procedures. Worm authors try to design good random number generators to gain better coverage in the global internet. Otherwise, some networks will not be scanned totally and others will be covered more than once, which may affect the worm propagation speed. Some worm authors resolve this issue by developing a technique to check each target before the infection process takes place to see whether it has been compromised previously or not. Usually, random scanning technique is the main mechanism for worms to allocate new targets with different parameters and different pseudo-random number generator algorithms [22].

Sequential Scanning

Using this technique, the worm starts by a specific IP address and then scans sequentially within specific domain for possible victims. Typically, the worm propagation is slower than other approaches when using the sequential scanning approach and depends on the number of vulnerable systems that can be hit in the early stages of the worm exploit. However, some networks contain reasonable vulnerable systems that can speed up the propagation time. Most worms combine this approach with other scanning strategies to overcome and improve the propagation speed.

Local Scanning

One of the techniques to avoid firewall rules is localized scanning. Most probably, any infected machine has a neighborhood system with the same vulnerability. Moreover, scanning within specific subnet or adjacent networks can increase the propagation speed and the infection process because of communication reliability, and network topology.

Subdividing Scanning

Divide and conquer strategy has been used to design one of the most efficient mechanisms for worm scanning. This approach divides target network space in a hierarchical tree. Each node is responsible for a specific division and part of targeted network. This scanning technique can be combined with other approaches inside any single division, which allows worms to spread very quickly and more efficiently.

Hit-list

One of the most dangerous worm strategies is the hit-list approach. These worms contain a list of information about certain number of vulnerable systems, which are generated before launching the worm. This list is used to initiate the worm network in the first stages to achieve most of the vulnerable systems in a short period of time. After reaching any target in the list, the worm splits the targeted list into two different parts, and so on until it finishes the entire list. At this point, the worm has a very good initial malicious network that can be used to target the rest of the Internet. Then, the worm starts with another scanning strategy to acquire more victims and vulnerable systems. When a worm contains all the vulnerable victims and does not need to look for other victims, we call it a *flash* worm. Worms from this category differ from each other in the way they collect their hit lists and the type of vulnerability they are targeting.

2.4.2 Email Harvesting

Emails have been used to propagate Internet worms in a very efficient way. Mass-mailing worms are not generating any random email addresses as scanning worms, but they collect valid email addresses from different locations by various methods. We will discuss some popular techniques of collecting email addresses, which are used by internet worms so far.

Address-Book

Most of the systems contain some address books to store peoples' contact information along with their emails. When the worm compromises any system, it parses any kind of address

book to collect all the email addresses that are stored on it. Then, it becomes ready to start its propagation and infection process to the new victims by sending itself to those email addresses.

File Parsing

Inside any system, there are usually some files that contain email addresses inside it. Using regular expression techniques, worms can dig inside specific files and look for email addresses to add it to its targeted list. There are some files that have been targeted by many worms. These files come with several extensions (e.g., HTML, PHP, ASP, WAB and TXT).

Web Searches

Another simple way to collect emails is to use public search engines to retrieve emails from the Internet. This technique has proved to be very efficient in collecting a large number of emails from different locations.

SMTP Access and Newsgroup

Some worms wait for the user to send any message, and then they forward themselves to that address. When using this technique, the worm has to be able to deal with SMTP commands in order to extract such emails from any SMTP communication or any other email system protocols.

Combined

Most of the worms use more than one technique at the same time to achieve an efficient number of victim emails.

2.4.3 Payload Propagator

Each worm has different mechanisms to propagate the brain of the worm after compromising the current system. In what follows, we review some of the common worm propagation strategies.

Direct Propagation

The easiest way to send the payload to the compromised machine is to use the infection communication channel, which is previously used for infection propagation. A worm propagator has to setup a signal to initiate the propagation of the payload in the same channel that has already been established at the infection process. Using this strategy, the worm can guarantee that the payload is delivered to a new node in a safe way instead of establishing a new channel, which can be blocked by firewalls or detected by intrusion detection systems. Moreover, this approach does not require any additional overhead on the two sides of communication to initiate another channel for payload propagation.

Child Request Propagation

This technique for payload propagation depends on the new victim. When a new host gets infected with the infection exploit, it tries to communicate back to the node that sends the infection to retrieve the payload of the worm. Using this scenario, the propagator has to

listen on a specific port to receive any signal from the new node, and then it starts the propagation of the payload directly to that machine. From the worm writer's perspective, the only disadvantage of this technique is the possibility of blocking the connection request by the firewall of the child node side.

Central Propagation

The third method of delivering worm payloads is using a third object, which contains the worm payload. Every infected host has to download the payload from a central node that is already configured by the worm author. One of the advantages of this technique is the ability to update and track all the victims in every single moment throughout the worm life cycle. However, this method can be easily detected by intrusion detection systems because of the large number of connections that go to the central node.

2.5 Detection Features of Internet Worms

In this section, we discuss some basic features of Internet worms, which can be used by worm detection algorithms. We only focus on the systems that depend on analyzing the worm network traffic and not on the contents of any worms.

As mentioned earlier in this chapter, Internet worms can be classified based on the service environment and software that are used during worm propagation. For each category, there are some characteristics that can be used for detection purposes. These worm features are used as parameters for designing new algorithms to fight against worm propagation. In this section, we list some of the features of vulnerability and mass-mailing worms.

2.5.1 Detection Features of Vulnerability Worms

The most common features of the vulnerability worms are the traffic that is generated from the infected system. From this feature, we can derive several parameters for detection, depending on the environment and the network design of the detection system. Some features of the known network worms that are used by most of the detection systems as parameters are discussed below.

Connection Degree

The connection degree of a given host is a measure of the number of connection requests generated by this host in a given time frame to different IPs. A normal host does not acquire many new connections in a specific time period to different destinations. On the other hand, an infected host usually sends many connection requests, which indicates the presence of worm activities from that node. Internet worms start communicating with many remote hosts at the same time by several protocols in order to allocate vulnerable machines. In this scenario, we have to keep track of all the connections for each single host and manage the ending sessions for each one of them, in order to determine the number of active connections at any time. When the number of active connections exceed specific limit, then it might be an indication of malicious activities from that host [23].

Failed Connections

One of the main characteristics of worms is that they behave in a different manner than legitimate hosts. Random scanning worms probe many IP addresses, which either belong to the black area of the IP space or even do not support the requested services. Internet

worms try to connect to any remote hosts that are chosen by a target locator component and wait for connection response, connection reset, destination unreachable, or no response at all. Observing a reasonable amount of packets that indicate failed connections (e.g., destination unreachable) can be considered as a sign of worm propagation [24].

Vertical Scanning

Internet worms look for vulnerable systems before attacking. As a result, a worm tries to scan the Internet space for these machines by generating random IP addresses or any different strategy for allocating the victims. These kinds of activities are common in most of the Internet worms. This process takes a long time to allocate victims and happens many times from the infected machine. Finding victims in the Internet space may be referred to as horizontal scanning and checking the services in a specific machine is called vertical scanning. Large number of vertical scans can be considered as a sign of worm propagation.

Sending the Same UDP Packets

When Internet worms use UDP protocol for propagation, packets sent by worms usually have the same size. Worms try to keep the packets so simple and small to speed up the propagation process. Thus observing a large number of similar packets targeting a specific service on many remote hosts is another good sign of worm propagation.

2.5.2 Detection Features of Mass-mailing Worms

Under normal situations, any host sends limited number of emails in a specific time. However, mass-mailing worms try to send as many emails as they can to speed up the propagation and increase the infection damage. Based on this observation, we can detect malicious emails, and consequently mass-mailing worms, by tracking all the emails that have been sent by a specific host in a short interval of time.

2.6 Classification of Worm Detection Systems

Worm detection Systems can be classified in many different ways. The most common classification is based on whether the worm identification decision is based on individual packets or the whole connection [12].

When using packet-oriented systems, the detection algorithm checks every packet and does the attack analysis on the basis of each packet, i.e., packets are not considered as a complete connection session, but individual packets are treated independently instead. The packet matching process can either be a deterministic, in which case arriving packets are compared to packets stored in a database, or statistical, in which case packet flow is statistically analyzed.

For connection-oriented systems, the detection methods interpret packets as part of a connection and base their analysis on the connections as a whole. The number of failed connection attempts as well as the connection rate can be counted and compared to a threshold. Exceeding this threshold indicates an attack.

2.7 Examples of Worm Detection and Mitigation Systems

The Network Security Monitor (NSM) was one of the first detection tools documented in the literature. NMS is based on the connection counter algorithm [25]. The open source network intrusion detection system Snort [26] offers the possibility to use this algorithm too.

The initial proposal that was based on the number of failed connections for the algorithm was published [24]. Several tools have been developed based on this algorithm. Bro is a Unix-based Network Intrusion Detection System [27] developed at Lawrence Berkeley National Laboratory [28] and it is one of the tools, which offer the possibility to use this algorithm. Credit-based Connection Rate Limiting (CBCRL) was developed in cooperation with experts from the MIT and Harvard University in Cambridge [29] and is based on the same algorithm. This method which is based on the observation that infected hosts try to connect with many unreachable hosts. The number of failed connection attempts during a certain time period is counted and compared to a threshold. The CBCRL describes a solution in which each host has its own contingent of available connections credits. A successful connection attempt will extend the host's credits. Whereas, a failed attempt will decrease the number of allowed connections for the concerning host. In addition, it is necessary to prevent a host from acquiring too large number of credits.

