DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEFENCE SYSTEM MODELLING AGAINST COMPUTER WORMS

by
Emre ERKAT

September, 2008
İZMİR

DEFENCE SYSTEM MODELLING AGAINST COMPUTER WORMS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Electrical and Electronics Engineering

by

Emre ERKAT

September, 2008

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DEFENCE SYSTEM MODELLING AGAINST COMPUTER WORMS"** completed by **EMRE ERKAT** under supervision of **ASST. PROF. DR. ZAFER DİCLE** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Asst. Prof. Dr. Zafer DİCLE
Supervisor

.....

(Jury Member)

.....

(Jury Member)

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to give my sincere thanks to my supervisor, Asst. Prof. Dr. Zafer DİCLE for his guidance, advice and encouragement along the fulfillment of this project.

Emre ERKAT

DEFENCE SYSTEM MODELLING AGAINST COMPUTER WORMS

ABSTRACT

As computer networks become prevalent, the Internet has been a battlefield for attackers and defenders. One of the most powerful weapons for attackers is the computer worm. Computer worms are a self-propagating computer program that is being increasingly and widely used to attack the Internet. Because they spread extremely fast and usually install malicious code, computer worms are so dangerous.

This thesis begins with definition, history and taxonomy. Also, it defines the structure and components of worms. It develops a life cycle model of worm defence, including prevention, prediction, detection and mitigation. It also discusses in detail about each of these techniques. It explains detection and defense techniques against to the computer worms. Group based model have been developed and discussed with the simulations results. It concludes that computer worms are dangerous but there are ways and means to mitigate their ill effects.

Keywords: Computer worms, Computer security, Network security, Defense system

BİLGİSAYAR KURTLARINA KARŞI SAVUNMA SİSTEMİ MODELLEMESİ

ÖZ

Bilgisayar ağlarının yaygın olması ile birlikte, saldıran ve savunanlar için internet bir savaş alanı oldu. Saldıranlar için en güçlü silahlardan birisi bilgisayar kurtlarıdır. Bilgisayar kurtları, sürekli artarak kendi kendine yayılabilen bilgisayar programlarıdır ve internete saldırmak için geniş bir kullanım alanına sahiptir. Oldukça hızlı yayıldıkları için ve genellikle zararlı kodlar yükledikleri için bilgisayar kurtları çok tehlikelidir.

Bu tez, bilgisayar kurtlarının tanımı, geçmişi ve sınıflandırılması ile başlıyor. Ayrıca, bilgisayar kurtlarının yapısını ve komponentlerini tanımlıyor. Bilgisayar kurtlarına karşı savunmada; önleme, tahmin, bulma ve azaltmayı içeren yaşam döngüsü modelini geliştirir. Ayrıca bu tekniklerin detayların da söz edilmiştir. Bilgisayar kurtlarını bulma ve savunma teknikleri açıklanmıştır. Hiyerarşik model geliştirilmiş ve simülasyon sonuçlarından bahsedilmiştir. Bilgisayar kurtlarının tehlikeli olduğu fakat kötü etkilerini azaltmak için yollar olduğu sonucuna varmıştır.

Keywords: Bilgisayar kurtları, Bilgisayar güvenliği, Ağ güvenliği, Savunma sistemi

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

This thesis provides a perspective of computer worms, explores the various worm technologies and popular worms of the past, present and future. It primarily deals with stopping a worm on its tracks without human intervention. Several strategies have been proposed and analyzed with simulations.

1.1 Contribution of This Thesis To The Field

This thesis begins by providing a model of a simple worm and an extensive background about worms of the past, present and future. It develops a simple comprehensible model of a worm. It discusses the various scanning techniques and gives a broad classiffcation of worms. It develops a life cycle model for defense against computer worms. It also discusses several defensive techniques and strategies like prevention, prediction, detection and mitigation. All the above together serves as a compact compendium of worm technologies for the computer security community. This is one of the contributions of this thesis to the computer and network security community.

It also develops and analyzes several indigenous and innovative techniques to address the problem of computer worms. It develops a mitigation model, the grop based model. This research shows how to stop worms in its tracks without human intervention. These form the contributions of this thesis to the field of computer and network security.

1.2 Research Objectives and Solutions

The objective of this thesis is to model and defend against worm attacks without human intervention. Several strategies have been analyzed and with simulations. We

attempt to answer the following important questions:

- What is a worm?
- What are the components of worms and how they propagate in the Internet?
- What are the detection methods against the worms? What are the advantages and disadvantages of these methods?
- What are the defence strategies and methods against the worms? What are the strengths and weaknesses of these methods?
- How can we defend against worms?

To investigate these questions, we apply mathematical modeling methodology and verify analytical results through simulations. Mathematical models can provide quantitative analysis on the propagation dynamics of worms and the effectiveness of defense systems. Simulations are used to verify our model.

In this thesis, the following four topics are investigated:

1. Worms and their scan techniques: In order to analyze the worms we start by providing a definition about computer worms and an extensive background about them including their history and taxonomy. At the core of any worm system are five components. A worm may contain any or all of these components, usually in some combination. In order to propagate itself in the Internet, a worm needs to find vulnerable machines and then infect them. To find vulnerable machines, a worm can either simply scan the entire IP address space randomly, or may perform various strategies to scan the entire or partial IP address space to find targeted hosts. We investigated various scan strategies and analyzed their spreading speed.

2. Analyzing of the worm detection methods: There are different methods of worm detection. These methods are traffic analysis, the use of honeypots, dark network monitors, and the employment of signature-based detection systems. These methods form the core of detecting both hackers and worms. The goal of our detection strategies is to detect nearly any type of worm with as little effort as possible. To do this, we will focus on the features common to most worm types and build strategies to detect these characteristics. While no single methods work for all worm types, a combination of efforts can provide more complete coverage.

3. Analyzing of the worm defence methods: There are various stages of the life cycle of worm defense. The life cycles contains following steps: Prevention, prediction, detection, analysis, mitigation, curing, vaccination and patch similar vulnerabilities. As defence strategies against the worms, there are 2 defence strategies, active and passive. These strategies have some weaknesses and some strengths.

4. Modeling a defence system against the computer worms: This model of defence is based on the willing co-operation of a set of hosts on a pre-arranged protocol. We develop mathematical models for the simplest of the scenarios. Then, we go on to develop simulations to study more complex scenarios of worm mitigation. Grop based model of worm defence is discussed with the simulations results.

1.3 Thesis Outline

This thesis starts off by providing a definition about computer worms and an extensive background about them including their history and taxonomy. Chapter 3 presents various techniques used by worms to scan the Internet to find hosts susceptible to infection. The chapter following that discusses future worms. Chapter 5 develops a life cycle model for the defense against worms. Chapter 6 analyzes various techniques about the worm detection. The next chapter mentions about the

active and passive defences against the computer worms. Grop based model of worm defence is discussed with the simulations results in the chapters following. The last chapter of this thesis present the conclusions and future directions of this research respectively.

CHAPTER TWO
WORMS DEFINED

2.1 Definition

A computer worm is a self-replicating computer program. It uses a network to send copies of itself to other nodes (computer terminals on the network) and it may do so without any user intervention. (Wikipedia, 2007)

Computer worms and viruses are typically grouped together as infectious agents that replicate themselves and spread from system to system. However, they have different properties and capabilities.

Computer worms must be differentiated from computer viruses if we are to understand how they operate, spread, and can be defended against. Failure to do so can lead to an ineffective detection and defense strategy. Like a virus, computer worms alter the behavior of the computers they infect. Computer worms typically install themselves onto the infected system and begin execution, utilizing the host system's resources, including its network connection and storage capabilities. Although many of the features of each are similar, worms differ from computer viruses in several key areas:

- Both worms and viruses spread from a computer to other computers. However, viruses typically spread by attaching themselves to files (either data files or executable applications). Their spread requires the transmission of the infected file from one system to another. Worms, in contrast, are capable of autonomous migration from system to system via the network without the assistance of external software.

- A worm is an active and volatile automated delivery system that controls the medium (typically a network) used to reach a specific target system.

- Viruses, in contrast, are a static medium that does not control the distribution medium.

- Worm nodes can sometimes communicate with other nodes or a central site. Viruses, in contrast, do not communicate with external systems.

2.1.1 A Formal Definition

From the 1991 appeal by R. T. Morris regarding the operation of the 1988 worm that bears his name, the court defined a computer worm as follows:

In the colorful argot of computers, a “worm” is a program that travels from one computer to another but does not attach itself to the operating system of the computer it “infects.” It differs from a “virus,” which is also a migrating program, but one that attaches itself to the operating system of any computer it enters and can infect any other computer that uses files from the infected computer.

This definition, as we will see later, limits itself to agents that do not alter the operating system. Many worms hide their presence by installing software, or root kits, to deliberately hide their presence, some use kernel modules to accomplish this. Such an instance of a worm would not be covered by the above definition.

We will define a computer worm as an independently replicating and autonomous infection agent, capable of seeking out new host systems and infecting them via the network.

2.2 Worm History

The term worm comes from the book Shockwave Rider by John Brunner. Published in 1975, it is a science fiction novel about the future of computing. In the

novel, the heroes defeat a government that has become an enemy by unleashing a computer worm. It congests the network to such an extreme that the government must shut it down.

2.2.1 The First Computer Worm

The Morris worm or Internet worm was one of the first computer worms distributed via the Internet; it is considered the first worm and was certainly the first to gain significant mainstream media attention. It also resulted in the first conviction under the 1986 Computer Fraud and Abuse Act.

According to its creator, the Morris worm was not written to cause damage, but to gauge the size of the Internet. An unintended consequence of the code, however, caused it to be more damaging: a computer could be infected multiple times and each additional process would slow the machine down, eventually to the point of being unusable. The Morris worm worked by exploiting known vulnerabilities in Unix sendmail, Finger, rsh/rexec and weak passwords.

2.2.2 Cycles of Worm Releases

Just as vulnerabilities have a window of exposure between the release of information about the vulnerability and the widespread use of exploits against them, worms have an interval of time between the release of the vulnerability and the appearance of the worm. Nearly any widespread application with a vulnerability can be capitalized on by a worm.

Table 2.1 shows the interval between the release of information about a vulnerability and the introduction of a worm that has exploited that weakness. Some worms are fast to appear, such as the Slapper worm (with an interval of 11 days), while others are much slower such as the sadmind/IIS worm (with a minimum internal of 210 days). This table clearly illustrates the need to evaluate patches for

known vulnerabilities and implement them as efficiently as possible as a means to stop the spread of future worms.

Table 2.1 Interval between Vulnerability Announcement and Worm Appearance

Name	Vulnerability Announced	Worm Found	Interval(Days)
SQLsnake	November 27, 2001	May 22, 2002	176
Code Red	June 19, 2001	July 19, 2001	30
Nimda	May 15, 2001 August 6, 2001 April 3, 2001	September 18, 2001	126 42 168
Sadmind/IIS	December 14, 1999 October 10, 2000	May 8, 2001	511 210
Ramen	July 7, 2000 July 16, 2000 September 25, 2000	January 18, 2001	195 186 115
Slapper	July 30, 2002	September 14, 2002	45
Scalper	June 17, 2002	June 28, 2002	11
Sapphire	July 24, 2002	January 25, 2003	184

This relates directly to the importance of the rapid deployment of security patches to hosts and the sound design of a network. Worms can appear rapidly (as the Slapper worm did), quickly changing the job of a security administrator or architect from prevention to damage control.

2.3 Worm Taxonomy

Figure 2.1 shows a generalized lineage of many of the worms discussed here. From their roots in the research at Xerox PARC to the Morris worm, UNIX and Windows worms have evolved somewhat independently. Although they share key concepts, the methodology of spreading differs between the two types of hosts.

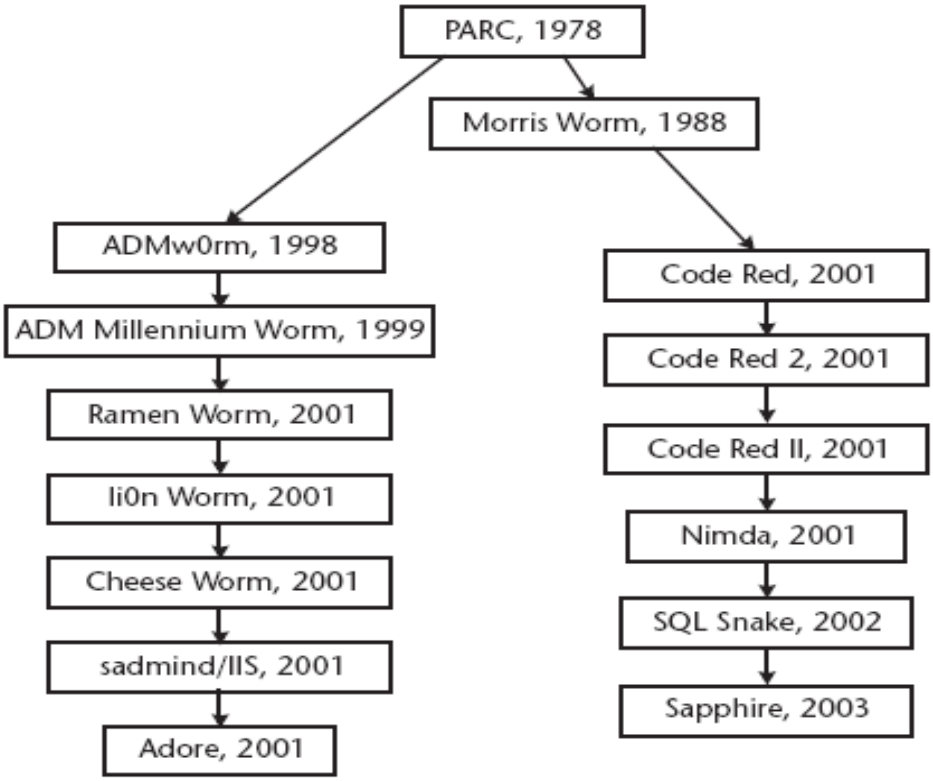

Figure 2.1 A Lineage of Internet Worms. UNIX hosts (left-hand column) and Windows hosts (right-hand column)

2.3.1 Unix Targets

While the free UNIX systems (Linux and the BSD systems) have lagged far behind Windows in terms of popularity, they have been the targets of several worms in recent years. Although these worms have not had as large an impact on the overall performance and security of the Internet when compared to Windows worm incidents, their impact has been noticeable, as described in the preceding chapter.