The Virus throttling algorithm [30] is based on the connection degree. This algorithm is basically a rate limiting mechanism. It uses specific parameters to restrict the host level contact rates to remote hosts. This algorithm works by keeping a working set of addresses for each host, which models the normal contact behavior of the host. The throttling mechanism

permits outgoing connections for addresses in the working set, but delays other packets by placing them in a delay queue. If the delay queue is full, further packets are simply dropped and an alarm indicating a worm propagation attempt is generated [24]. The same strategy was applied to mass-mailing worms [30] and IM worms when we check the type of messages that are used by IM programs such as text messages, URL links, and file transfer. Another technique was proposed in [31]. It only monitors the URL links and file transfers.

Another approach to mitigate the effect of worms and hackers is the use of honeypots. A honeypot can be defined as a vulnerable network used for several purposes such as distracting attackers, gathering early warnings about new attack techniques, and facilitating in-depth analysis of an adversary's strategies [32]. The idea of using honeypots are to capture and analyze a worm's behavior has been tackled by many researchers [33]. The honeynet project uses a network of high-interaction honeypots over a DSL connection and managed to capture various worms in action. Analyzing network traffic and the honeypot's state have produced detailed descriptions of worms' behavior. Honey pots are also used to slow down worms that employ IP address space scanning by keeping TCP connections open for an indefinite period of time. This approach proved to be useful on delaying the propagation of a worm, but it would be useless in the case of a UDP worm such as Slammer worm. The honeypots are highly-interactive and run real versions of popular applications to protect from malicious attacks. To avoid the compromise of the honeypots, the applications should run on either a sandboxed environment or a high performance virtual machine. They are also monitored for illegal behavior, and when such behavior is detected, the security hole that caused the malicious activity is located and then a patch is automatically

generated and distributed through a software update service. Such active measures cannot be always trusted; an automatically generated patch could harm the system rather than protect it and leaves the opportunity for hackers to exploit the system.

2.8 Example for Real Worms

In what follows, we give a brief description of the three Internet worms that are used in our experiments

2.8.1 Blaster.A

The Blaster worm [11] has first been discovered on August 11th, 2003. It has infected between 200,000 and 8,000,000 hosts, running Windows 2000 or Windows XP operating systems.

Target Locator strategy: The target IP addresses are generated in the following manner; with a probability of 60%, the first three bytes of the address is chosen completely randomly and the fourth byte is set to zero. With 40% probability, it chooses the first two bytes of the local address, the third byte is also taken from the local address, but if it is greater than 20, a random number from 0 to 19 is subtracted from it and the last byte is set to zero. The worm will then increment the last byte of the IP address by one until it reaches 254 and will try to infect all the hosts located at these addresses. *Propagation Mechanism:* During the initialization phase, the worm writes registry entries and creates the IP addresses as described above. Afterward, it tries to set up a TCP connection on port 135 (Windows RPC port). Blaster worm scans blocks of 20 sequential IP addresses by sending a connection

attempt to each one simultaneously [34]. After two seconds, Blaster tries to send the code exploiting the RPC vulnerability to the hosts where a TCP connection successfully could be established. If the code is successfully transmitted and the victim is vulnerable, it causes a command shell to be bounded to port 4444 on the infected target. This shell is used to send commands to the victim machine (e.g. starts a TFTP client). The transmission of the worm code is finally done using TFTP running on UDP port 69.

2.8.2 Sasser.B

Similar to Blaster, the Sasser worm is classified as a vulnerability worm, which targets systems for specific common security vulnerabilities. The Sasser worm has been discovered on May 1st, 2004. It exploits the Local Security Authority Subsystem Service (LSASS) vulnerability, which offers the possibility to execute arbitrary code on the target host over the Internet. *Target Locator strategy:* To choose the next IP addresses, which are targeted for worm infection, the worm retrieves the local IP address of the compromised host and ignores a prespecified list of addresses [11]. The local IP address is then used as the base of selecting all the new victim IP addresses. With a probability of 25%, the last two octets of the IP address are changed to random numbers. With a probability of 23%, the last three octets of the IP address are changed to random numbers and with a probability of 52%, the IP address is completely random. *Propagation Mechanism:* The worm first connects to the chosen IP addresses on TCP port 445 to check if the remote computer is online. If a connection can be established, it sends a sequence of Server Message Block (SMB) packets in order to retrieve the host's SMB banner, which gives a hint of the Windows system version [35]. If the worm could establish this TCP connection, it sends shell codes

to the target machine, which may cause it to open a remote shell on TCP port 9996. As a result, the victim opens an FTP connection to the attacking machine on TCP port 5554 to download the worm. The worm tries to infect 128 hosts in parallel, which results in a heavy decrease of the performance of the infected hosts.

2.8.3 Netsky.D

Netsky is a mass-mailing worm, which has been observed in over 30 variants. The variant D was first discovered on April 1st, 2004 and is the first variant of Netsky, which has infected more than 1000 hosts. This worm uses its own SMTP engine to propagate and uses different subjects, bodies, and attachment names. The attachment always has a .pif file extension.

Target Locator strategy: The worm sends itself to email recipients and therefore does not need any IP addresses. The worm scans files with certain file extensions on drives C to Z to collect email addresses. *Propagation Mechanism:* Netsky.D sends itself (using its own SMTP engine) to each email address found. The worm uses the DNS server configured locally, if available, to perform an MX lookup for the recipient address. If the local DNS fails, it will perform the lookup from a list of hard-coded servers [11]. Therefore, a DNS lookup can be observed and then the SMTP connection based on TCP port 25 will be initiated.

CHAPTER 3

The Proposed System

In this chapter, we describe the details of the implemented worm detection and mitigation system. This chapter is organized as follows. In the next section, we explain how the correlation of Domain Name System (DNS) queries with outgoing connections from the network can be utilized for the detection zero-day scanning worms and mass-mailing worms; an observation that has been previously noted by other security researchers [19,20,36]. In section 3.2, we present the main components of our system and explain the details of the worm detection and mitigation process, including the firewall updates and user notification processes. Finally, some snapshots of the system graphical user interface (GUI) are presented in section 3.3.

3.1 DNS Anomalies of Worms

A typical user usually uses human readable domain names for accessing various services on the Internet. These domain names have to be translated to its associated IP addresses before initiating any communication. A DNS server is the designated network authority responsible for this address mapping. RFC 1035 [37], describes the process of message exchange between

the DNS server and a network node. This process is initiated once a network node requests DNS service for domain name lookup. In response to that request, the DNS responds by sending a DNS response message, which contains a complete description of the domain name stored in the DNS server database from which the lookup has been requested. Once the DNS response message is received, the requesting host retrieves the desired IP address and then starts its communication with its target host using the retrieved IP address.

On the other hand, scanning worms do not contact DNS servers for domain name mapping. Instead, to locate their victims, these worms generate IP addresses directly by themselves without consulting any DNS server. Thus, and as noted in [19], the presence of new connection requests that are not preceded by DNS queries can be considered as a sign of worm infection.

It should be noted, however, that there are other legitimate connections that can be initiated by any host without any prior DNS queries. Among the available application protocols, there are some that can carry embedded IP addresses inside their contents. HTTP is one such protocol that may contain hard coded IP addresses, and URLs that may point to other web servers to retrieve some pictures for performance purposes. VoIP and instance messages protocols may also use many embedded IP addresses for allocating voice relay servers. These embedded IP addresses has to be extracted and their associated connections have to be identified as legitimate. This implies that every network packet has to be parsed to retrieve theses embedded IP addresses. Once extracted, they are added to the safe list. Also, the IP addresses of servers that are running inside the local network are added to this safe list. Moreover, connection initiated by any type of applications that use hard

coded or preconfigured IP addresses inside their configuration files are declared as legal and such addresses are also added to the white list. In addition, IP addresses, which can be categorized as public allocated address space and reserved IP addresses are used publicly over the Internet as well as within local networks for broadcasting and multicasting are also stored white list of our system.

Fig. 3.1 illustrates the difference in the communication steps followed by a normal user and a scanning worm when initiating a new connection.

Similarly, the procedure used by mass-mailing worms to deliver their malicious emails is different from the normal procedure followed by the normal email communication protocol. Normally, each host that wants to send an email has to communicate with the mail server responsible for delivering emails to its intended recipients. Mail servers in turn communicate with DNS servers for determining the target recipient email servers. A special type of communications has to take place between the mail server and the DNS server. Mail server has to initiate Mail eXchange DNS query (MX query), which contains the Fully Qualified Domain Name (FQDN). Afterwards the mail server sends this query to the DNS server in order to retrieve the IP address of the mail server that is responsible for the target host mail box. Then, the mail server starts communicating with the remote mail server to deliver the email.