The popularity of free UNIX systems as a target for worms is probably due to three factors. First, they are a popular choice as a workstation platform for many attackers, giving them ample time to develop familiarity with the weaknesses in UNIX systems. Secondly, UNIX lends itself well to scripting and networking, which

are backbone assets in worm systems. Last, compilers are freely available for the systems, meaning that attackers can develop binary worm components for use on these systems.

2.3.2 Windows Targets

At this time, Microsoft Windows systems make up a majority of the personal computers today. As such, they make an attractive target for a worm to attack. Several recent incidents have shown the scale of damage that can be done by attacking even just one vulnerability in these systems. Windows worms have quickly gone from simple to efficient, each time increasing their capability to do damage.

More than 90% of the personal computer systems in operation use some form of Microsoft Windows. This homogeneous environment mimics that capitalized on by the Morris worm in 1988. By developing an attack for one type of widely deployed host, an attacker can expect to leverage a broad base for their worm.

The more devastating Windows worms have attacked IIS Web servers. Web servers, by their design, communicate to the world at large and handle requests from a multitude of clients. IIS, Microsoft's Web server software, has been the subject of much scrutiny by the security community. As flaws have been found, exploits have been developed against them, some of these being incorporated into worms.

2.4 Components of Worm

At the core of any worm system are five components. A worm may contain any or all of these components, usually in some combination. These components are:

- **Reconnaissance:** The worm network has to hunt out other network nodes to infect. This component of the worm is responsible for discovering hosts on

the network that are capable of being compromised by the worm's known methods.

- **Attack components:** These are used to launch an attack against an identified target system. Attacks can include the traditional buffer or heap overflow, string formatting attacks, Unicode misinterpetations (in the case of IIS attacks), and misconfigurations.

- **Communication components:** Nodes in the worm network can talk to each other. The communication components give the worms the interface to send messages between nodes or some other central location.

- **Command components:** Once compromised, the nodes in the worm network can be issued operation commands using this component. The command element provides the interface to the worm node to issue and act on commands.

- **Intelligence components:** To communicate effectively, the worm network needs to know the location of the nodes as well as characteristics about them. The intelligence portion of the worm network provides the information needed to be able to contact other worm nodes, which can be accomplished in a variety of ways. (Nazario, 2001)

2.4.1 Reconnaissance Component

This is the mechanism by which the system extends its view of the world around itself, determines information about the systems and networks around it, and identifies targets.

When an attacker performs these actions, they have at their disposal a suite of methodologies. By identifying the characteristics which define a system tobe of one

type, or more importantly of a vulnerability, they can identify systems which will become targets.

This component of the worm performs these same processes, but in an automated fashion. This includes scans and sweeps, such as port scans of a block of machines or service sweeps of a network, which are usually active in nature. The system sends stimuli at a possible target, and based upon the responses received it can determine what hosts are active and listening, what ports are open and accessible, and even what operating system the target is running. The configuration of the machine may also be examined by the worm to determine trusted hosts, a technique utilized by the Morris worm.

Having analyzed the network and hosts around itself, the system node can identify targets on a variety of criteria. This includes the capabilities available to the system, position in a network in relation to a goal, or the system profile, such as a poorly configured, rarely monitored target.

Currently, a variety of methods exist to obtain this information in a manual fashion. This can be readily scripted to perform wide area intelligence gathering, but the data is usually manually analyzed. By incorporating these techniques into a worm system component, the system can gain information as it progresses. This information can be shared using communications channels and stored in the intelligence component, if so desired.

2.4.2 Attack Component

The worm's attack components are their most visible and prevalent element. This is the means by which worm systems gain entry on remote systems and begin their infection cycle. These methods can include the standard remote exploits, such as buffer overflows, cgi-bin errors, or similar, or they can include Trojan horse methods.

This component has to be further subdivided into two portions: the platform on which the worm is executing and the platform of the target. This attack element can be a compiled binary or an interpreted script, which utilizes a network component from the attacking host, such as a client socket or a network aware application, to transfer itself to its victim.

A main factor of the attack component is the nature of the target being attacked, specifically its platform and operating system. Attack components that are limited to one platform or method rely on finding hosts vulnerable to only this particular exploit. For a worm to support multiple vectors of compromise or various target platforms of a similar type, it must be large. This extra weight can slow down any one instance of a worm attack or, in a macroscopic view, more quickly clog the network.

Other attacks include session hijacking and credential theft (such as passwords and cookies) attacks. Here the attack does not involve any escalation of privileges, but does assist the worm in gaining access to additional systems.

These attack elements are also most often used in intrusion detection signature generation. Since the attack is executed between two hosts and over the network, it is visible to monitoring systems. This provides the most accessible wide area monitoring of the network for the presence of an active worm. However, it requires a signature of the attack to trigger an alert. Furthermore, passive intrusion detection systems cannot stop the worm, and the administrator is alerted to the presence of the worm only as it gains another host.

2.4.3 Communication Component

Because the nodes of the worm network reside on different systems, they must have some form of communications. This allows for the transfer of information. For reconnaissance information, network vulnerability and mapping information must be distributed to nodes which can use this information in an attack. For commands, they

must be able to send requests to the action nodes, to initiate a scan, an attack, or other activities.

Communications channels are usually hidden by the worm using the same techniques hackers use when they have manually compromised a machine, such as rootkits.

They typically include network clients to various services or transport mechanisms such as ICMP packets.

2.4.4 Command Component

A system of nodes is only worthwhile if they are able to be controlled by some means. This can either be an interactive control mechanism, where a user is able to direct actions of the node, or through some channel for the system itself to control a node.

In this part, worm networks are akin to a network of systems in a distributed denial of service (DDoS) ring. Usually these nodes have two types of command interfaces, one interactive, where a remote control shell is obtained, and one that is automatic, where the node is in control of some master.

Traditionally the attacker has placed some form of a backdoor entry into the system. On UNIX systems this can include a trojanned login daemon which is configured to accept a special passphrase that grants administrative access. On desktop systems, such as Windows PC's and Macintosh systems, this can be a simple `Trojan Horse' program, which listens on a network socket for commands.

The objective is quite simple, to allow for the system itself, using a master-slave node relationship, to have an extended reach or capability, or more simply to allow an intruder unfettered access to the system to manually command it. In one form or another, most worm systems have some form of a command interface. This prevents

the worm system from lacking any structure, so that it may be used in a controlled fashion. Commands such as file uploads or downloads, status reports, or actions such as “attack this target” have all been possible through this interface.

The command interface can be connected to by another node of the worm network, such as the parent or a child, or manually by an attacker. The command interface is tightly coupled to the communications channels, but is separate as different communications mechanisms can be used to contact the same command interface.

2.4.5 Intelligence Component

The worm system maintains a record of its members and their locations in some form or another. This is useful so that the nodes can brough together for some additional action. Control, through the command interface, can be taken by a person or by another node of the worm system. However, this requires knowing how to contact the nodes, which requires knowing their network locations.

The simplest fashion for this to occur is via an update message from a newly acquired node. The new member's address, and any pertinent information, and sent to a some facility and recorded.

This information can manifest itself in intangible ways, as well. For example, many Windows worms use their presence on a network chat room, such as IRC, an an intelligence mechanism.

They arrive once infected, announce their location and any passphrases needed to gain entry, and simply sit and wait. In this fashion, the worm network knows about its members, their location and potentially any capabilities they possess.

2.4.6 Assembly of The Components

Figure 2.2 shows the pieces as they would be assembled in a full worm. For example, the reconnaissance component sends information to the attack module about where to launch an attack. It also sends this information to an intelligence database, possibly using the communication interface. This communications interface is also used to interface to the command module, calling for an attack or the use of the other capabilities against a target.

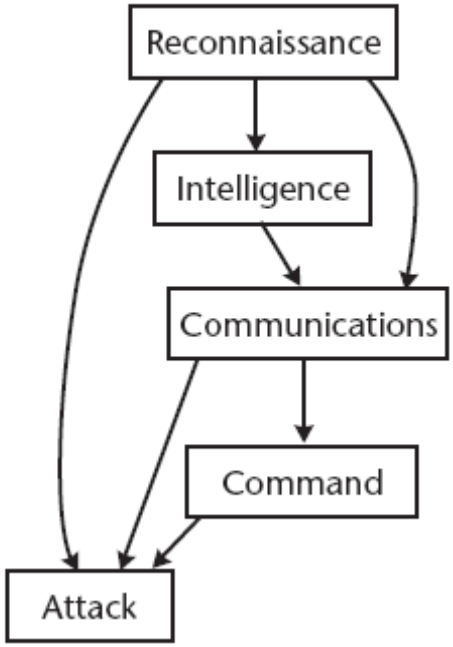

Figure 2.2 Assembly of the worm's components

2.5 Worm Traffic Patterns

The worm network actively seeks new hosts to attack and add to the collection nodes in the network. As it finds hosts and attacks them, the worm network grows exponentially. This growth pattern mimics patterns seen for communities occurring naturally,such as bacteria and weeds.

Worm infections can grow in an exponential pattern, rapidly at first and then slowing as a plateau value is reached. This is a typical kinetic model that can be described by a first-order equation:

$$N\,da = (Na)K(1-a)\,dt$$

It can then be rewritten in the form of a differential equation:

$$\frac{da}{dt} = Ka(1 - a)$$

This describes the random constant spread rate of the worm. Solving the differential equation yields

$$a = e^{K(t-\tau)} / (1 + e^{K(t-\tau)})$$

where *a* is the proportion of vulnerable machines that have been compromised, *t* is the time, *K* is an initial compromise rate, and *T* is the constant time at which the growth began. Rate *K* must be scaled to account for machines that have already been infected, yielding $e^{K(t-\tau)}$

While more complicated models can be derived, most network worms will follow this trend. We can use this model to obtain a measure of the growth rate of the worm. Some worms, such as Nimda and Code Red, have a very high rate constant *k* meaning that they are able to compromise many hosts per unit of time. Other worms, such as Bugbear and SQL Snake, are much slower, represented in the smaller rate constants for growth.

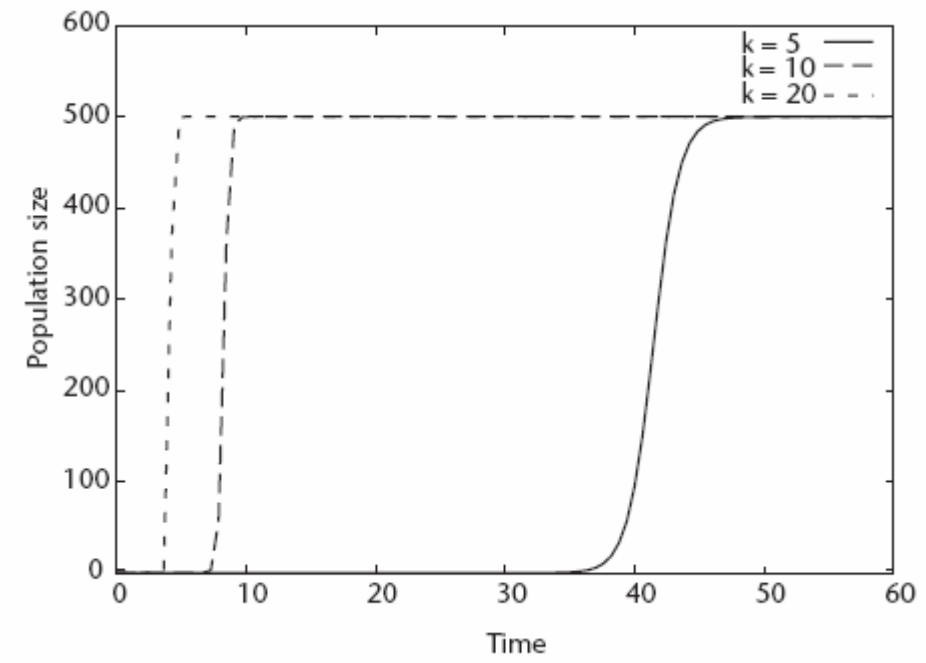

Figure 2.3 Worm Traffic Pattern

Figure 2.3 shows a simple graph of using several values of k. The equation shown in this figure is the sigmoidal growth phase of a logistic growth curve. The initial phase of exponential growth and the long linearphase as the worm spread scan be observed. As the worm saturates its vulnerable population and the network, its growth slows and it approaches a plateau value.

These equations are highly idealized, because the value of N is assumed to be fixed. This assumes that all hosts that are connected at the outset of the worm attack will remain attached to the network. This constancy assumes that hosts will remain vulnerable and patches will not be applied. Furthermore, the model assumes a similar amount of bandwidth between hosts which also remains constant during the worm's life cycle. In the real world, not all hosts have the same amount of connectivity, and bandwidth is quickly consumed by the worm network as it grows to fill the space. Despite this, these equations provide a good representation of the observed data for a reasonably fast moving worm.