In contrast to the above procedure of normal email delivery, mass-mailing worms usually use its own SMTP-engines to bypass local email servers security measures. However, they still rely on the DNS servers for locating the respective mail servers of their intended

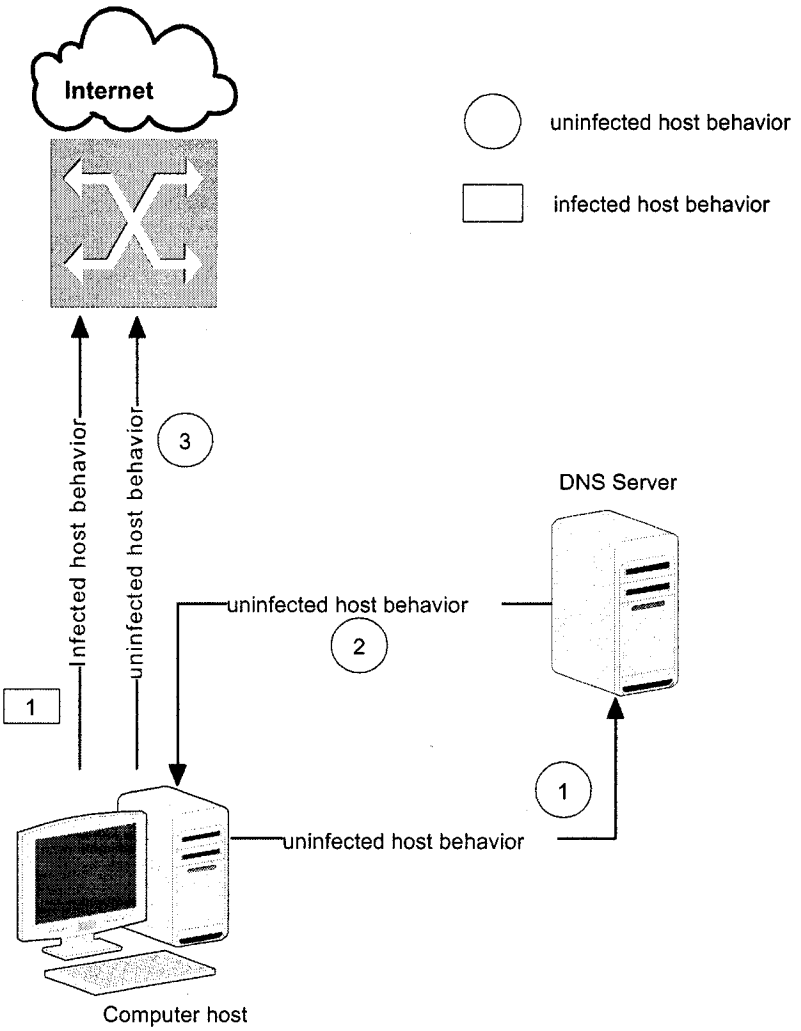

Figure 3.1: DNS anomalies of scanning worms

victims. Creating host-based MX requests is a violation of the typical communication pattern because these requests are supposed to only take place between mail servers and DNS servers [37, 38]. Fig. 3.2 illustrates the difference in the communication steps followed by both the normal host and infected hosts.

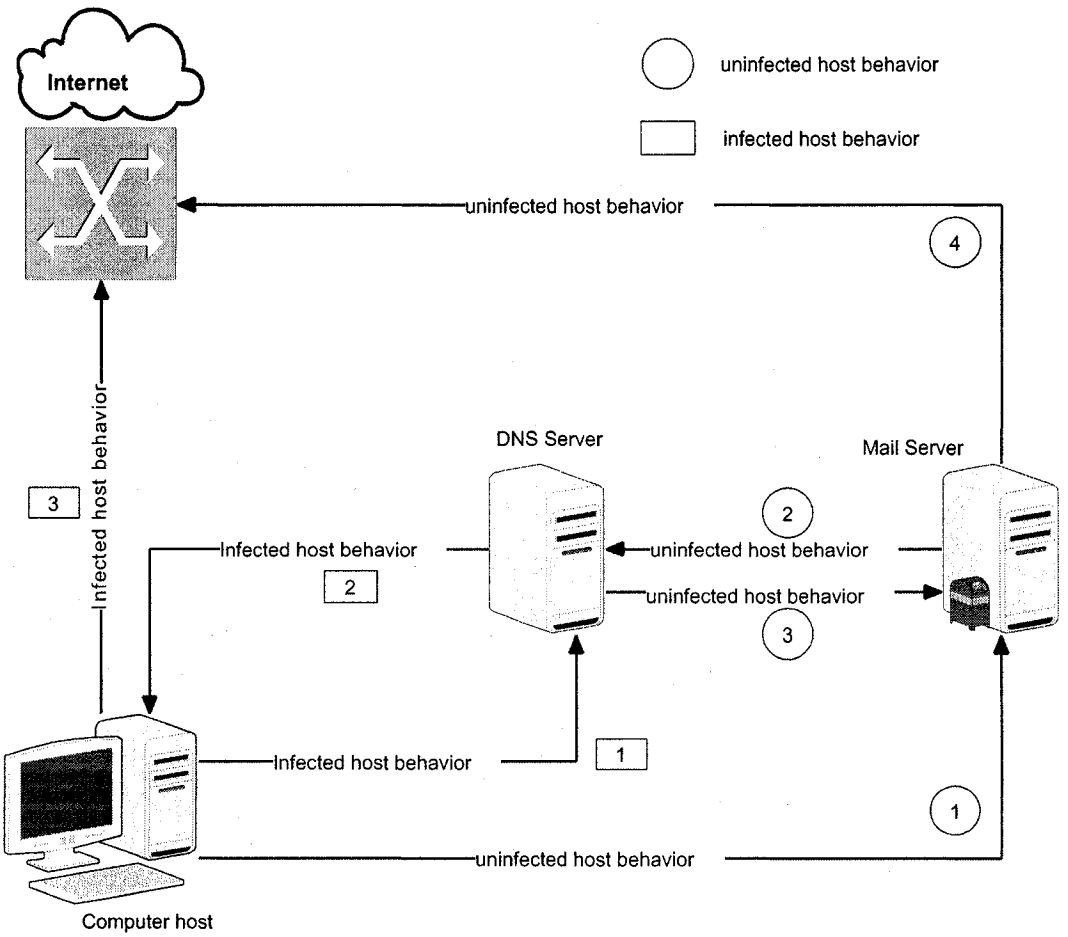

Figure 3.2: DNS anomalies of mass-mailing worms

3.2 System Modules

Fig. 3.3 shows the five main components of our worm detection and mitigation system. In this section, we explain the main functionality of each one of these components and illustrate its interaction with the other components.

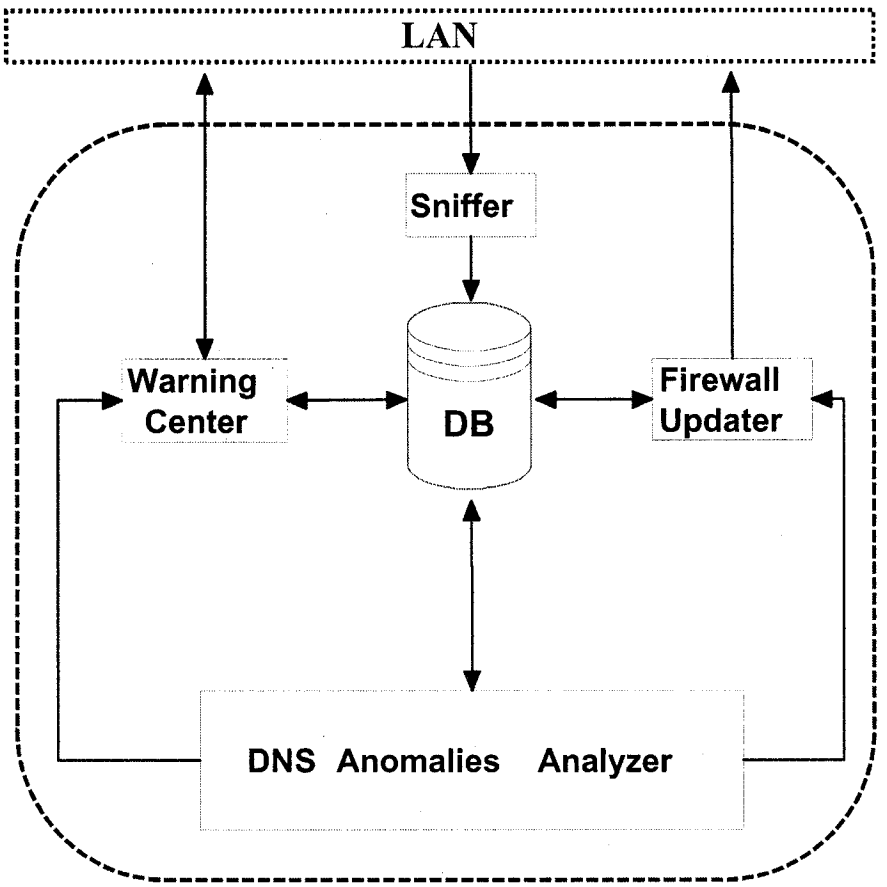

Figure 3.3: Main components of the proposed system

Traffic Sniffing Module

The main objective of this module is to capture all the local area network (LAN) traffic and log it, in real time, to the detection system database. In our implementation, we used Snort [26] Network Intrusion Detection System (NIDS) for implementing this module. By configuring Snort in the sniffing mode, we are able to collect all the network packets and log them, using libpcap, into TCP dump [39] file format.

Database Module

In order to have a scalable and flexible packet analysis module, the sniffed LAN traffic is stored inside a database. We used the database design schema of Snort system to interact with the collected network traffic. This provides an efficient representation of network traffic and helps us to analyze the stored data efficiently. The stored database tables are then managed using MySQL [40] open source Database Management System (DBMS).

DNS Anomalies Analysis Module

This component presents the core of the detection system. By analyzing the sniffed network packets stored in the database, the presence of worms can be detected by checking for any DNS anomaly as described above. Following the same architecture proposed in [19], this module has been decomposed into two components that collaborate together to perform the process of detection and analysis of network traffic: Packet Processing Unit (PPU) and DNS Correlation Unit (DCU).