CHAPTER THREE
WORM SCAN TECHNIQUES

In order to propagate itself in the Internet, a worm needs to find vulnerable machines and then infect them. To find vulnerable machines, a worm can either simply scan the entire IPv4 address space randomly, or may perform various strategies to scan the entire or partial IPv4 address space to find targeted hosts. In this section, we discuss various scan strategies and analyze their spreading speed.

3.1 Random Scan

A worm randomly searches the entire IPv4 address space, which contains 2^{32} possible IP addresses, to find vulnerable machines. We call such scan method random scan. There are two existing models to simulate the random scan worm propagation. One is the epidemiological model proposed by Kephart and the other is AAWP model proposed by Chen. (Xia, Vangala, Wu, & Gao, 2006)

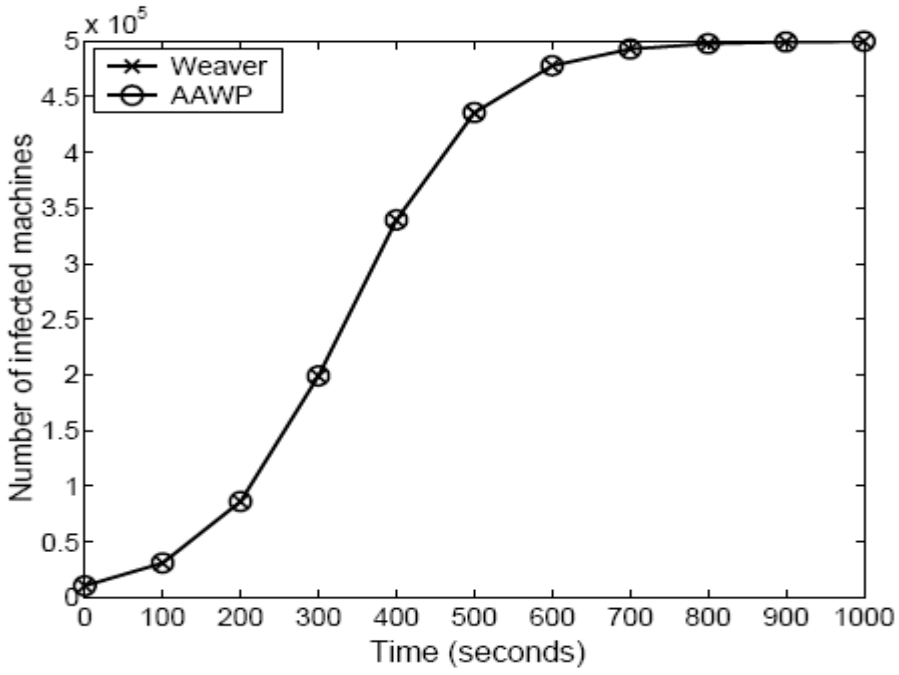

Figure 3.1 Comparison between AAWP model and Weaver's simulator

Due to the equivalence of these two models as shown in Figure 3.1, we adopt the AAWP model in this thesis. Based on the AAWP model, the spread of worm is characterized as follows:

$$n_{i+1} = n_i + [N - n_i][1 - (1 - 1/\Omega)^{sn_i}] \qquad (1)$$

N: the total number of vulnerable machines in the Internet

Ω: number of the addresses that a worm performs random scan

s: the scan rate (the number of scan packets sent out by an infected machine per time tick)

n_i: the number of infected machines up to time tick *i*.

In Equation (1), the first term on the right hand side denotes the number of infected machines alive at the end of time tick *i*. The term, $N - n_i$, denotes the number of vulnerable machines not infected by time tick *i*. The remaining term, $(1 - 1/\Omega)^{sn_i}$, is the probability that an uninfected machine will be infected at the end of time tick $i + 1$. We do not consider the death rate due to computer crash and patching rate due to maintenance here. Code Red is a typical example of random scan worms.

3.2 Selective Random Scan

Instead of scanning the entire IPv4 address space blindly, a worm can scan the partial IPv4 address space that is more likely to be used in the Internet. This will help the worm spread faster by reducing the waste of time on scanning unallocated addresses. The selected address list can be obtained from other resources such as IANA's IPv4 address allocation map. Such scan technique with target selection is called selective random scan. The Slapper worm has used this scan technique to spread rapidly. However, worms using the selective random scan need to carry information about the selected target addresses. Carrying such information enlarges the worm's code size and slows down the spreading and infection processes. This

information can be hundreds of bytes long and therefore, may not provide much advantage over the random scan.

The SQL Snake worm array is shown next. This array was used to generate a biased list of addresses for the worm to probe and attack:

sdataip = new Array(216, 64, 211, 209, 210, 212, 206, 61, 63, 202, 208, 24, 207, 204, 203, 66, 65, 213, 12, 192, 194, 195, 198, 193, 217, 129, 140, 142, 148, 128, 196, 200, 130, 146, 160, 164, 170, 199, 205, 43, 62, 131, 144, 151, 152, 168, 218, 4, 38, 67, 90, 132, 134, 150, 156, 163, 166, 169);

This array represents the first octet in the network address to scan, and it has been chosen because these networks lie in the space between class A (0/8 through 126/8) and class C networks (ending at 223.255.255.255), inclusive. This array is then used to build a second array with a nonrandom frequency of these numbers. The second octet is a random number chosen from between 1 and 254, with the scanner operating on more than 65,000 hosts (in a /16 network block) sequentially.

However, not all of the address space that can be allocated and used in this range is actually used. For various reasons, many networks are empty and have few or no hosts assigned to them. If the worm were to attempt to probe or scan these networks, the rate of scanning would not be bound by the number of hosts to scan, but instead by the timeout values for the inability to connect. When a network range is scanned, the number of addresses attempted can grow to the tens of thousands, causing a significant delay in the worm's overall spread.

To compare the spreading speed between random scan worms and selective random scan worms, we do not consider such additional payload information on selected target addresses. Figure 3.2 compares the spreading speed of worms that use random scan and selective random scan techniques.

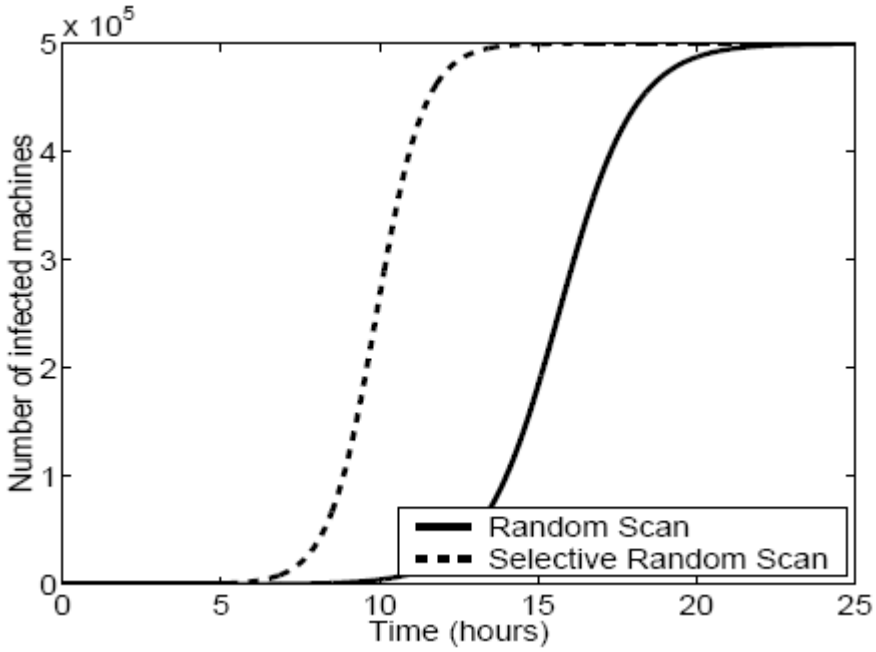

Figure 3.2 Spreading speed of random scan and selective random

The parameters are chosen as the same for both the random scan and the selective random scan. The total number of vulnerable machines N is 500,000; the scan rate s is 2 scans/second. The random scan worms use the entire IPv4 address space which has about $2^{32} \approx 4.3 \times 10^9$ addresses. The selective random scan worms use only 162 /8 address blocks which contain about 2.7×10^9 addresses. Figure 3.2 demonstrates that worm can spread much faster using a selective address pool than using the entire IPv4 address space.

3.3 Hit-list Scan

Nicholas Weaver described a new type of worm and he dubbed it the Warhol worm. We analyzed this worm in Chapter 4. The biggest jump in design in a Warhol worm is the use of a hit list to scan and attack. This hit list contains the addresses and information of nodes vulnerable to the worm's attacks. This list is generated from scans made before unleashing the worm. For example, an attacker would scan the Internet to find 50,000 hosts vulnerable to a particular Web server exploit.

This list is carried by the worm as it progresses, and is used to direct its attack. When a node is attacked and compromised, the hit list splits in half and one-half remains with the parent node and the other half goes to the child node. This mechanism continues and the worm's efficiency improves with every permutation.

The exact speed with which near complete infection of the Internet would occur is debatable. Weaver's estimates for probe size, infection binary size, the speed with which this infection can be transferred between parent and child node, and network bandwidth are all speculative. However, there is no doubt that this infection design is highly effective.

While effective, this mechanism has several drawbacks. First, the necessary scans are likely to be noticed. While widespread vulnerability scanning has become commonplace on the Internet and is possibly accepted as background noise by some, widespread scanning for the same vulnerability still generates enough traffic in the monitoring community to raise some flags. Second, the network bandwidth consumed by a fast moving worm is likely to choke itself off of the network. As more worms become active, network connections fill, restricting the ability for the worm to move as efficiently. However, if the hit list were to be sorted hierarchically, so that larger bandwidth networks were hit first and the children nodes were within those networks, concerns about bandwidth could be minimized.

3.4 Routable Scan

The fourth type of network scanning that worms perform is typically called routable scan. In order to further reduce scanning address space, a worm may avoid scanning the address space that could not be routed in the Internet. It means that a worm can obtain all routable addresses as scan targets in order to spread fast and effectively. However, this worm has to carry a database of routable IP addresses in its code. The size of this database will affect the propagation speed. A database of larger size will lead to a longer infection time, resulting in slower worm propagation.

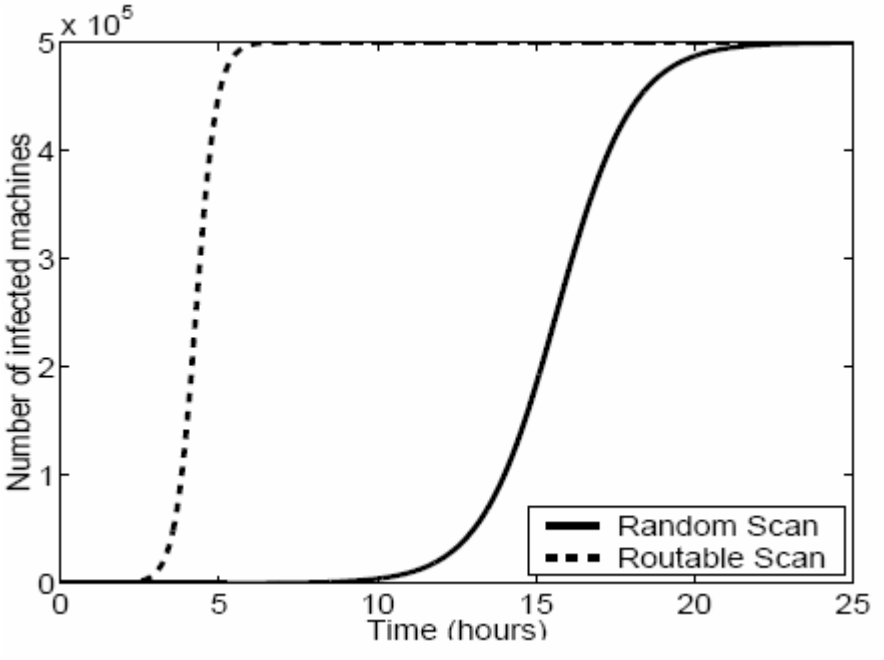

Figure 3.3 Spreading speed of random scan and routable scan

The worm that employs routable scan needs to scan only 10^9 IP addresses instead of 2^{32} addresses, which is four-fold smaller. Hence, routable scan worm has a scanning space of size $\Omega \approx 10^9$. For other parameters, we use the same settings as random scan. Figure 3.3 shows the spreading speed of routable scan and random scan. We find that if random scan worm needs to spend about 24 hours to infect almost whole vulnerable machines, the routable scan worm only needs to spend about 7 hours to do it. Clearly, routable scan strategy greatly increases the worm spreading speed.

3.5 Scanning Constraints

Some interesting problems arise for the worms that try to spread fast. Their ability to scan the network are usually constrained by bandwidth limits or latency limits:

Bandwidth Limited: Worms such as the Slammer that use UDP to spread face this constraint. Since there is no connection establishment overhead, the worm can

just keep transmitting packets into the network without expecting an acknowledgement from the victim. Modern servers are able to transmit data at more than a hundred Mbps rate.

Let us perform some simple calculations. Consider a Slammer-like worm that uses a single UDP packet of 400 bytes to spread. It resides on an infected machine with a 100Mbps link to the Internet. Assuming the network is otherwise quiescent, the total capacity of the link divided by the number of bits in the worm packet gives the scanning rate. Initially, this is $100x\ 10^6 / (400 \times 8) \approx 30,000$ scans per second.

But the network soon saturates with traffic from several copies of the same worm from different victims or the same victim, each of which generates data at its maximum possible rate. As a result, the spread of the worm is constrained. Thus a worm becomes a bandwidth limited worm.

Latency Limited: A worm that uses TCP to spread is constrained by latency. These kind of worms need to transmit a TCP-SYN packet and wait for a response to establish a connection or timeout. The worm is not able to do anything during this waiting time. In effect, this is lost time for the worm. To compensate a worm can invoke a sufficiently large number of threads such that the CPU is kept busy always. However, in practice, context switch overhead is significant and there are insufficient resources to create enough threads to counteract the network delays. Hence the worm quickly reaches terminal spread speed.

3.6 Summary

We described various scanning techniques that are employed by worms. Hit-list scanning seems to be the most effective to spread a worm in the smallest amount of time possible. This chapter explained the bandwidth and latency constraints faced by high-speed worms.

CHAPTER FOUR
FUTURE WORMS

4.1 Intelligent Worms

A Polish security researcher, Michal Zalewski, released a paper describing a design for a smarter worm. Entitled “I Don’t Think I Really Love You, or Writing Internet Worms for Fun and Profit,” the ideas in Zalewski’s paper, provide a compelling vision of worms. Many of the techniques he describes have been incorporated into tools used by attackers during unautomated attacks.

The analysis begins with the idea that the Melissa virus was not as devastating as it could have been. After all, the virus used a simple engine to spread, always executed using the same mechanism, and thus had a static signature. Many mechanisms exist to detect and disable such worms and viruses, as evidenced by the large antivirus industry.

Zalewski and other hackers introduces a project which name is Samhain. Intending to design a more effective Internet worm, they listed seven requirements and guidelines for their system:

- In order to achieve the largest possible dispersal, the maximum number of target hosts must be used. For this, it should be portable. It means that it should be compatible with all of the possible operating systems and the hardware arhitectures.

- Invisibility from detection. Once found, the worm instance can be killed on the host, disrupting the worm network.

- Independence from manual intervention. The worm must not only spread automatically but also adapt to its network.

- The worm should be able to learn new techniques. Its database of exploits should be able to be updated.

- Integrity of the worm host must be preserved. The instance of the worm's executables should avoid analysis by outsiders.

- Avoid the use of static signatures. By using polymorphism, the worm can avoid detection methods that rely on signature-based methods.

- Overall worm net usability. The network created by the worm should be able to be focused to achieve a specific task. (Zalewski, 2000)

From these seven requirements came an implementation in pieces that, when assembled, formed a worm system.

By far one of the most challenging things the Samhain worm would have to achieve is portability. Source code that is intentionally written and extensively tested has difficulty in doing this correctly under all circumstances. Because of their "fire and forget" nature, worms do not have the luxury of debugging in the field.

The Samhain worm attempts to achieve this by relying as little as possible on architectural specifics. This includes favoring interpreted languages over compiled languages when possible and using generic coding techniques that attempt to use the most common factors available. While not all languages are present between UNIX and Windows, for example, enough functionality is possible. Furthermore, with additional features within the worm, once built on one system, a worm component can easily be requested and installed by any node.

The overriding philosophy for this design decision is that for a worm to be truly disruptive and effective, it has to affect as many hosts on the network as possible. When limited to, say, Linux or Microsoft Windows, only a part of the total possible space is explored by the worm. Enough vulnerabilities exist between these major

hosts that they can be used to target nearly all hosts on the Internet, creating a large-scale disruption and problem worse than any seen previously.

Once inside the child host, Zalewski notes, the worm needs to attempt some form of invisibility. This sort of hiding is desirable because the worm will want to survive on the host for as long as possible. A longer lived worm can find more hosts and attack more targets, increasing the worm's spread. This invisibility is necessary mainly to hide from system administrators or investigators.

The worm can utilize either of two different main mechanisms for hiding on a system. The first method does not rely on privileged execution, but instead hides in the open. Because most systems are busy, the worm simply adopts the name of a process on the system. This might include processes that have multiple instances of themselves running, such as "httpd." In doing so, an administrator would most likely skip right over the worm process, not noticing its presence.

The second method relies on the worm processes having elevated privileges on the target system. In this case, the new processes can insert kernel modules that can redirect system calls. These altered system parameters can be used to hide worm files and processes on a system. Additionally, altered binaries on a host that simply do not report the worm's processes and activities can also be inserted into the system.

The next design requirement for the worm that Zalewski described is the ability to operate independently. While worms do replicate and work automatically, in this scenario this requirement is more significant. Because the worm has to target multiple host types and adapt to the local environment in order to hide itself, the worm's intelligence must be beyond that of most worms.

To accomplish this, Zalewski proposes that a database of known attack methods and exploits be made available to the worm. For example, a worm encounters a host running a particular server version and launches one of the attacks it knows about. The attacks focus on platform independence, such as file system races and

configuration errors, rather than architecturedependent attacks such as buffer overflows and signal races. This gives the worm the platform independence specified by the first design goal. Known attacks would be sorted by their effectiveness with the list passed to the child nodes. The executables for the worm could also be distributed from other nodes in the system. For example, when a node is attacked but it lacks any means to compile the executable, or the parent node is missing the binaries for the child node, they are simply retrieved from another node that already has these pieces.

An additional design goal for the worm described by Zalewski is the ability to update to learn new attack methods. To do this, the worm nodes would establish a network, much like those discussed in earlier chapters. From one or more central sites the worm network would receive updates to this database of attack methods, allowing it to adapt to new methods and capabilities, improving its overall life span.

In the paper, Zalewski revives an older method for finding new hosts to attack—observing the host system's behaviors. The Morris worm found new victims to attack by investigating the list of trusted hosts. The worm designed by Zalewski would observe the servers to which the worm node normally connects (from its users) and attack them. The primary benefit of this is the ability to hide in the normal traffic for the host, and also being able to observe some facets of the target server before an attack is launched.

Two additional methods are described to achieve the design goal of maintaining the integrity of the worm node. The first is to hide from any monitoring and investigation by detaching from process tracing methods. The worm simply detects the attachment of a process tracing facility and disables it while continuing its execution. This hampers investigation and, sometimes, sandboxing of the executable.

Secondly, the use of cryptographically signed updates means that an adversary would encounter difficulty in injecting updates that would compromise the worm node. These would include poison or empty updates that would effectively disable

the worm node. These sorts of attacks are described in more detail in Chapter 7. By ensuring that only trusted updates are inserted into the system, the overall integrity of the worm node can be maintained.

One of the most commonly used detection methods is a static signature. As described in Chapter 6, these can include log signatures, network attack signatures, or file signatures. To bypass these detection methods, some viruses employ a strategy termed polymorphism. The worm described by Zalewski also uses such a principle.

The fundamental method used by malicious polymorphic code is simple encryption, with decryption occurring at run time. By using a random key each time, the encrypted file has a different signature. In this way, the malicious payload is able to escape signature detection.

The worm designer's final goal is to make it usable. The worm must do more than simply spread as far and as wide as possible. It must be usable for some higher purpose. While it may be tempting to develop the worm initially with this ultimate use in mind, one strategy outlined by Zalewski was to have the worm spread to its final destinations and then use the update capabilities to begin its mission. This purpose could include the retrieval of sensitive files, destruction of data, or network disruption.

It is interesting to note that some of the adaptations have been used by worms since Zalewski's paper. The Adore worm, for example, used kernel modules to hide its presence on a host. Variants of the Slapper worm would use the process name "httpd" to hide in with other Web server daemon processes it used to gain entry to the system. In this latter case, the worm process was distinguished by its lack of options similar to the normal web server daemon processes.

Furthermore, the use of multiple forking to evade process tracing has been found in the wild. While this makes investigation and sandboxing difficult, it is not impossible. An additional design goal that has been seen in the wild for many years

is the use of polymorphism. This design premise was borrowed from the world of computer viruses, where polymorphic viruses have been found in the field for several years. They present a significant challenge to detection and investigation, but not a total one.

Two other design ideas developed by Zalewski have also been seen in worms found in the wild. Updatable worms have been found, namely, the Windows Leaves worm. Using a modular architecture, updates can be distributed on the Internet and the worm can retrieve them. Second, multiple attack vectors are not uncommon for worms to use, though none have presented a sophisticated system for sorting their attack mechanisms or attempted to use platform-independent methods.

4.2 Modular and Upgradable Worms

Nazario describes worms on the basis of the five components outlined in Chapter 2: reconnaissance actions, attack capabilities, a command interface, communication mechanisms, and an intelligence system. These components were then identified in three existing worms found in the wild to illustrate how they can be combined into a larger functional worm.

In the analysis of the potential future of Internet worms, there are several problems with the design and implementation of current worms. These are necessary to assess a likely future for worm designs. The first limitation is in the worm's capabilities. These limitations are found in all aspects of the worm's behavior, including its attack and reconnaissance actions. For network-based intrusion detection, the signatures of the remote attacks can be quickly identified and associated with the spread of the worm. This reconnaissance traffic can also be associated with the worm, identifying the source nodes as compromised.

The second major problem with worms, they have a finite set of known attacks they can use. They have a limited pool of potential targets. It means that limited lifespan for the worms.

Finally, a worm that does utilize a database of affected hosts typically uses a central intelligence database. The central location means that the worm is open to full investigation. An attacker or investigator can easily enumerate all of the worm nodes and either overtake them or clean them up. Alternatively, an attacker or investigator can move to knock out the location, either by firewalling the destination at the potential source networks or at the incoming transport mechanism. Examples of this include an e-mail inbox, a channel in a network chat system, or a machine to which it is connected directly. By blocking the delivery of the updates from the new nodes to the central source, no additional information is gathered about the worm.

4.3 Warhol Worms

Nicholas Weaver proposed a new model for worm spread. (Weaver, 2001) This model was dubbed the Warhol worm. A Warhol worm is an extremely rapidly propagating computer worm that spreads as fast as physically possible, infecting all vulnerable machines on the entire Internet in 15 minutes or less. The term is based on Andy Warhol's remark that "In the future, everyone will have 15 minutes of fame". A worm author could collect a list of 10,000 to 50,000 potentially vulnerable machines, ideally ones with good network connections. When released onto a machine on this hit-list, the worm begins infecting hosts on the list. When it infects a machine, it divides the hit-list into half, communicating one half to the recipient worm and keeping the other half. The creation of the hit list can be readily accomplished using existing Internet mechanisms. These mechanisms were enumerated by Staniford:

- **Single-source scans:** Utilizing a single, well-connected host, the entire Internet space can be scanned for known vulnerabilities, and these data organized for retrieval later. The speed of any scan will depend on the

bandwidth available to the source, the nature of the scanning tool (such as the number of threads available to it), and the data gathered. A simple TCP connect scan, for example, will consume fewer resources than a service analysis or even a banner grab.

- **Distributed source scans:** Utilizing the same type of network used by DDoS systems, multiple sources can be used to scan the Internet for vulnerabilities. The distributed nature of the scan will improve efficiency as well as mask the scale of the scan, because the aggregate bandwidth will scale with the network. In either case, single host or distributed, large-scale scans no longer receive much attention from the Internet community due to their pervasiveness. Furthermore, if speed is not a concern, the scan can hide below the threshhold of the Internet security community at large.

- **DNS searches:** Some types of servers are so well advertised by the DNS system, such as name servers (using NS records) and mail servers (using MX records) that they can be enumerated via a simple DNS query.

- **Public survey projects:** Web servers are well categorized by their server address, type, features, and usually the banner by projects such as the Netcraft survey. Using this database, gathered by others for use in a respected project, could save the attackers time and make building a large hit list a relatively easy task.

- **Passive data gathering:** Many vulnerable systems advertise themselves on the Internet without any work required by an attacker. These include peer-to-peer networks as well as nodes affected by other worms, announced as they scan for new victims. Well-connected sites could gather lists of hundreds of thousands of vulnerable hosts due to these sorts of actions. (Staniford, 2002)

4.4 Flash Worms

An improvised Warhol strategy would be to program the worm to divide the hit-list into `n' small blocks instead of 2 huge ones, infect an high-bandwidth address in each block and pass on to the child worm the corresponding block. This process would be repeated by each child worm.

A threaded worm could start infecting hosts before it had received the full host list from its parent to work on. This maximizes the parallelism of the process, and the child worm can also start looking for multiple children in parallel.

4.5 Polymorphic Worms

Any worm that changes its form or functionality as it propagates from machine to machine can be called a Polymorphic Worm.

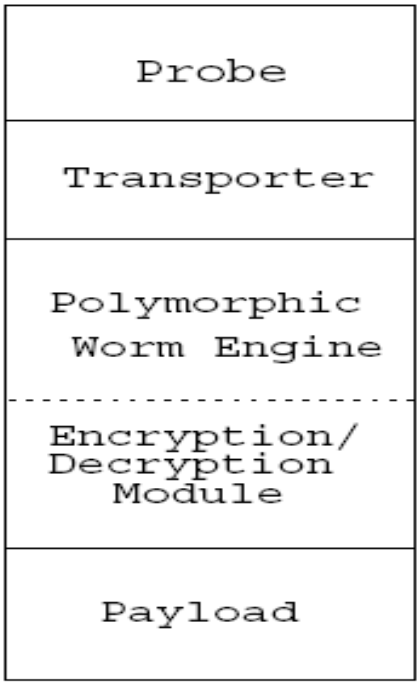

Figure 4.1 Components of a polymorphic worm

The Worm Engine contains an additional module called the Encryption Engine or the Mutation Engine that is responsible to change the form or look of the worm when it moves from one host to another. Figure 4.2 shows the typical polymorphic worm structure. (Lee, 2006)

Figure 4.2 Typical polymorphic worm structure

The encryption engine could be something very simple, for example, that just inserts some no-ops into the worm code to evade systems that use signatures for detection or could be something as sophisticated as encrypting the entire worm including itself using a random seed for every hop so as to evade detection during transit. It could even reprogram itself to exploit different vulnerabilities on depending on the host operating system or other such parameters.