The PPU is responsible for translating the logged network traffic into useful information and formulating it into specific data structure that would be used by DCU. This unit is connected to the database to retrieve and store the processed information. In particular, the PPU has two main functions: building a local DNS cache, and extracting embedded IP addresses. Local DNS caches can be built by parsing all the DNS protocol packets that come through TCP [41] and UDP [42] connections using port 53. Each DNS cache candidate consists of 4 tuples including source IP address of the DNS record, the domain name that is mentioned in the request, the reply of the query, and the Time To Live (TTL) attribute

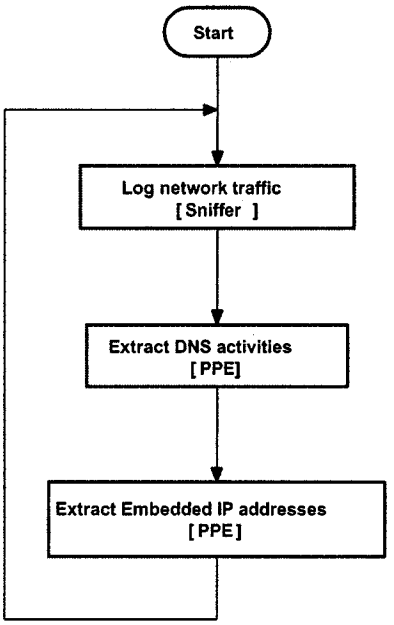

Figure 3.4: Network traffic management flow chart

that indicates when we have to discard the current DNS record. The process of extracting embedded IP addresses is achieved by applying regular expressions functions [43] to each packet payload in order to match and retrieve any numeric IP address representation within a string. The general usage of regular expression scripting language is to match and grip any specific string from text. Fig.3.5 shows an example for a regular expression that can be used to extract both the IP and reverse IP addresses, respectively. Finally, we log all the captured embedded IP addresses in the database along with the whole packet header information for further processing. The above process is illustrated in Fig.3.4.

The DCU is responsible for validating each new connection to check whether it is legitimate or not. It takes the PPU processed information, which is the DNS cache list, the embedded IP addresses list, and the white list. Then, it retrieves every new network

```
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?).(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?).(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)

([0-5]?52|[0-9]?[0-4]?5|[0-9]?[0-9]?[2]|[0-9]?[0-9][01]?)
.([0-5]?52|[0-9]?[0-4]?5|[0-9]?[0-9]?[2]|[0-9]?[0-9][01]?)
.([0-5]?52|[0-9]?[0-4]?5|[0-9]?[0-9]?[2]|[0-9]?[0-9][01]?)
.([0-5]?52|[0-9]?[0-4]?5|[0-9]?[0-9]?[2]|[0-9]?[0-9][01]?)
```

Figure 3.5: An example for regular expression to match IP addresses

connection from the database. For instance, SYN packets are considered as the new connection for the TCP protocol while each UDP packet or echo message is treated as a new connection. After that, it compares the source and the destination IP addresses with every single entry inside all the legitimate lists: DNS cache, embedded IP addresses, and white list. Using the source and the destination IPs, the DCU tries to find any DNS query that is originated from the source IP address and extract the destination IP address from the DNS answer to that query. Then, it checks whether the source or the destination IP addresses are contained inside the embedded IP address list. After this, it checks for any exception in the white list by finding the source or the destination IP addresses inside the white list. During these steps, if any matching entry is found, the DCU skips the current connection and starts analyzing subsequent new connection requests. Otherwise, the DCU identifies this connection as illegitimate connection request and adds it to the malicious activities list. Furthermore, the DCU module sends a request to the firewall management center to block this connection. It also sends another request to the warning center to notify the user of the infected host. New alarms resulting from the same suspicious connection are not sent again to the end-user, i.e., every suspicious connection results only in one alarm sent to the

end-user.

The time expiration information that is stored inside the DNS cache is also maintained by DCU. Usually, the DNS resolver has to manage and flush out the expired DNS records that are locally maintained inside the host machine. However, we have designed our system to flush the expired DNS entries at the end of each day. This is done because there is a high possibility that the system may initiate new connections even after the expiration of the DNS record that was supposed to validate that particular connection.

Similarly, the DCU identifies hosts infected by mass-mailing worms by looking for any MX queries that initiated from hosts, which are not authorized to issue these kinds of queries. The firewall update and the end-user notification procedures are identical to the process above.

Warning Center

This module is responsible for alerting end-users about any suspicious activities that originate from their systems. When using our worm detection system, an additional software component is installed on each end-user host. The warning system interacts with these software components using UDP protocol (using port 1055 on the end-user side and port 1055 on the worm detection machine).

Whenever a connection is flagged as suspicious by the worm detection system, the warning system extract the source IP address and port number, the destination IP address and the port number, protocol, and the timestamp of this connection. Then the warning center checks the history of notification messages for the same connection. Only connections that do not have records within the notification history generate new alarms.