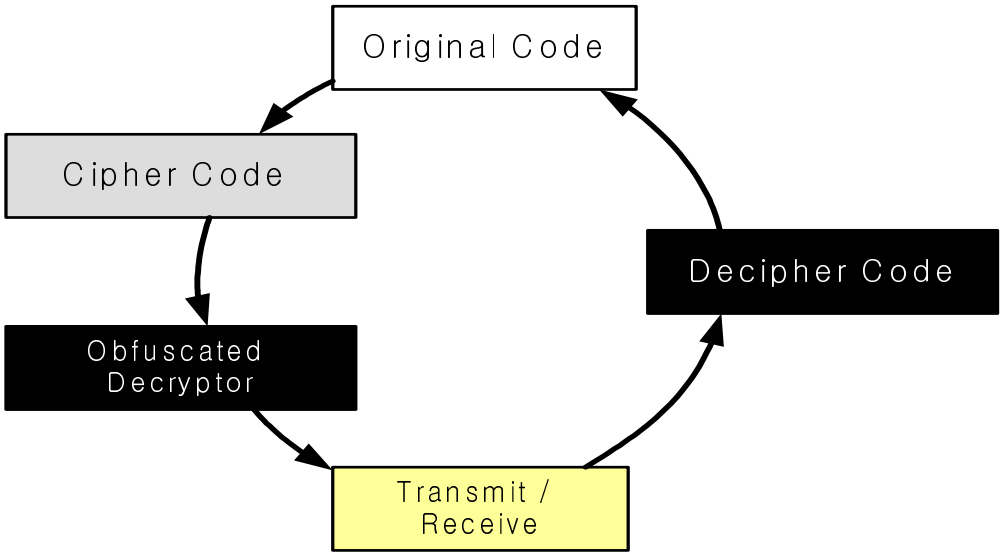

Figure 4.3 Polymorphic worm cycle

4.6 Miscellaneous Worms and Viruses

Viruses are a different class of programs that need human intervention to spread from one host to another. Early viruses attached themselves to other popular programs and spread when people exchanged or copied these programs from one machine to another through floppy disks or other manual means. Later viruses attached themselves to e-mails that a user sent out. Some viruses automatically sent e-mails to addresses in the address book on the infected system. Since these didn't require human intervention, these were called e-mail worms.

Some of the second generation viruses include:

- **Retro Viruses:** Viruses that fight back against anti-virus tools by deleting virus definition tables, memory resident scanners, etc., These viruses could be used as pilot viruses to a malicious worm that would come by later. This way, for example, the worm following the Retro virus would not be detected by IDSs.

- **Stubborn Viruses:** These can prevent themselves from being unloaded from an infected Windows system. However, techniques that could achieve this have not been fully explored.

- **Wireless Viruses:** These are viruses that infect wireless devices by making use of their ability to exchange applications \through the air".

- **Coffee Shop viruses:** These viruses attach themselves to computers that are plugged into the network of some chains of coffee shops. They don't try to hop from one machine to another. They just wait at the coffee shop for a vulnerable host to come by and connect to that network.

CHAPTER FIVE
THE LIFE CYCLE MODEL OF WORM DEFENSE

The problem of worm defense can be broken down into various stages and fit into a life-cycle model. This is a problem where the defenders are perpetually in a race against unknown and unseen opponents. Hence the model is cyclic. Figure 5.1 gives a diagrammatic representation of the life cycle. (Cheetancheri, 2004)

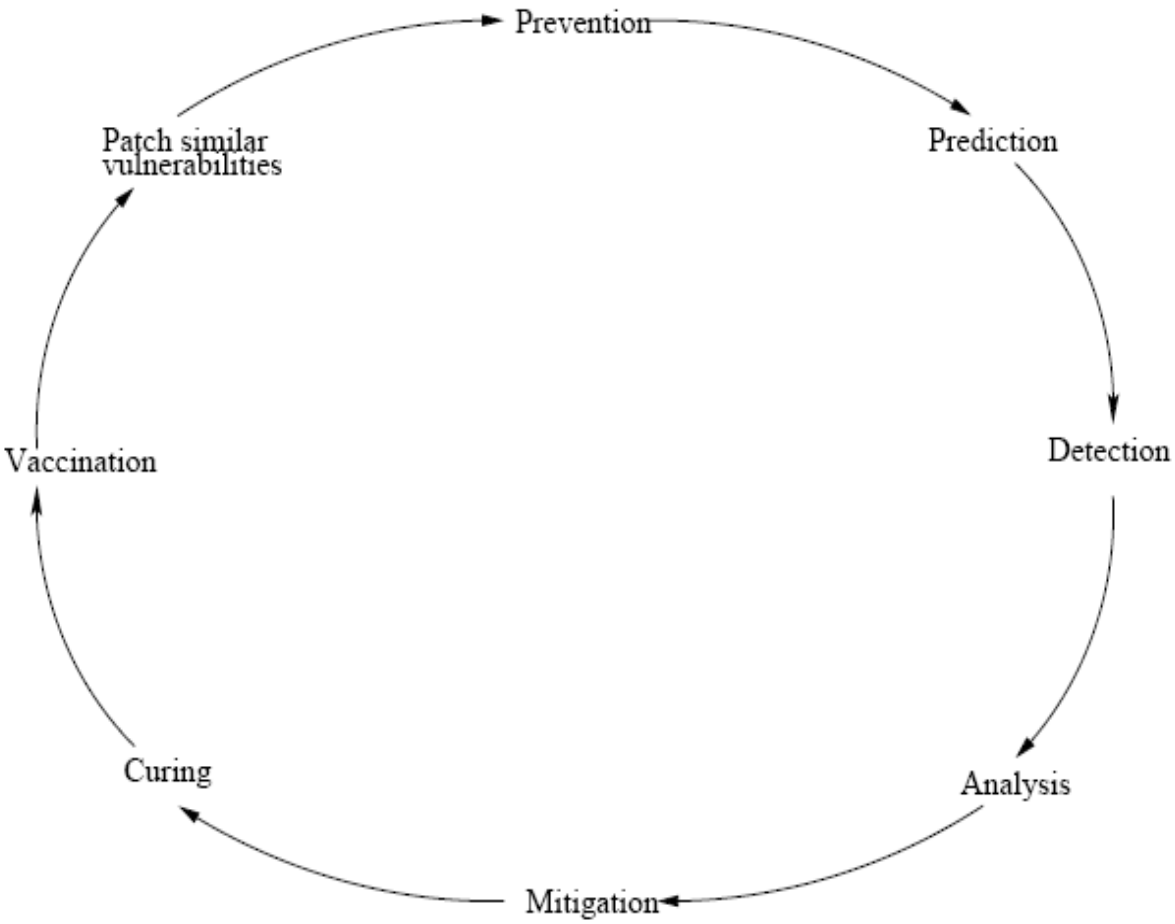

Figure 5.1 The various stages of the life cycle of worm defense

5.1 Prevention

The best way to stop a worm is to prevent its incursion into a particular site. Prevention is better than cure. Once a suspicious activity is discovered, fix holes that are being exploited and distribute the patch widely. This step applies even when there is no worm spreading. Only constant watch and vigil can prevent worms. However, it is next to impossible to have no holes at all points of time. But an earnest approach to plug holes identified by advisories from trusted security sources is a good step in that direction.

5.2 Prediction

The observation of suspicious and similar behaviour at various places is a good indication of the genesis of a worm. This needs quite an amount of co-operation and correlation amongst various sensors. The "Group Based Model" in Chapter 8, does this as a part of its mitigation strategy.

5.3 Detection

Detection of a worm is either an easy or hard job depending on the kind of worm we are dealing with. Fast spreading worms are easy to detect. They show themselves through various symptoms on the network and on the individual hosts that they infect. The most obvious symptom usually is abnormally excessive cpu load at the host level and bandwidth saturation at the network level. Fast spreading worms have the following characters:

- They write heavily to the network.

- They copy themselves frequently. Frequent fork()ing is a symptom of this behaviour.

- They scan the network heavily, usually looking at a single port.

- They open up a lot of TCP connections.

5.4 Analysis

Once we detect a worm in action the immediate analysis should focus on identifying the signature so that we can try to stop traffic that match the signature. In the presence of a very fast worm, the solution might be to stop all traffic. But normal traffic should be allowed to resume as soon as possible. Otherwise, the cost of traffic locking could be more than the damages that the worm could cause. The later analysis, after the worm is defeated, should focus on identifying the intent, means and damage caused, to help cure the infected hosts and take steps so that it doesn't re-surge, as does the Code Red worm that keeps re-surging monthly. For example, Nimda is still not fully understood.

5.5 Mitigation and Response Strategies

We cannot stop a fast moving worm at all places as soon as it is discovered at one place. Even though we fix one host, there are already several others infected which continue to spread the disease to other susceptible hosts. This doesn't mean infected hosts should not be fixed to stop the worm. They should be fixed. But before that, the situation warrants a different approach to arrest the spread: at least, slow down the worm and mitigate the disaster.

Some of the hypothesized high speed worms like Flash worms should be responded to automatically. These cannot be managed by human intervention. All damage would be done even before we could react. To respond to such a worm with human speed is simply not possible.

5.6 Curing The Infected Hosts

Even though we could reboot an infected machine to kill a worm instance, this machine will be re-infected sooner or later unless the vulnerability exploited is fixed. So, the most logical step after a worm attack is to fix the vulnerabilities that were exploited by the worm. This involves using the results of the analysis and acting upon them. Closing all relevant back doors and fixing the bugs exploited by the worm are only a few of the pertinent activities in this step.

5.7 Vaccinating Uninfected Hosts

Even uninfected hosts should be patched up. Mitigating the spread of worm involves turning on filter rules at fire-walls and patching. Filters decrease performance. So, the filters have to be turned off eventually. Once the firewall rules are turned off, there are chances of reinfection un-cured hosts.

5.8 Patching Similar Vulnerabilities

One of the important lessons learnt from any worm incident should be an awareness of the vulnerability exploited. Once this is learnt, similar vulnerabilities should be sought out and fixed. For example, the Morris worm showed that the sendmail program had a bad default Debug option. Once this was realized all other programs should be checked for similar oversights. Fixing such vulnerabilities should be an on-going process all the time. This naturally blends into the first step of prevention.

In an ideal situation we should be spending the most time in the prevention phase of the worm life cycle. That would mean we are maintaining a more secure Internet.

CHAPTER SIX
WORM DETECTION

This part builds on this information in an attempt to illustrate three methods of detecting worms. These methods are traffic analysis, the use of honeypots and dark network monitors, and the employment of signature-based detection systems. These methods form the core of detecting both hackers and worms.

The goal of our detection strategies is to detect nearly any type of worm with as little effort as possible. To do this, we will focus on the features common to most worm types and build strategies to detect these characteristics. While no single methods work for all worm types, a combination of efforts can provide more complete coverage.

6.1 Traffic Analysis

Briefly, traffic analysis is the act of analyzing the network's communications and the patterns inherent in it. The characteristics of the traffic that are studied can include the protocols, the ports used in the connections, the success and failures of connections, the peers of the communications, and the volume of traffic over time and per host. All of these characteristics can be combined to develop a picture of the network under normal circumstances and also used to identify the presence of a worm.

With respect to analyzing traffic to monitor for worms, we are interested in monitoring three major features. These three characteristics are common to nearly all worm scenarios and hence of interest to us. Furthermore, the ease of monitoring these features makes them especially attractive.

The first facet of a network we should monitor to detect the presence and activity of worms is the volume of traffic. Most worm models use a logistical growth model,

meaning the number of hosts grows exponentially in the initial phases. As hosts are brought on-line into the worm network, they perform scans and attacks. Their combine traffic leads to an increase in the volume of traffic seen over time. This is best monitored at a network connection point, such as a router or a firewall, and not necessarily an edge node.

The second feature of the network's traffic we are interested in monitoring in the number of type of scans occurring. Most worms use active measures to identify new targets to attack, using scans of hosts and networks to find suitable targets to attack. These scans can be tracked using monitors and measurement tools and analyzed to reveal worm hosts either on the local network or attacking the local network from remote sites.

The third feature we are interested in for the purposes of traffic analysis is the change in traffic patterns when a host is part of a worm network. Each host on a network has a well-defined set of characteristics in its traffic that typically change after compromise by a worm. By monitoring hosts and their traffic patterns, the presence of a worm on the local network can be identified.

The use of traffic analysis to detect the behavior of network worms is a powerful technique due to its generality. Larger network events are typically monitored and analyzed to search for trends. While not all of the observations that are associated with worms are unique identifiers of worm activity, when combined with other analysis methods a more detailed picture emerges. The main drawbacks to traffic analysis, including a large data set and a number of observation points, make it a challenging endeavor.

6.1.1 Strengths of Traffic Analysis

Traffic analysis, which focuses on general aspects of the network and the trends therein, has several advantages over specific detection methods and black hole and

honeypot monitors. The first is that it works for almost all worm types, specifically for worms that use active target identification methods and exponential growth models. Scans can be measured and tracked as a general phenomenon, and the exponential growth of the overall volume of the network can also be observed.

Secondly, signature detection fails for worms that use any variety of dynamic methods. These can include modules that can be updated to accommodate new attack methods or scan engines, or worms that behave in a manner similar to polymorphic viruses. Furthermore, signature detection at the network level will fail for worms that use either encoded or polymorphic attack vectors. By observing the traffic characteristics generally, the presence of the worm can be identified.

6.1.2 Weaknesses of Traffic Analysis

The analysis of network traffic to identify the presence of a network worm has several drawbacks. The first is that it is labor intensive, requiring a reasonably lengthy time period to develop an understanding of the normal traffic on a network. This time frame is usually 1 to 2 weeks for a LAN of several thousand hosts and requires a monitoring infrastructure. Coverage is also a significant challenge for a network with a hierarchical structure. For larger networks that only want a gross measurement of their traffic, it will suffice to monitor only a border router or major switches.

The second major weakness to traffic analysis is the most worms seen so far operate in a predictable fashion.

By studying one instance of the worm, we have identified the behaviors of nearly all of the worm nodes. However, this will not always be the case. Worms that have updatable modules or even random behavior in their static modules will be difficult to track using specific traffic analysis based on signatures. This is why the methods described here focus on the general properties of the network’s traffic.

The next major weakness of the traffic analysis method to understanding worm behavior is due to the speed of the worm's propagation. A worm that moves sufficiently slowly or only infects a handful of nodes per round will be more difficult to track using traffic analysis than other means (such as honeypot, black hole, or signature-based analysis). The difficulty in this scenario stems from the amount of data when compared to the background traffic on the network.

Traffic analysis will also create some false positives due to the anomalies that appear to be similar between a worm and an attack or a sudden surge in a site's number of clients. For example, while an attack like Code Red would be detected as an exponential increase in HTTP traffic all to Web servers on port 80 with the same request, a site which has immediately attracted widespread attention would show similar behavior. Here, the sensor may classify this as the activity of a worm. However, with some more careful analysis, this can be distinguished. The number of sites being targeted remains constant (in this case one Web server) despite a rapid exponential increase in similar traffic.

Lastly, consider a worm that uses passive mechanisms to identify and attack targets. For example, a worm that attacks Web servers and, rather than hopping from Web server to Web server, now attacks clients that connect to that server. The traffic characteristics remain much the same for the server, such as connections from random clients to the server and then from the server back to clients. This would be difficult to identify, based solely on the patterns of traffic, because little change is observable. The Nimda worm utilized this strategy as a part of its spread, using a vector to jump from server to clients by inserting a malicious file onto the compromised Web server. The Morris worm also followed the paths set up by the compromised system to identify new targets based on the established trust using the remote shell system. In this scenario, the major change in the network's characteristics visible via traffic analysis would be the upsurge in traffic from the compromised systems.

However, none of these weaknesses should prevent the use of traffic analysis in worm detection. For the foreseeable future, most worms will be detectable by these methods and once established they can provide data with minimal ongoing maintenance. Furthermore, the data gathered in this approach can also be used to detect additional network anomalies.

6.2 Honeypots

A network honeypot is simply a system you expect to get probed or attacked so that you can analyze these data later. A honeynet differs from a honeypot in that it is a network of honeypots made of full production systems. (Spitzner, 2003) This network can be logically and geographically dispersed. Because of their nature, worms will indiscriminately attack any available host on the network, including honeypots. The value of this approach is that you can analyze the attack after it has happened and learn about the methods used by the attacking agent. Honeypots come in three basic varieties:

- Full dedicated systems, which are typically nonhardened installations of an operating system. These are installed with a minimum amount of setup in an attempt to mirror a default installation and then placed on the network. External monitors are typically used to capture the network traffic to and from the host.

- Service-level honeypots are hosts that have one or more services installed in logical “jails,” areas of protected process and memory space. An attacker can probe and attack the service, but any compromise is contained to the virtual machine running on the host. Commercial as well as open-source versions of these tools are available.

- Virtual hosts and networks, which provide the illusion of a host and its associated services to an attacker. This is typically housed in a single host on the network, spoofing other hosts.

Each of these approaches offers varying degrees of accessibility and value, along with associated risk. For instance, it can be more costly to implement a set of honeypots with full, dedicated systems, though you may capture more data with real services. A virtual honeypot, however, has an advantage in that you can more readily deploy an additional host or even network into your monitored space.

Honeypots have an inherent risk factor associated with them that has to be stated. Because a honeypot is designed to allow an attacker to enter and gain control (for the purposes of monitoring their actions), it is possible the compromised host may be used to spread more attacks. For this reason it is vital to monitor it closely and both control the outbound connections as well as close the host down when it has been compromised. Also, it should never be deployed on a production subnet where it can interfere with legitimate network activities and be used to gain entry to a protected network.

6.2.1 Strengths of Honeypot Monitoring

Perhaps the single biggest advantage to be gained when using a honeypot is the depth of information available from a compromised honeypot. Because an attacker or, in this case, a worm has attacked the system, a full set of changes to the system can be obtained. This can be useful in determining the nature of the attack. Furthermore, the actual executables used in the worm's propagation are typically also available. With these two pieces of information, a nearly full analysis of the worm can be achieved.

Additionally, with a honeypot, a wealth of additional detection data can be generated. Based on the patterns of attack by the worm and the nature of the

executables, file system signatures of the worm's behavior can be generated. The network behavior signature, including the attack, any communication messages generated, and any probes, can also be identified. With this information, a rich detection system can be developed to look for the worm's behavior.

6.2.2 Weaknesses of Honeypot Monitoring

Honeypot monitoring has a few weaknesses that are worth acknowledging. The first is that typically only one or a small handful of honeypot systems are deployed. While each system gives a detailed set of data about the worm's behavior, they offer only a limited perspective on the network being monitored.

Second, honeypots are labor intensive. They require extensive setup to be effective, and the maintenance and monitoring needed to prevent the use of the honeypot to act as a worm springboard is quite extensive. Properly set up firewall rules, for example, are needed to prevent the system from being a reflector for worm activity.

Due to the variety of systems that are targeted by worms, and the inability to predict what systems will be struck in the future, honeypots necessarily have to be set up with only a limited subset of systems that can be attacked. Worms typically attack systems that are exposed to the world at large, hence services that are exposed to the larger world are best generated using a honeypot.

Lastly, honeypots do not give early warnings about worms; they are typically hit only during the peak times of worm activity. This is due to the limited visibility they have for the network. As such, they can only provide data at the height of the worm's spread.

6.3 Black Hole Monitoring

The implementation of unused IP space in worm tracking has proven to be an even more effective technique in worm detection and tracking. This unallocated, unadvertised network space has no DNS entries, but does have valid routes to reach it. Because it is not in use (no machines are deployed within it) and no photons are traveling along the fiber, it is called a dark space or a black hole network.

Monitoring this dark IP space is effective because of the persistent and complete coverage by Internet worms. Worms, unlike many real attackers, do not monitor DNS entries or service advertisements to determine who to attack. They simply find a network block to scan and begin doing so. Hits in that space are therefore interesting, because no legitimate traffic (in the absence of DNS, application, or routing errors) should be seen in that network.

The scale of the unused network space does not have any direct impact on the usability of the method, although a larger space will give a larger vantage point on the operations of a worm. A network such as a corporate or academic network may have unallocated /27 sized spaces lying about, while network researchers may be able to monitor a space as large as a /8, allowing for a full view of 1/256th of the Internet.

Black hole monitoring generally can be done in one of three ways:

- The first is to monitor what is called backscatter, or the reply packets sent by spoofed sources. If the forged source lies within the monitored dark network space, the replies will be visible. These include SYN-ACK and RST packets from SYN flood attacks, and ICMP errors and control messages from packet floods. This kind of analysis, pioneered by the CAIDA research group, helps in the analysis of DoS and DDoS attacks. This kind of analysis, however, is minimally useful in the analysis of

worms. Because worms typically establish bidirectional communications channels, they generate little backscatter from forged addresses.

- The second method is to simply monitor the number of requests for access to the unallocated network space. These requests are typically monitored by a router that advertises routes to these networks internally. The requests for those routes can be measured either by the flow export data or from the routing table data maintained by the system.

- The third method is to view the network or subnet as a black hole and anything going into it as interesting traffic. This monitors both reply packets as well as requests, such as SYN packets from worms and other scans. While some spurious traffic is certain to enter this space, worm traffic will also enter this monitored area. Captured signatures can then provide a basis for worm analysis, allowing for an estimation of the spread and activity of a worm.

6.3.1 Strengths of Black Hole Monitoring

The biggest strength of network black hole monitoring is the relative ease of data collection. Worms that actively scan will constantly generate data as connection requests are sent to these unused networks. Because worms typically do not correlate the use of networks with their probes, most worms will generate probes to unallocated network space.

The largest challenge facing the use of black hole monitoring is the discrimination of regular probes and attacks from activity from worms. This can generally be done by looking for an exponential rise in the number of sources that parallels a rise in activity sent toward the dark network space. However, this typically yields a larger picture of network activity than other monitoring methods do due to the large scale

of coverage possible. The intentions of the client computer can be assessed on the basis of the intended network destination.

When the third type of black hole monitor is set up (which responds to connection requests to receive the first data packet), worm activity can be measured. In this scenario, the payloads of the captured packets are stored and compared to look for worm activity. This gives deep insight into worm activity, along with a large degree of coverage without the requirement of known signatures, as would be needed for a NIDS monitor.

6.3.2 Weaknesses of Black Hole Monitoring

As described earlier, the biggest weakness in black hole network monitoring is the growing presence of worms that use lists of allocated addresses to target. These threaten to minimize the utility of global-scale dark network monitoring for worm activity. While some worms, such as Code Red and Nimda, will indiscriminately attack any valid IPv4 class A, B, or C address (which does include unallocated space), newer worms such as Slapper and SQL Snake have incorporated lists of allocated network blocks to target. The increased use of this approach will gradually diminish the utility of dark network space monitoring.

Similarly, the threat of hit list scanning, as proposed for Warhol worms and the like, diminishes the utility of dark space monitoring. Since hit lists are built from allocated and in-use system data, the likelihood of a system migrating from allocated to unallocated space is minimal. As such, dark space monitors are of no help in these kinds of worms.

Again, worms that utilize a passive target acquisition model are also likely to be missed by dark network space monitoring techniques. Because worms that use this target acquisition model attack only hosts that are known to be active, they do not

reside in unused network spaces. Hence, they will not be monitored for the kinds of use that dark network space monitoring tracks.

Lastly, changes in network allocation will require updates to the dark network space monitors. For example, if a local subnet becomes used, its utility as a dark space monitor becomes impossible. Similarly, when new networks are allocated in the global IPv4 space, changes must be propagated to the dark network space monitors.

6.4 Signature-Based Detection

At the heart of signature-based detection is pattern matching. A dictionary of known fingerprints is used and run across a set of input. This dictionary typically contains a list of known bad signatures, such as malcious network payloads or the file contents of a worm executable. This database of signatures is the key to the strength of the detection system, and its prowess is a direct result of its speed.

We are interested in three main types of signature analysis for worm detection:

- The first is the use of network payload signatures, as is used in network intrusion detection systems (NIDS). The detection methods used by NIDS engines perform an evaluation of packet contents received from the network, typically using passive capture techniques. (Bace, & Mell, 2001) This can include matching signatures based on payload contents measured by string comparisons, application protocol analysis, or network characteristics. A list of unacceptable patterns are compared against a list of network traffic and alerts are issued when a match is found.

- The second type of signature matching is based on logfile analysis. Application and system logs can contain information that can be used to fingerprint the behavior of a network worm. This can include attack

contents, such as in Web server logs, or simple application errors issued when a worm probes a machine. This is a relatively simple approach but, when joined with other detection methods, provides a sophisticated detection framework.

- The third type of signature detection is the most popular method, file signatures. File payloads of worms and their executables are typically monitored using host-level antivirus products. Several commercial products exist to do this and are typically found on home PCs.

Signature-based detection methods are a powerful way to match known worms through multiple mechanisms. By examining network traffic, file system contents, and server logfile entries, it becomes possible to specifically track the progress of worms as they move on the network. Unlike other detection methods, with a properly crafted signature, detection can be precise and specific, allowing for high-resolution results.

However, it is the specificity of the signature that is also its weakness. Simple mutations or alterations in the contents of the data being screened, such as an altered attack signature or file contents, renders signature-based methods nearly totally blind. These mutations happen frequently, leaving systems exposed that look for only those known contents. Furthermore, signatures can only be generated for known worms and other malicious contents. As such, they cannot be used to identify emerging worms, unlike other methods of worm detection.

6.4.1 Strengths of Signature-Based Detection Methods

The biggest strength to signature-based detection methods is the ease with which they can be developed and deployed. Once a worm (or any piece of malware) is captured and studied or even simply observed, only a brief analysis is needed to develop a signature. This analysis is performed to identify the characteristics that

make the malicious software or traffic uniquely identifiable when compared against a backdrop of normal data. The features that are used in the monitor can be, as noted above, in the logfile entries, the payload of files either on disk or in transit, or in the network traffic generated by the worm.

The relative speed of signature-based detection systems is also another benefit of using them. Large numbers of optimized engines have been developed that can perform pattern matching efficiently, a requirement as communication volumes and the bandwidth of a typical network increase. These detection engines must keep up with this pace and react quickly.

An additional benefit for signature-based detection methods is the ease of removal of the malicious content. For a mail or file server that is being used to distribute the worm, content screening immediately identifies the malicious payload and can quarantine the data. For a network-based intrusion detection system, reactive systems can be triggered to close a malicious connection or install a network filter on a router or firewall to block the compromised machine from continuing the worm's spread. Server level firewalls can also be configured dynamically by analysis engines once a malicious client has been identified from logfile entries.