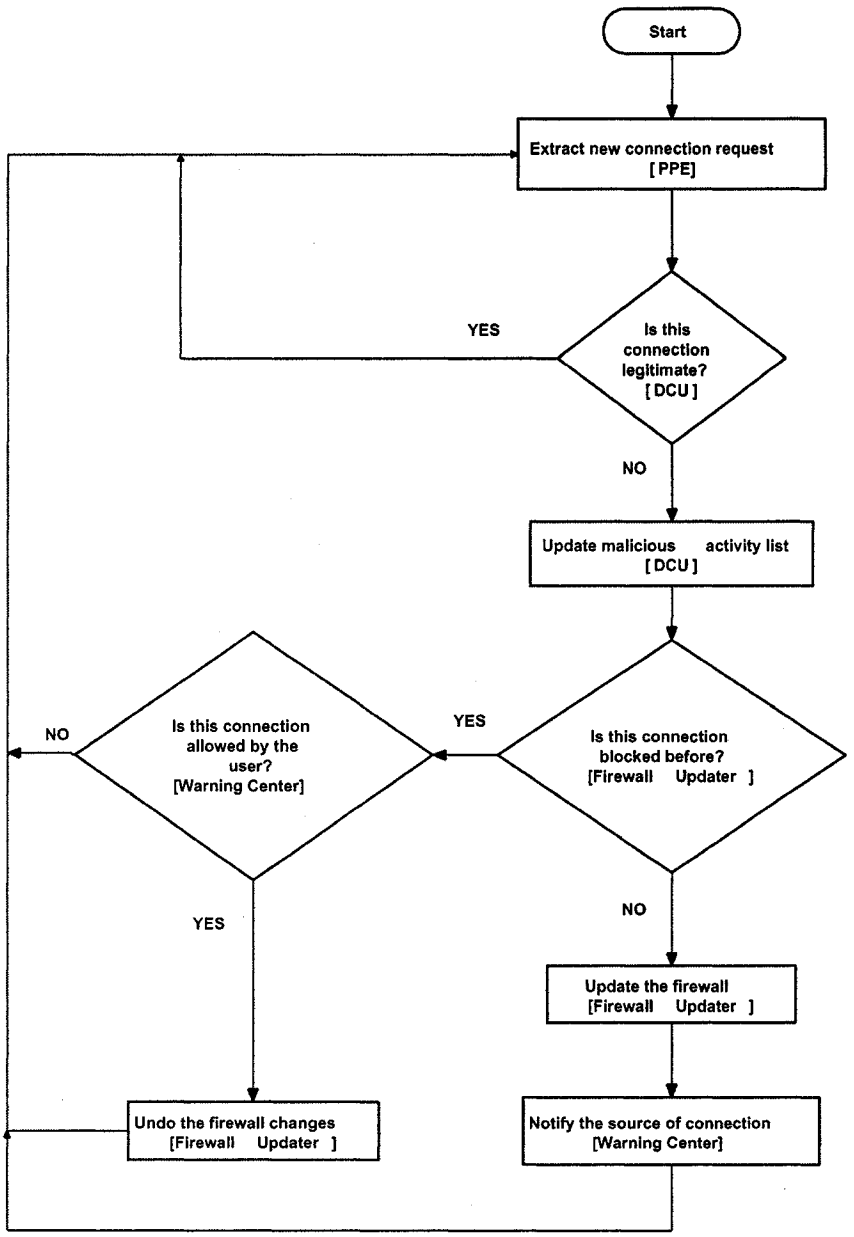

Figure 3.6: Vulnerability worm detection flow chart

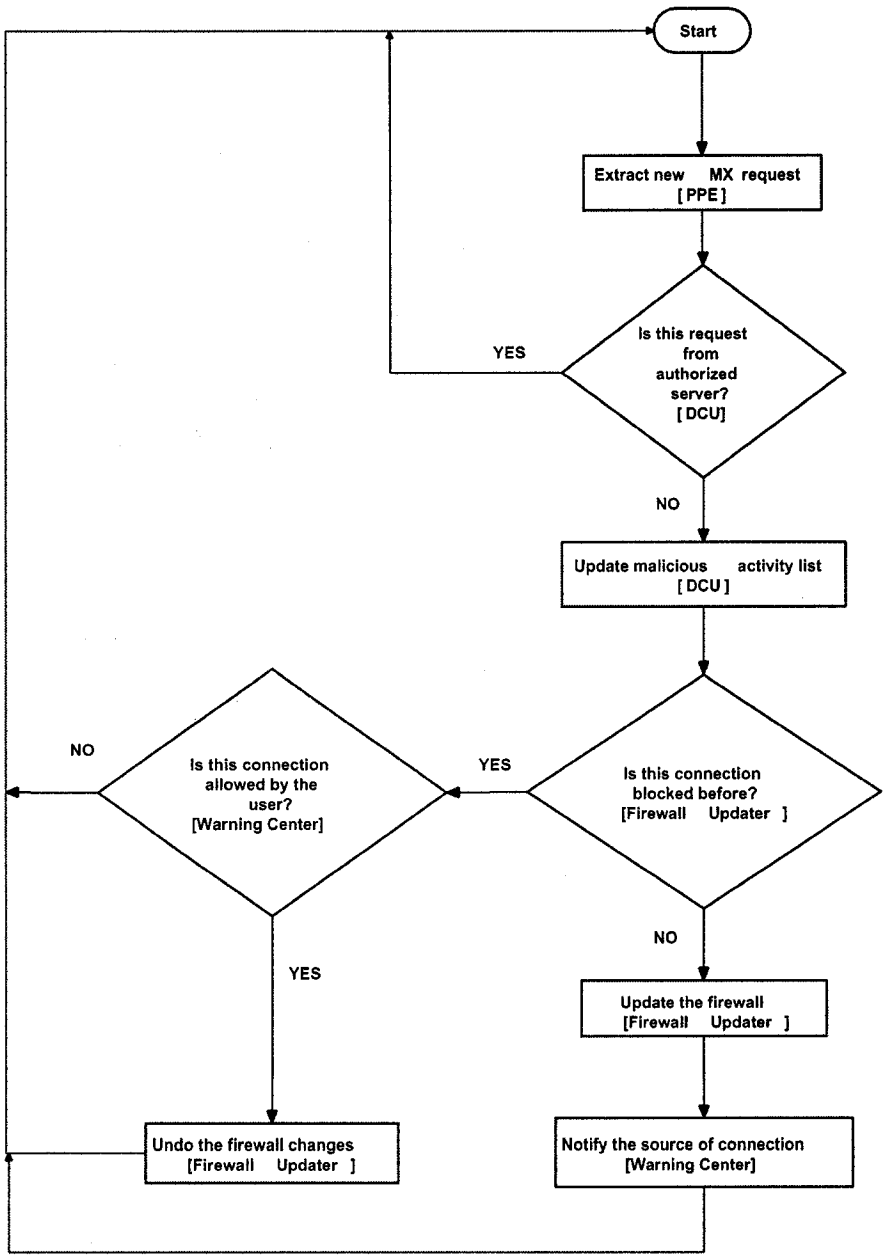

Figure 3.7: Mass-mailing worm detection flow chart

Development and Production Environment	Description	License
Development Platform	Windows XP	Licensed
Development Tool (java)	NetBeans 5.5	Free
Database Server	MySQL	Free
Programming Language	Java	Free
DNS Server	Microsoft Windows Server 2003	Licensed
Router and Firewall System	Fedora core 4	Free

Table 3.1: Detection system development environment

Firewall Updater

The objective of this component is to automatically mitigate the effect of worms by isolating the infected machines (or subnets). Whenever a connection is flagged as malicious, the firewall is automatically updated to block all future connections originating from the suspected host. The firewall updater module is connected by Secure Shell (SSH) protocol [44] directly to the firewall system and is responsible for updating the IP table rules. The system can be configured such that the user response to the alert sent by the warning center can be used to undo the firewall updates and hence helps reduce the effect of false alarms.

Table 3.1 lists the development environment for each component in our detection system design and implementation. The main factor behind our choice is to use open source solutions to facilitate any further extensions.

Fig.3.6 shows the interaction between the various system components during the scanning worm detection process. Similarly, Fig.3.7 shows the corresponding process for mass-mailing worms.

3.3 Graphical User Interface

In this section, we briefly describe the Graphical User Interface (GUI) of our proposed system. One interface is used on the detection machine that is managed by the system administrator and the other one is installed on individual LAN hosts that is use for interaction with the end-user.

Fig. 3.8 shows a snapshot of the main screen of the detection system. Fig. 3.9 shows a snapshot of the white list management screen. Similarly, Fig. 3.10 shows a snapshot of the warning center screen.

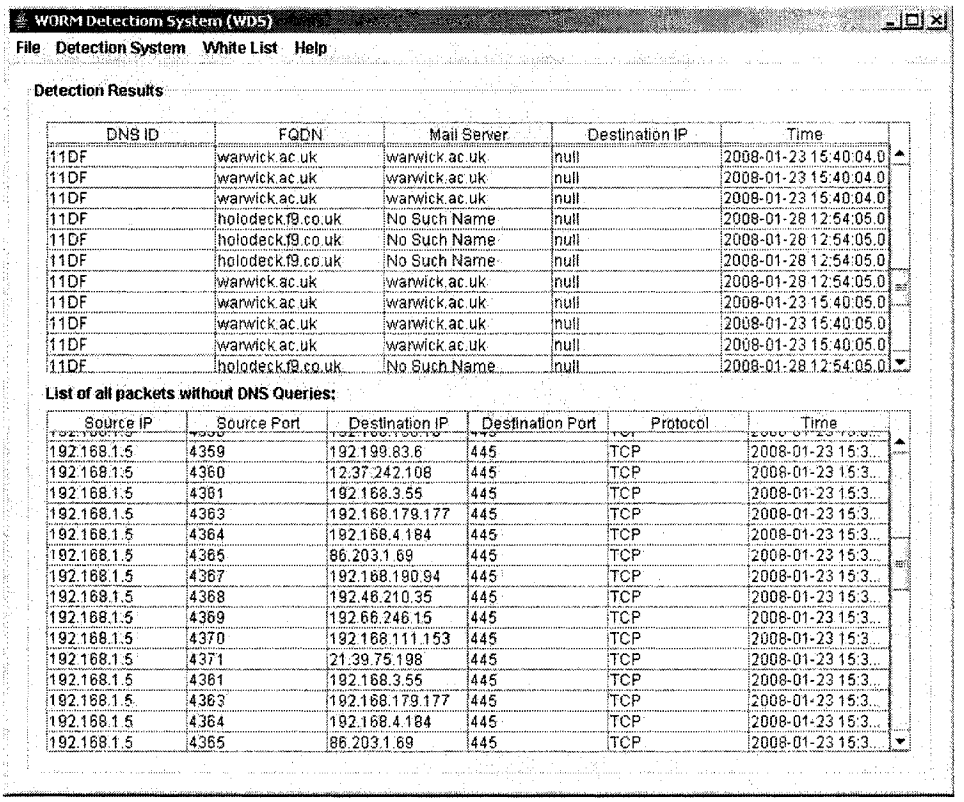

Figure 3.8: Main screen of the detection system

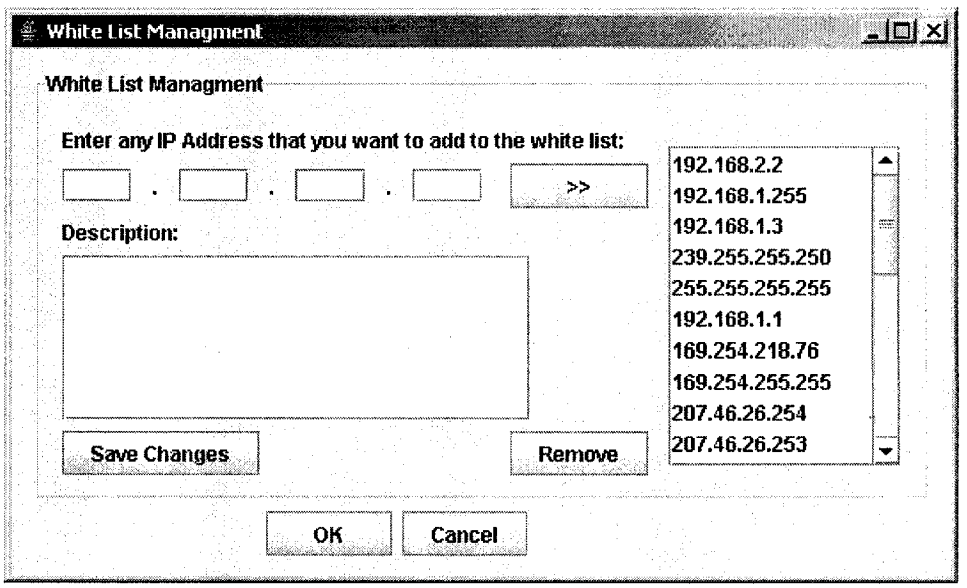

Figure 3.9: White list management screen

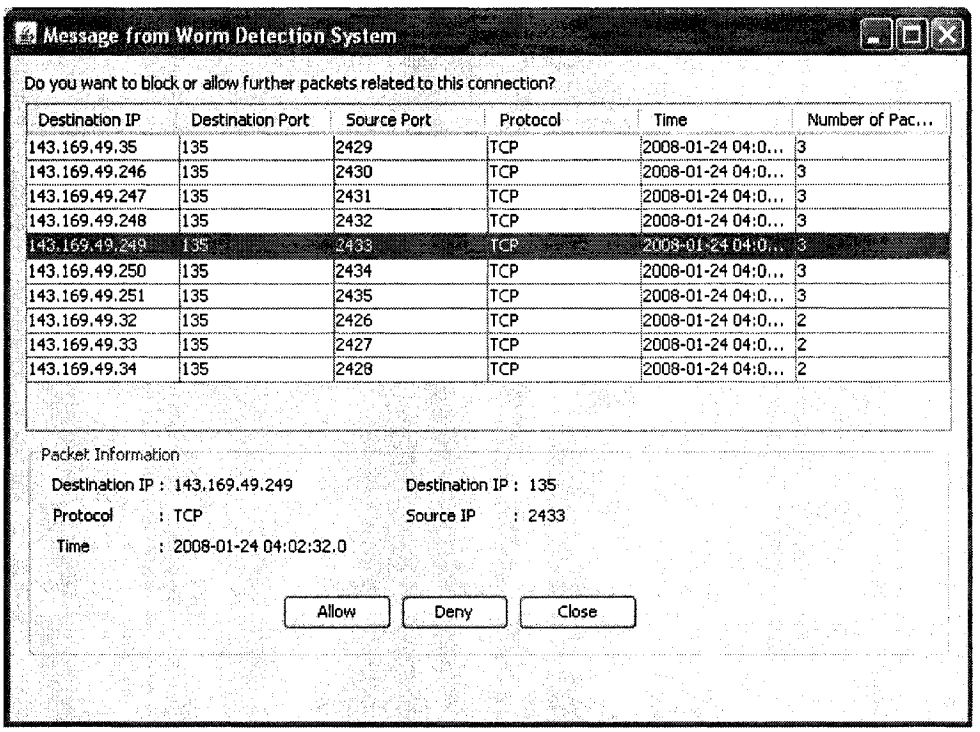

Figure 3.10: Warning center screen

CHAPTER 4

Experimental Results

In this chapter we describe the network setup used to carry out our experiments. We also provide a summary of our experimental results together with some analysis for the obtained results.

4.1 Objectives

In order to test the effectiveness of our proposed system, we carried out two sets of experiments. The objective of the first set of experiments, which was carried out without any real worms, is to identify the causes of false positive [1] alarms. i.e., the cases in which a legitimate user connection is identified by the worm detection system as being generated by a worm.

The objective of the second set of experiments, which was conducted using some real worms in a controlled network setup, is to test the robustness of the overall system. In other words, we want to verify that the developed detection system, including both the automatic firewall updater and warning center, cannot be over run by the scanning speed of modern

[1] In general, a false positive denotes the case when a non-match is declared to be a match.

Internet worms. It should be noted that our system decision is based on deterministic observations. Hence, by design, false negatives do not exist[2]

4.2 Network Setup

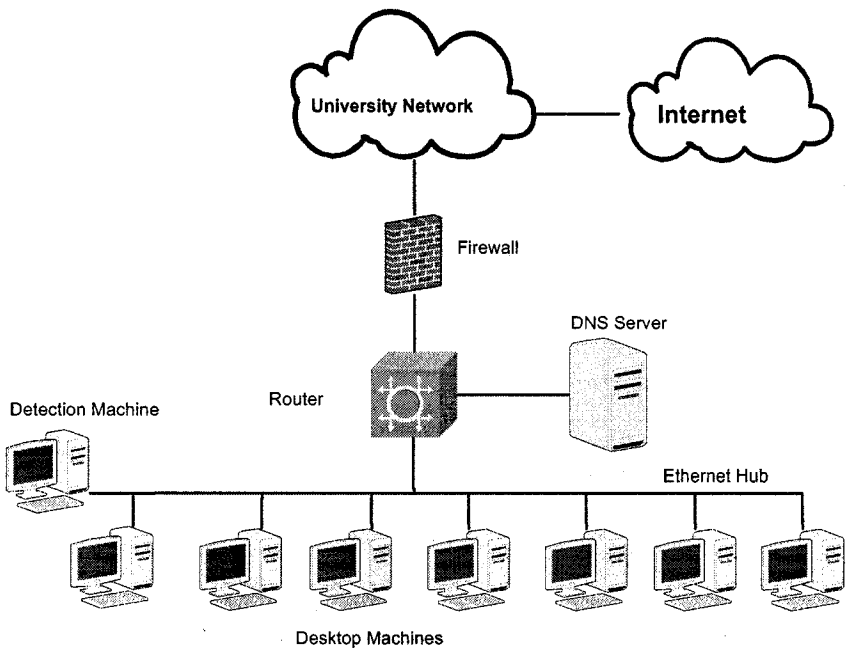

Figure 4.1: Experimental network setup

Fig.4.1 shows the network setup used throughout the first set of our experiments. It consists of two different network segments that are separated by a network router. In the first segment, as shown in the lower part of the figure, we have a set of seven desktop computers[3] that are linked together by an Ethernet hub. Our detection system is part of

[2]A false negative denotes the case when system should have generated an alarm but did not exist.

[3]These computers are the normal workstations used by graduate students, in the Cryptography and Data Security laboratory at Concordia Institute for Information Systems Engineering, to carry out their daily research activities.

this segment. On the other side, we have the DNS server, which is configured to function as a local DNS for the first segment. The LAN and the DNS server are connected to the university network via a firewall. The firewall is configured using Linux system for controlling the outgoing and incoming data flow. Our test network is then connected to the Internet through the university network.

The detection machine, which runs the DNS analysis engine, continuously monitors all the traffic that is generated or received by any local system within the test network. The DNS server is responsible for providing the DNS services for all systems inside the LAN. The firewall system is a configurable software program that is implemented inside a Linux system, which can be controlled remotely by our detection system in such a way that allows new rules to be added and the existing ones to be modified dynamically.

Fig.4.2 shows the network setup used throughout the second set of our experiments. Similar to the first network setup, it consists of two different network segments that are separated by a network router. In the first segment, as shown in the lower part of the figure, we have a set of three desktop computers infected by three Internet worm; two of them are scanning worms (Sasser [11] and Blaster [11]), and the third one is a mass-mailing worm (Netsky [11]). The firewall is configured to block all the traffic originating from these infected machines.

4.3 Experimental Results

In the first phase of our experiment, we run our system for 17 days without interruption. Before starting the experiment, all the laboratory machines were restarted and the DNS

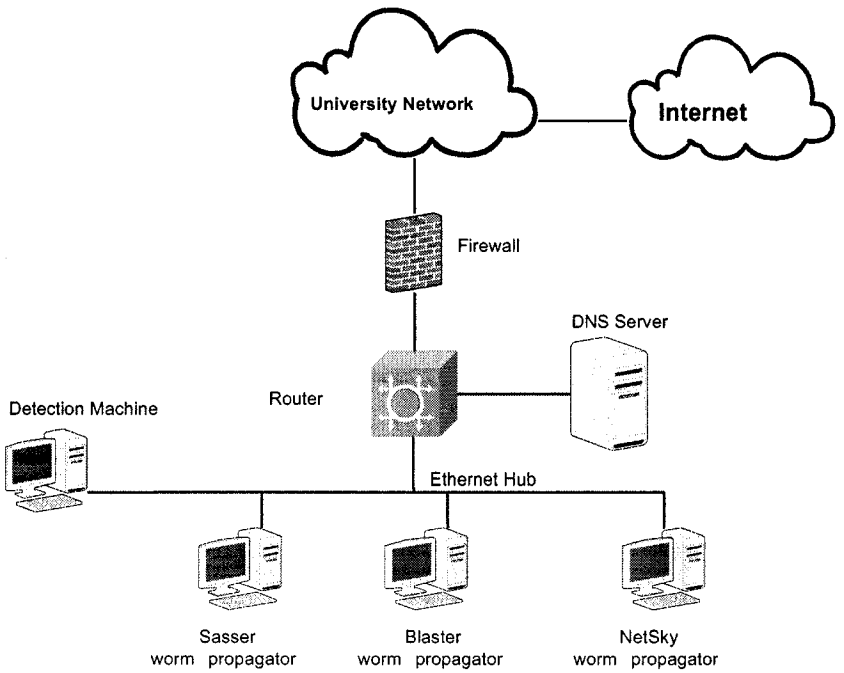

Figure 4.2: Experimental network setup

cache list within every system was flushed to make sure that no DNS queries have been resolved before starting the experiment.

Fig.4.3 shows the total number of new connections that originate from the work stations during the 17 day interval where all connections with the same target IP address were counted once. If a connection with a given target IP address was observed in a given day, it is not counted again if it was observed in any of the subsequent days.

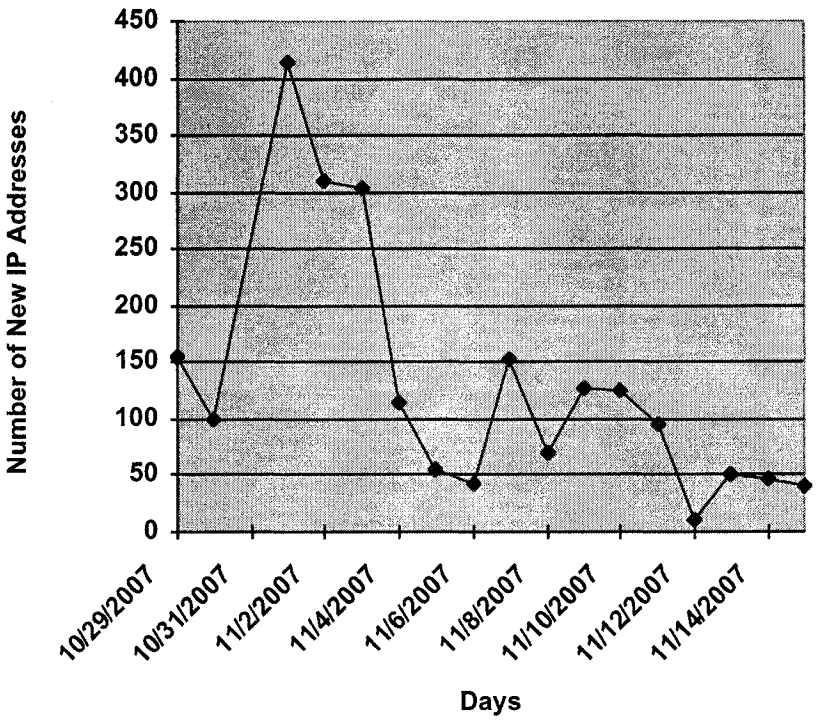

Figure 4.3: New observed unique IP addresses

	IP Address	Purpose
1	192.168.2.2	local DNS server
2	192.168.1.3	Detection Machine
3	192.168.1.1	Default Gateway
4	207.46.26.254	Echo server of Microsoft Windows
5	207.46.26.253	Echo server of Microsoft Windows
6	207.46.130.100	Time Server
7	132.205.100.1	Concordia Network Server
8	132.205.100.255	Concordia Network Server
9	132.205.109.158	Concordia Network Server
10	132.205.15.77	Concordia Network Server
11	132.205.15.82	Concordia Network Server
12	132.205.2.245	Concordia Network Server
13	132.205.222.120	Concordia Network Server
14	132.205.7.51	Concordia Network Server
15	132.205.7.60	Concordia Network Server
16	132.205.96.48	Concordia Network Server
17	132.205.96.93	Concordia Network Server
18	132.205.96.94	Concordia Network Server

Table 4.1: White list entries corresponding to IP addresses of network services

	IP Address	Purpose
1	169.254.0.0/16	link local
2	255.255.255.255	Broadcasting
3	239.255.255.250	Multi Casting
4	192.168.1.255	Broadcasting

Table 4.2: White list entries corresponding to Internet assigned network addresses

Table. 4.1 illustrates all the services that have been recorded in our white list.