Lastly, due to the great quantity of malware that exists for the Windows platform, signature-based detection systems in the form of commercial antivirus tools are the easiest route to take. There are simply too many threats to monitor and keep active against without a large pool of resources, which are provided for by the antivirus software vendors.

6.4.2 Weaknesses In Signature-Based Detection Methods

The single biggest drawback to signature-based detection methods is that they are reactionary, they rarely can be used to detect a new worm. Only after an attack is known can it be fingerprinted and made into a signature for use by a sensor. Only if

the attack used by the worm is recycled from a known attack can it be used to proactively detect a worm. Some metasignature detection methods, such as protocol analyzers and related tools that understand protocol parameters, can be used to detect a worm early on. However, these are uncommon in large, coordinated NIDS deployments at this time.

The second drawback to signature-based detection methods is that they don't scale well to large operations. These include networks such as an enterprise or campus networks with thousands of users. Desktop-based remedies are difficult to maintain actively, though many centralized management tools have been developed to overcome this obstacle. However, the volume and distributed nature of the problem makes the issue of scale a difficult challenge to adequately address.

The next major difficulty in a successful deployment of signature-based methods is that it is hard to keep up with variants of worms and viruses. Variations inevitably appear that can evade signature-based detection methods on all levels. Furthermore, when polymorphic techniques are introduced into worms, the challenge rises significantly, making the reliable detection of worms much more difficult.

Network-based signature detection suffers from a number of weaknesses, including payload fragmentation and forgery. These issues are still present in many NIDS products and have been well described by Ptacek and Newsham.

Last, unless in-house signature generation is done, detection is always at the mercy of the supplier of these signatures. While many large and popular packages have rapid responses, as was demonstrated by the Code Red and Nimda worms, this turnaround time can result in a significant delay in relation to the rate of the worm's spread. Signature-based detection methods are only reactionary and always lag behind the introduction of the worm.

CHAPTER SEVEN
DEFENCES

7.1 Firewall and Network Defences

Firewalls have become a commercially successful market item, because of such features as ease of use, application layer filtering, and line speed. Despite these enhancements, little has changed in their basic design principles.

Firewalls are devices that enforce a network security policy. This policy can be the authorization to establish communications between two endpoints, controlled by the ports, applications, and protocols in use. The firewall evaluates connection requests against its rule base and applies a decision to the requested action. (Wack, Cutler, Pole, 2001) Network architects and administrators employ firewall technology to accomplish several key tasks (Wack, 2002):

- **Protection from vulnerable services:** Firewalls protect potentially dangerous or malicious applications from entering or leaving a network.

- **Controlled access to systems:** Filters can control the destinations and sources of network communications.

- **Concentrated security:** By focusing many of the security measures on a single host, the overhead for management and costs of a distributed security system can be alleviated.

- **Enhanced privacy:** A network filter can protect services from being viewed by unauthorized parties.

- **Logging statistics for Internet activities:** This logging of activity can include both normal usage patterns as well as malicious activity originating either internally or externally.

Most firewalling devices are of two basic types. The first is a packet filter, which performs policy enforcement at the packet level. (Chapman, 1992) As each packet in a communications stream passes through the router or bridge, it is compared to a set of rules to determine the action to take, determining the passage or rejection of the packet. The criteria for this decision are typically the source and destination addresses and ports along with a protocol. These usually define the communicating parties and the applications in use.

Packet filters can be either stateful or stateless. A stateful filter understands the context of a communication and can conditionally pass or reject packets that are a part of the communication (or merely appear to be). A stateless firewall, in contrast, only monitors any single packet without any concept of the context of the surrounding traffic. Here, filtering rules would be applied on a packet-level basis as opposed to a connection-level basis.

A second type of firewalling device, a network proxy, performs its decision at the application layer. These devices have additional potentials for security applications and are discussed as proxy-based defence at next section.

7.1.1 Example Rules of Firewall Defences

While IP traffic filtering is itself common, the syntax used by different vendors or firewalling packages varies. The languages used by each reflect various attributes of each product. Several examples are shown to illustrate the fundamental principles of packet filtering. This set is by no means a comprehensive list of all firewall products or their capabilities.

Obviously a firewall is only as good as the rules it contains and enforces. A filter set that defaults to an open policy and has a minimal set of rules does little good and can be trivially circumvented. The syntax and structure of the rules determine the strength of the firewall in relation to the security policy desired.

Cisco IOS-based routers have had filtering capabilities for several of their versions as of this writing. IOS uses access-list (ACL) statements, access-group statements and rules to manage traffic decisions. An example collection of several IOS access-list statements in a configuration would appear as follows:

access-list 100 deny icmp any any fragments
access-list 100 permit icmp any any echo
access-list 100 permit tcp 192.168.1.0 0.0.0.255 any eq 22

These rules will tell the router to drop any ICMP fragmented traffic and allow any ICMP "echo" traffic (typically associated with the ping program). Also, these rules state that the network 192.168.1/24 is allowed to pass for TCP port 22 traffic (associated with the SSH protocol). The use of access-group statements facilitates the management of access lists, allowing for the grouping of rules and addresses.

The Cisco PIX product, a dedicated firewall device, features a filtering statement in addition to the access-list and access-group statements found in IOS. The shun statement provides a coarse-grained filtering capability for filtering networks, as shown below:

shun 10.1.1.27 10.2.2.89 555 666 tcp

This statement would block any TCP traffic from 10.1.1.27 with a source port of 555 to the host 10.2.2.89 with a destination port of 666. The Pix product, like many commercial and dedicated firewall devices, features several other policy enforcement tools, such as virtual private networking services and authentication mechanisms for networks, in addition to application layer handling.

Juniper routers are also capable of handling filter statements in their configurations. The following stanza from a JunOS configuration illustrates the typical layout of such a configuration:

```
term a {
        from {{
                destination-address {
                        10.1.1.1/32;
                }
                protocol icmp;
          }
         then {
                discard;
}
}
```

This rule would block any ICMP traffic to the host 10.1.1.1. JunOS filter rules typically follow the format of containing a statement of criteria to match and then a decision, such as discard or permit, or it may include options as well, such as logging or rate limiting. Arbitrary criteria can also be utilized with this setup.

Lastly, the popular and freely available IP Filter (IPF) tool from Darren Reed can also be used to build a filtering host (http://coombs.anu.edu/~avalon). IPF is available as a module for many popular operating systems, both freely available and commercially supported. Typical syntax for this type of filtering is shown here:

```
pass in proto tcp from 10.2.2.2/24 to \
   10.1.1.2/32 port = 6667
block in on fxp0 proto tcp/udp from any to any \
   port 511<>516
```

These two rules illustrate the syntax for varying rule types. In the first, traffic between two hosts using protocol TCP to port 6667 (associated with the IRC protocol) is allowed to pass. The second statement blocks traffic that arrives on the interface fxp0 (a fast Ethernet interface) of either protocols TCP or UDP between ports 511 and 516. Unlike many commercial firewall packages, IPF does not offer encryption services or rate limiting.

7.1.2 Strengths of Firewall Defences

Because firewall systems are available in a wide variety of scales for line speed, ease of configuration, and in many routers, they are a readily deployable security tool. This can be useful when a new worm appears that uses traffic patterns that can be easily blocked using a network filter.

Because firewalls can permit or deny traffic on a large set of arbitrary criteria, they can be an effective security tool. As demonstrated with IPF and PIX filters, firewall rules can be either coarse grained or fine grained, depending on the filter language used. Combined with packet inspection and dynamic rule sets, a selective filter can be created to enforce a network security template.

Lastly, as described in this chapter, a firewall can be configured to keep a worm out or inside a network. This can be useful to contain a locally found machine that has been compromised by the worm being defended against.

7.1.3 Weaknesses of Firewall Systems

At this time, most firewall systems are able to only filter on the basis of the packet headers. As a result, a typical firewall system is ineffective at defending against a worm for services that must be accessible to the world, for example, Web servers in

exposed networks. Furthermore, because a typical firewall does not examine the contents of a packet, it may block legitimate traffic.

A stateful firewall can be unduly stressed by a large number of active connections. This is typically seen with worms that perform heavy amounts of scanning for target hosts. Due to memory constraints, the firewall may begin to fail and disrupt normal communications during periods of heavy worm activity.

7.2 Proxy-Based Defences

A second type of network firewall is the proxy server. Firewalls built on proxy servers use a technology based on a third party brokering a request for a client to a server. This third party is made up of the proxy server, which is connected to and passes the resulting information back to the client. Through the configuration of a proxy server, network policy can be enforced, controlling applications and network endpoints. This policy enforcement can occur at the level of the connection endpoints, the application in use, or the content of the material being transmitted.

Proxy servers, or application gateways, provide their services by being an intermediate system for a network connection. A listening agent on the proxy server receives a request for a network action and, on behalf of the client, fulfills the request. The connection comes from the proxy server to the destination and the data are passed back to the proxy. The final data transfer occurs between the gateway server and the client. At no time do the client and final destination make direct contact.

7.2.1 Example Configuration of Proxy-Based Defence

In many ways, proxy servers are configured much like listening applications. They are specified to listen on interfaces and accept connections. However, unlike many services in use on a network server, access controls are typically standard for

an application gateway. Additionally, the second endpoint, the intended destination of the client system, can be controlled by the proxy server. If the client is making a request to a system that is off limits, the connection can be blocked at this stage. Application gateway systems can be configured in a variety of ways, some of which are shown in this section.

An application gateway can be used to provide a minimal amount of filtering activity. The Web server Apache, for example, can be used to provide a basic fiter for a site. The following configuration stanza would install a minimal Web-based proxy for normal HTTP communications at the IP address 192.168.1.1:

Listen 192.168.1.1:80
ProxyBlockContent Java
ProxyBlockList /etc/firewall/lists/hacker
<Directory Proxy>
    allow from 0.0.0.0
</Directory Proxy>

As is evident in the above configuration, only a minimal amount of security filtering is in place. Almost any host is allowed to connect without any authentication, and only hosts listed in the file /etc/firewall/ lists/hacker and Java-based content are filtered. Other directives can be employed, as well, including caching content locally or connection controls.

Because proxies work at the level of the application, a variety of access control mechanisms can be employed. These can include network sources and destinations or application-level authentication. For example, a proxy firewall may specify a handful of networks as being “secure” because they are local networks and trusted:

10.100.0.0/16
10.200.0.0/16
10.201.10.0/24

Here, three network segments have been specified as secure networks. This can be used, for example, to configure a variety of services with minimal restrictions and only local network access, no authentication.

A Telnet gateway directive for the FW-Cop application gateway is shown below. Here a Telnet proxy is configured with minimal requirements besides resource controls via a maximum number of connections:

```
# Telnet Proxy Configuration Lines
 proxy {
            maxprocs 999
            path /usr/local/etc/tnproxy tnproxy
            listen 23
            listen 10.100.10.2:23
            maxconn 10.0.0.0/255.0.0.0 10 15
            maxconn 0.0.0.0/0 1 2
 }
```

This is a minimal installation, useful for resource management via a central gateway site.

A Telnet gateway from the Firewall Toolkit (fwtk) can be similarily configured. (Ranum and Avolio, 1994) Again, allowing only hosts on the local network (10.100.10.0/24) to use the gateway, they must authenticate via a password:

```
tn-gw: timeout 3600
tn-gw: permit-hosts 10.100.10.* -passok -xok
tn-gw: permit-hosts * -auth
```

The final line of this configuration stanza allows any hosts from any network to use the gateway provided they have been authenticated. This can be useful for allowing incoming connections from the outside world that have been authenticated.

As a final step, the gateway device, which is also typically the default router for the clients it serves, is configured to not forward packets at the network layer for the networks it serves. This prevents circumvention of the proxy server by making a direct connection to the server on the part of the client. If this were to happen, any security enhancements made by the introduction of the proxy server would be defeated. The only way for the clients to pass to the outside world would be through the application gateway, both the device and at the application layer.

Obviously, application gateways can be far more complex than those shown here. Authentication systems, encryption enabling devices, or content filtering can all be installed in almost any combination. This provides a rigorous control of connections via the gateway server. When combined with packet filtering, the use of proxy servers can be forced and application use restricted.

7.2.2 Strengths of Proxy-Based Defences

Once a client application is configured to use the proxy server, access to network services appears transparent to the client process. The difficulty of the negotiations is handled quietly by the application and data are seamlessly transported back to the client.

Unlike a packet filter, which can only understand the contents of a packet, a proxy device offers true application-layer filtering. This can give the advantage of content specific filtering. As described above, this also gives the advantage of normalizing the communications stream, removing the ambiguity for signature matching and content-inspection or application handling. This gives the network administrators full control over the content of the communications.

Application gateways default to a block policy during failure. Because no other gateway for communications is enabled, if the application gateway fails due to attack or an error, all internetwork communications will cease. No open window for an attack is created by the failure of the network filtering device.

Lastly, because a proxy acts as an intermediate party for the communications, it can fully log all actions. This is dramatically different than the inference from watching packets passively. While this can be used for filtering purposes, it can also be used for audit trail creation.

7.2.3 Weaknesses of Proxy-Based Defences

One of the biggest drawbacks to an application gateway is the latency that it introduces to a communications stream. Because the requests and data are stored before forwarding, a noticeable lag occurs in the time between the request and the completion of that action. Proxying would therefore not work for applications that require real-time performance, such as streamed communications applications.

Because of their placement, the use of application gateways only works for transmissions crossing a border where the filtering devices are in place. It cannot be used to monitor or control intranetwork communications.