Table. 4.2 lists all the observed assigned IP addresses that were obtained during our training period and added to our white list.

Fig.4.4 shows the total number of alerts generated by our system during this interval.

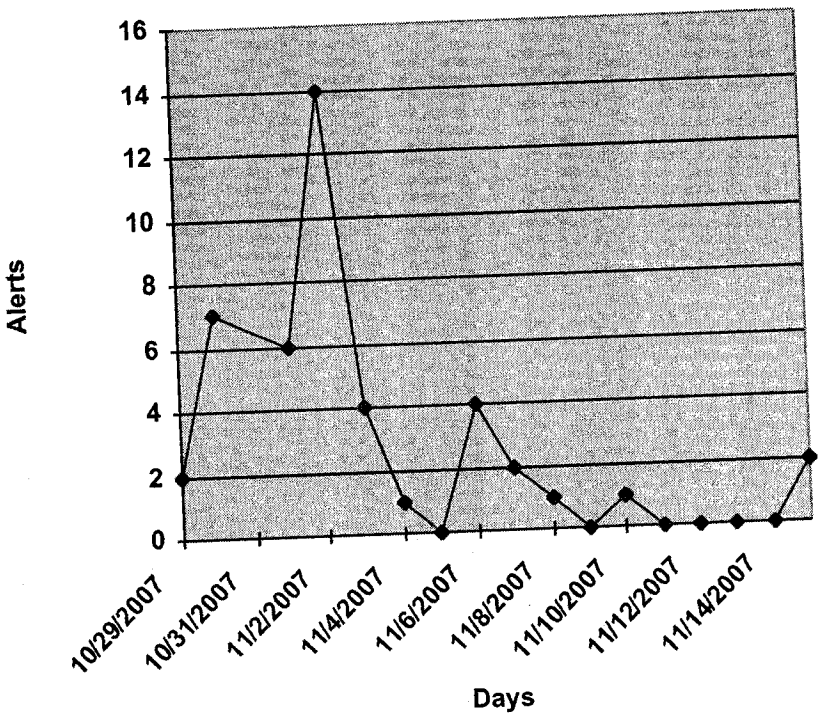

Figure 4.4: Total number of alerts generated by the detection system

Fig.4.5 shows the number of alerts that would have been generated if the white list is excluded from the detection process.

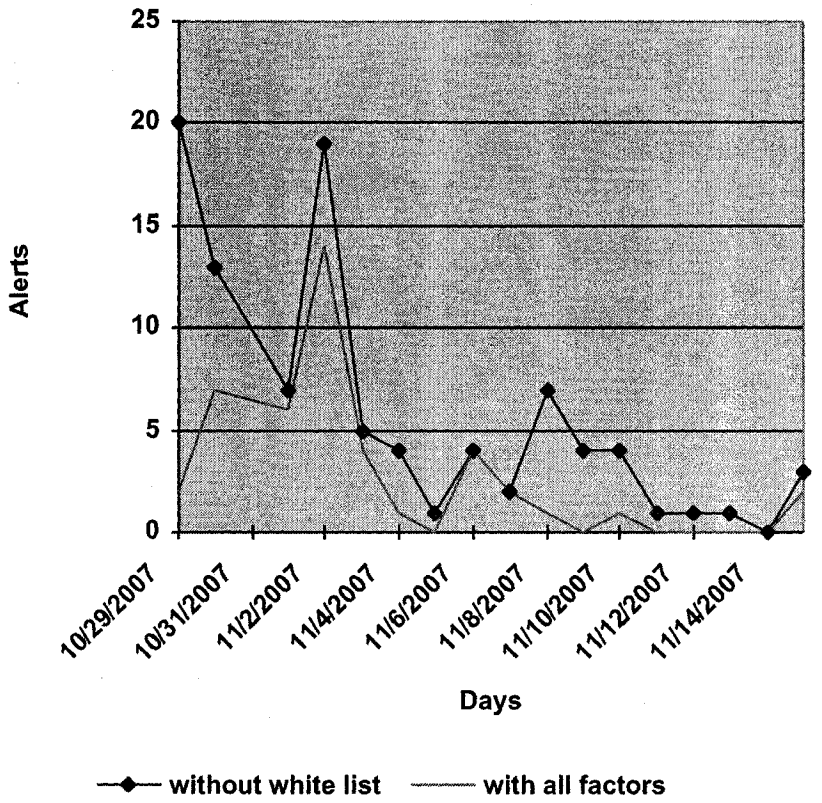

Figure 4.5: Detection system alerts without white list

Fig.4.6 shows the number of alerts that would have been generated if the HTTP embedded IP address list is excluded from the detection process.

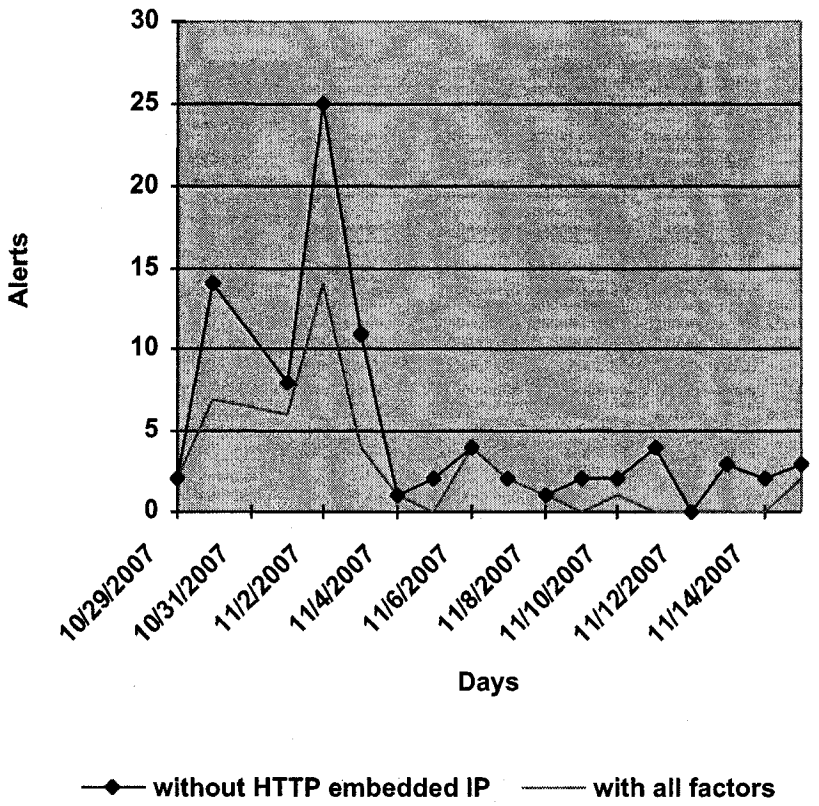

Figure 4.6: Detection system alerts without HTTP embedded IP addresses

Similarly, Fig.4.7 shows the corresponding alarms if the TCP and UDP embedded IP addresses are excluded from the decision process.

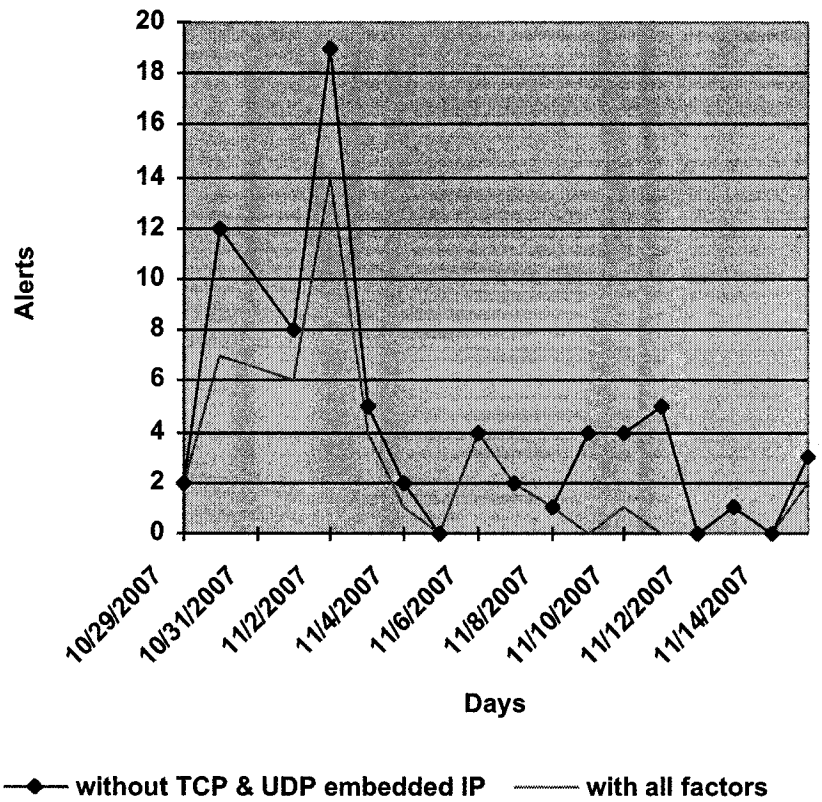

Figure 4.7: Detection system alerts without TCP and UDP embedded IP addresses

White list	Embedded UDP/TCP IP addresses	Embedded HTTP IP addresses
52	28	42

Table 4.3: Total number of eliminated alerts

Day	Sasser	Blaster	NetSky
1	41160	36213	58451
2	41140	38531	60774
3	41150	37411	59390
4	41150	35759	59152

Table 4.4: Number of scans and MX queries generated by the infected machines

Table 4.3 shows a summary for the total number of alerts that were eliminated due to the white list, embedded IP UDP/TCP IP addresses, and embedded HTTP IP addresses, respectively.

From the results above, it is clear that number of false positives can be greatly reduced by by carefully updating the safe lists, e.g., by adding connections manually allowed by the end-user to this list.

Table 4.4 shows the total number of scans generated by Sasser and Blaster, and the number of MX queries generated by Netsky during a 4 day test interval. All these anomalies were successfully captured by our system.

CHAPTER 5

Conclusions and Future Works

In this chapter, we provide a summary of the main features of our developed system. We also point out some future enhancements that can be applied to our system.

5.1 Summary

Throughout this work, we have developed an integrated worm detection and mitigation system. The detection algorithm is based on the observation that DNS activities associated with both scanning worms and mass-mailing worms violate the typical communication pattern followed by normal users. Our system is characterized with the following features:

- Ability to detect all zero-day scanning and mass-mailing worms that cause the prescribed DNS anomalies.
- Scanning rate independence: it can detect slow as well as fast scanning worms.
- Speed: It has the capability to detect infected systems after only a single malicious worm connection attempt.
- Automatic firewall update: this feature mitigates the damage that can result from waiting for a manual action to be performed by system administrators and

isolate infected hosts and subsets.

- User friendly interface: the GUI is carefully designed to facilitate both the system configuration and the monitoring processes.

In addition to its primarily objective, i.e., worm detection and mitigation, the developed system can also be used as an academic tool to study the behavior of real Internet worms.

5.2 Future Works

- Testing the developed system on a larger LAN will provide a better evaluation for the robustness of the systems as well as a better measure for the level of co-operation of typical (non cooperative) end-users.
- Careful investigation of different Internet protocols can lead to discovering other (deterministic or statistical) anomalies associated with worm activities. The system developed in this thesis is based only on DNS anomalies. Considering other protocol anomalies will enable the system to discover larger families of worms.
- Integrating the developed system with other statistical based systems will result in lowering the number of false alarms and improving the detection capabilities of the system.
- The importance of usability issues of security systems have not been addressed adequately in the literature [45]. We believe that the success and wide spread of any security product, among other factors, depends heavily on the ease of its use. Further experiments have to be done to investigate the behavior of the

end-user. For example, mitigating the effect of irresponsible users that always confirm that the suspected connection attempt originating from their machine are legitimate (either because of their negligence, ignorance or because they do not want to interrupt their work) is an interesting multidisciplinary research problem.

- In order to reduce effect of false alarms on the continuity of service to the suspected LAN users, techniques such as virus throttling can be applied before totally isolating suspicious hosts and subnets.

List of References

[1] J. Shoch and J. Hupp, "The worm programsearly experience with a distributed computation," *Commun. ACM*, vol. 25, no. 3, pp. 172–180, 1982.

[2] J. Rochlis and M. Eichin, "With microscope and tweezers: The worm from MIT's perspective," *Commun. ACM*, vol. 32, no. 6, pp. 228 – 245, 1990.

[3] J. Nazario, *Defense and Detection Strategies against Internet Worms.* Norwood, MA, USA: Artech House, Inc., 2003.

[4] "Code Red worm exploiting buffer overflow in IIS indexing service DLL," 2001, last accessed: February 2008. [Online]. Available: http://www.cert.org/advisories/CA-2001-23.html

[5] "Exploitation of vulnerabilities in microsoft sql server," 2002, last accessed: February 2008. [Online]. Available: http://www.cert.org/incident_notes/IN-2002-04.html

[6] J. Ullrich, "MSSQL worm (sqlsnake) on the rise," 2001, last accessed: February 2008. [Online]. Available: http://www.sans.org/resources/idfaq/spider.php

[7] "CERT advisory ca-2001-26 Nimda worm," 2001, last accessed: February 2008. [Online]. Available: http://www.cert.org/advisories/CA-2001-26.html

[8] "CERT advisory ca-2001-26: MS-SQL server worm," 2003, last accessed: February 2008. [Online]. Available: http://www.cert.org/advisories/CA-2003-04.html

[9] "CERT advisory ca-1999-04 melissa macro virus," 1999, last accessed: February 2008. [Online]. Available: http://www.cert.org/advisories/CA-1999-04.html

[10] "CERT advisory w32/novarg.a virus," 2004, last accessed: February 2008. [Online]. Available: http://www.cert.org/incidentnotes/IN-2004-01.html

[11] "Symantec security response," 2008, last accessed: Febrauary 2008. [Online]. Available: http://www.symantec.com/security_response/

[12] G. Christoph and R. Hiestand, "Scan detection based identification of worm-infected hosts," Master's thesis, ETH Zurich, April 2005, master Thesis.

[13] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," *Security and Privacy*, pp. 226 – 241, 2005.

[14] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, 2002.

[15] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03)*, pp. 251–261.

[16] H. Javitz and A. Valdes, "The NIDES statistical component description and justification," Computer Science Laboratory, Technical Report, 1994.

[17] K. Wang and S. Stolfo, "Anomalous payload-based network intrusion detection," in *in Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID '2004)*, pp. 227–246.

[18] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *in Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '2003)*, April 2003.

[19] D. Whyte, E. Kranakis, and P. van Oorschot, "DNS-based detection of scanning worms in an enterprise network," in *Network and Distributed System Security Symposium (NDSS'05)*, February 2005.

[20] D. Whyte, P. van Oorschot, and E. Kranakis, "Addressing SMTP-based mass-mailing activity within enterprise networks," *Computer Security Applications Conference*, vol. 22, no. 06, pp. 393 – 402, December 2006.

[21] P. Szor, *The Art of Computer Virus Research and Defense.* Addison-Wesley Professional, 2005.

[22] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode.* New York, NY, USA: ACM, 2003, pp. 11–18.

[23] C. Bo, B. Xingfang, and X. Yun, "A new approach for early detection of internet worms based on connection degree," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, August 2005.

[24] J. Twycross and M. Williamson, "Implementing and testing a virus throttle," in *Proceedings 12th USENIX Security Symposium*, August 2003.

[25] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," in *In Proc. IEEE Symposium on Research in Security and Privacy*, 1990, pp. 296–304.

[26] "The open source network intrusion detection system," 2008, last accessed: February 2008. [Online]. Available: http://www.snort.org

[27] V. Paxson, J. Rothfuss, and B. Tierney, "Bro user manual," Bro IDS, Internet–Draft, December 2004, available at :http://www.bro-ids.org/Bro-user-manual.pdf.

[28] "Bro intrusion detection system," last accessed: February 2008. [Online]. Available: http://bro-ids.org

[29] S. Schechter, J. Jung, and A. Berger, "Fast detection of scanning worm infections," in *In Proceedings of Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[30] W. Matthew, "Design, implementation and test of an email virus throttle," in *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003).*, December 2003, pp. 76 – 85.

[31] M. Mannan and P. Oorschot, "On instant messaging worms, analysis and countermeasures," in *WORM '05: Proceedings of the 2005 ACM workshop on Rapid malcode.* ACM, 2005, pp. 2–11.

[32] D. Dagon, X. Quin, W. Lee, J. Grizzard, J. Levine, and H. Owen, "Honeystat: Local worm detection using honeypots," in *Seventh International Symposium on Recent Advances in Intrusion Detection*, September 2004.

[33] "Know your enemy: Worms at war," November 2000, last accessed: February 2008. [Online]. Available: http://project.honeynet.org/papers/worm/

[34] "Analysis: Blaster worm," 2003, last accessed: February 2008. [Online]. Available: http://research.eeye.com/html/advisories/published/AL20030811.html

[35] "Analysis: Sasser worm," 2004, last accessed: February 2008. [Online]. Available: http://research.eeye.com/html/advisories/published/AD20040501.html

[36] Y. Musashi, R. Matsuba, and K. Sugitani, "Indirect detection of mass-mailing worm-infected PC terminals for learners," in *ICETA2004*, 2004.

[37] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, Tech. Rep., November 1987.

[38] J. Postel, "Simple mail transport protocol," RFC 821, Tech. Rep., August 1982.

[39] "TCP dump," 2008, last accessed: February 2008. [Online]. Available: http://www.tcpdump.org

[40] "MySQL AB :the world's most popular open source database," 2008, last accessed: February 2008. [Online]. Available: http://www.mysql.com

[41] J. Postel, "Transmission control protocol," RFC 793, Tech. Rep., September 1981.

[42] J. Postel, "User datagram protocol," RFC 768, Tech. Rep., August 1980.

[43] "Regular expression," last accessed: February 2008. [Online]. Available: http://en.wikipedia.org/wiki/Regular_expression

[44] "Secure shell," last accessed: February 2008. [Online]. Available: http://en.wikipedia.org/wiki/Secure_Shell

[45] L. F. Cranor and S. Garfinkel, *Security and Usability : Designing Secure Systems that People Can Use.* O'Reilly, 2005.