Lastly, the setup of an application gateway can be significant for a new application. The interface and specification must be studied and the application altered to accommodate the proxy service. Furthermore, this approach is not available to all protocols and applications, including diagnostic tools such as ping of traceroute. Encrypted communications, such as secure Web transactions using the HTTPS protocol, cannot be proxied without defeating their security measures.

7.3 Active Worm Defence

The previous section have focused on passive defence measures. Hosts can be hardened sufficiently to ensure that a worm that attacks it will fail or be unable to initialize itself. The network overall can be configured and defended to minimize the exposure to an untrusted Internet and the content of malicious requests and data removed. In this way, the worm will attempt to compromise new hosts but fail.

An additional defence strategy is to attack the worm network.(Mullen, 2002) This will essentially turn the devices of the worm network against itself, offering both an entry point into the network as well as a built-in mechanism to utilize in this pursuit. The advantage of this approach is a slowdown of the worm's progress overall, which will eventually lessen the load of any worm on the local network.

Some counterstrike methodologies are based on host-level measures.(Mullen, 2002) Methods such as kernel alterations, interfering with selective processes, or networking attempts by hostile software will not be discussed here. However, they are an interesting design consideration for future methods at the operating system level to defeating hostile executables, regardless of the source.

By attacking the worm network itself, the end goal is to stop one or more nodes of the worm network from continuing to propagation. The major strategies towards this include:

- A message to the network to shut down
- Forged replies to a query that you are already infected
- Poison updates to the worm
- Stalling the worms.

Some are more effective than others, but all can provide an accessible way to help stem the spread of a worm.

The general principle in this section is to find a worm node, using information gathered from an IDS, the system logs, and the like, and attack it back. Because this strategy assumes that each host must be contacted singly, you will have to enumerate each host for worms you wish to target. Because this is a very controversial method for defending against an Internet worm attack, the target select caveats are discussed later in this chapter.

We now look at general strategies. Most of the methods for attacking the worm network outlined above rely on a failure to gracefully handle errors or authenticate data from other nodes. These failures can be used to perform arbitrary actions on the worm node, including shutting it down or stopping the worm process.

Many attack programs are themselves poorly developed and contain unchecked runtime errors. These errors include many of the same types of errors that they are designed to exploit on a target system. By identifying and exploiting these weaknesses in the attacking agents, a decoy target can alter the behavior of the malicious client.

For example, an inspection of the Scalper worm exposes several vulnerabilities. An interesting one is a possible overflow in the handling of cookies sent by the targeted server. In the ViewWebsite() function, only 256 bytes are allocated for the storage of the cookie, and are copied without bounds checking:

```
void ViewWebsite(char *http,char *cookie) {
        char *server,additional[256], cookies[1024],
        location[1024];
        unsigned long j,i;
        struct ainst up;
```

```
        char num=0;
        if (!strncmp(http,"http://",7))
          server=http+7;
        else
          server=http;
        for (i=0;i<strlen (server);i++)
        if (server[i] == '/') {
         server[i]=0;
         num+=1;
         break;
        }
        memset(additional,0,256);
        if (cookie) {
         for (j=0;j<strlen (cookie);j++)
         if (cookie[j] == ';') {
         cookie[j]=0;
         break;
        }
        sprintf(additional,"Cookie2:"
        "$Version=\"1\"\r\ncookie: %s\r\n",
        cookie);
}
...
```

The value of *cookie is set by reading the returned string from the server, also without bounds checking. The failure to do this check can result in a failed worm process when an overly long cookie is encountered. This long cookie is then copied into the array additional, which is smaller than the allowable size of cookies. This can be used by a malicious decoy to attack a worm client and stop the process. Inspection of many of the attack programs available on the Internet reveal similar errors.

7.3.1 Shutdown Messages

The first way to attack a worm network is to tell each node to stop its worm behaviors. This is done by either telling the host to stop all worm-associated processes or to simply shut down. For worms that accept network communications and commands, such as Slapper (accessible via UDP interface) or the IIS worms Code Red and Nimda residual cmd.exe shell, it is possible to send the worm a remote command and to shut the worm system off.

There are two ways to gain entry to a worm node. The first is to attack the worm's communications interface. In the case of the Slapper or Scalper worm this is through the listening interface on UDP port 2002 that accepts commands from other worm nodes. The second is to attack the wormcompromised host in the same way the worm did and to exploit a vulnerable service.

The use of the communications interface assumes that there are no authentication mechanisms in the interworm connections. When this is the case, as is with Slapper and Scalper, one can simply send a command to be run to the worm node via the listening interface. The commands typically remove worm-associated files and kill the worm's processes, such as its scanner and attack components. For a Code Red or Nimda compromised host, the following request format should typically work:

http://172.17.3.45/scripts/root.exe?/c+shutdown

The IP address 172.17.3.45 will, of course, depend on the attacking host. The shutdown command tells the system to stop its operations and begin shutting down, stopping the worm's activity.

The second method of gaining entry to the remote worm host, by attacking the host itself, is a little trickier. The basic operation is to perform the same exploit of the vulnerability that the worm used to gain entry but to use a different set of options. Whereas the worm itself will typically install the files needed to target hosts and

attack them, in this scenario, the commands remove worm files and kill processes associated with the worm component. This system will not work for hosts that have been upgraded by the worm, which has been performed by some worms but not by several of the more major, recent worms, such as Code Red and Nimda.

These methods treat the worm host as a server to which a machine under your control connects. Typically, some information about the worm, including the worm executables themselves, is required. With the information from the analysis of those pieces, vulnerabilities in the design of the worm can emerge.

The natural defence for a worm against such an attack is to strongly authenticate messages received from the network, which can be done with the use of cryptography. Then an adversary, namely, an administrator attempting to inject messages to shut down the worm host, would have to break the encryption used by the worm network in order to have the message accepted. While it may be possible to break into the worm host using the methods first used by the worm to gain entry, if the worm fixes the vulnerabilities it used during installation then this becomes difficult to do. Some worms, such as the ADMw0rm, used these methods to keep would-be attackers away.

7.3.2 "I am already infected"

The next method of attacking the worm network by using its own methods against it is to convince the attacking worm that the target is already compromised by the worm. This works for worms that first check for their presence on the target system before launching. This check can be for a process name, a filename, or some other indication that the worm is already installed on the system.

Such an attack is possible against a handful of worms, including Code Red and Slapper. Code Red looks for the file C:\\notworm and, on finding it, ceases operation. Slapper, in contrast, is unable to begin operation if its filename is already in use and

UDP port 2002 is unavailable to begin listening. This attack is also possible gainst Warhol worms, which use an indicator to the attacking node during the permutated scans. This method of delaying the worm's spread was also discussed during the outbreak of the WANK and OILZ worms. (Oberman, 1989)

The attack works by exploiting the check made by the worm for its own presence. Some worms, such as those listed above, will attempt to avoid double infection on any host. A quick check for the worm's indicator on the system is performed before launch. Other worms, such as Slapper, ungracefully handle the condition of double infection due to colliding requirements during startup.

The attack against such a method used by the worm is often quite easy to perform. It is typically enough to either install stub files of the worm process or to start a process with the same name as that used by the worm. In the case of Code Red, for example, you would create an empty file C:\\notworm. For the Slapper worm, in contrast, you would simply bind a listening process on UDP port 2002 that will cause the worm's startup to fail.

As a defensive measure, an administrator can install worm files with the same name and make them immutable. During the attack and installation of the worm, the worm application cannot install new files. This effectively blocks the worm before it launches as it cannot install itself. Note that this method does not stop the attack of the remote worm system on a local host. Rather, it simply prevents the worm from installing and launching locally. This method also takes advantage of the predictable nature of most worms.

7.3.3 Poison Updates

The next method of attacking the worm network as a countermeasure assumes that the worm can be updated. Most worms are typically static and not able to accept changes in their behavior via updated modules.

Typically, a worm such as this would be updated by its users, often those who wrote or launched the worm. In this countermeasure this mode of entry is abused by outsiders. The attacker, such as an administrator for a network, sends the worm node or even the network a new module. However, unlike the updates sent by the users of the worm system, the new module is designed to hinder the worm's operation, not enhance it. The module can contain empty attack routines, for example, which return success despite not actually attacking a node.

An alternative strategy is to disable the worm entirely. The injection of modules that contain broken routines that fail no matter what will achieve this goal. Because the update crashes the worm programs (or even the entire system), the worm can not operate and the worm node is effectively shut down.

For creators of worms and those who would use them, two major defences are possible. The first is to authenticate modules in much the same way as was used by a worm receiving messages. This ensures that the modules came from a trusted source and not an outside attacker. Public key cryptography, for example, would allow for the authentication of the source of the module. The second method is to not discard the old modules when an update is received. Instead, keep the old modules intact and use them as needed. The worm can choose from known modules and still achieve success. An obvious attack against this is to send so many modules to the worm node that it consumes all of its storage space and only contains the attacker's modules.

7.3.4 Slowing Down The Spread

One simple way to slow the spread of a worm network is to abuse two key features of how a typical worm operates. First, you abuse the scanning and reconnaissance operations of the worm by giving it extra work to do. Secondly, you abuse protocol options to make your section of the network "stickier" than it should

be. In this way you can hold the worm around longer, preventing it from spreading as fast.

Network worms will typically begin by scanning a network for targets to attack. Scans such as this will make a connection to the host service being offered before they launch an attack. Since nodes on a network do not know which addresses are occupied and which are not, they will scan all addresses in a given network space.

This method of attacking the worm works by sending forged replies for hosts that do not exist. The worm scans will attempt to make a connection to a host, requiring an ARP mapping be made. The subnet's router will attempt to resolve this so it can forward the connection request. In the absence of a host listening at that address, the requests will go unanswered:

23:27:27.312595 arp who-has 68.40.154.84 tell 68.40.152.1
23:27:30.527061 arp who-has 68.40.154.84 tell 68.40.152.1
23:27:37.088597 arp who-has 68.40.154.84 tell 68.40.152.1

The method is then simple: A host will forge replies to these requests and handle the connection. What it does next, then, is part of the trick. It advertises a receive buffer in the SYN-ACK packet it sends back, but since it never really established a connection, it will never continue the dialogue. The worm system will send an initial payload to it but will stall when it has nothing left to send, having filled the receiving host's window.

A second method employed here is to use an HTTP option to keep the connection alive. This method normally reuses a Web connection for multiple objects. However, by setting the connection to be persistent, the client will stay connected to the server for a longer period of time.

Using these techniques, LaBrea is able to have worm-infected hosts stick around longer. The larger advertised network along with the persistence of the connections

stalls the progress of the worm. Though this does not eliminate it, it does provide an increased window of time to implement a larger solution.

This method of attacking the activity of the worm can be utilized by a honeypot installation. By creating many virtual honeypots as described in Chapter 6, the network population is artificially inflated and the worm is given more work to do. By using the black hole monitoring technique of sending a single packet to establish the connection from these virtual hosts, the network can stall the progress of the worm

7.3.5 Strengths of Attacking The Worm Network

Obviously the biggest advantage of attacking the worm network is that the attacks, either in the form of probes or actual attacks, are stopped at the source. Provided the attack was successful, the worm will be stopped at that node.

For the method used by the LaBrea tool, which can also be used by the dark network monitor tools described in Chapter 6, the main advantage for a security administrator is that the worm's progress is slowed. In the time gained by slowing the worm's spread, site officials can take corrective actions and remedy the problems at the host itself.

7.3.6 Weaknesses of Attacking The Worm Network

Because these methods all attack one node in a worm network at a time, they are time consuming and laborious. After detection, each node must be attacked individually to stop its behavior. This can quickly become intractable in scale.

While the strategy of using the same files and methods the worm uses, and making them immutable, is tempting, it is trivially overcome. One simple method to overcome this is the use of random file and process names for worm components.

This would prevent the use of empty or immutable files to block the worm's installation on the host. To block injected messages, such as shutdown messages, the worm could easily employ some strong form of proof that the target host is already infected, using an encrypted nonce for example. Lastly, the worm could simply ignore attempts if the target is already compromised and accept attempts at double infection.

A worm can take two major defences to defeat LaBrea-type countermeasures. First, the use of aggressive scan timeouts by the worm will decrease the impact of the added "hosts" on the network. Secondly, a worm that only launches its attack against known servers would be largely immune from this method. The targeted type of worm in this method is the type that uses active scanning to identify targets.

Furthermore, the methods outlined here are reactive in their nature. They do nothing to protect a host or a network from worm attacks as they happen or while an administrator is away. While they may remedy the situation for a brief time period, they are best used long after the worm's initial spread is over.

CHAPTER EIGHT
GROUP BASED MODEL OF WORM DEFENCE

8.1 Introduction

This model of defence is based on the willing co-operation of a set of hosts on a pre-arranged protocol which is described below.

An alert message is spread to the set of participating hosts in order to stop the spread of the worm. This alert can be sent from the detector to the entire set or a small subset of participating hosts. We will describe all properties of this message below. Our goal is to maximize the number of hosts that can be prevented from contracting the worm.

In this chapter as well as the next, we develop mathematical models for the simplest of the scenarios. Then, we go on to develop simulations to study more complex scenarios of worm mitigation.

8.2 The Model

8.2.1 Definition

Once a worm is detected, an alert message is spread to the set of participating hosts to stop the spread of the worm. Using this alert message, we prevent from infection to the our nodes and we reduce to infection of worm. It is true, when some nodes detect the worm and send the an alert message for this. But, if we do not detect the worm, we do not this.

How a worm is detected or declared has been discussed in chapter 6. But the worm may be polymorphic or its signature is not known or it is not identified, etc. Due to these reasons, the worm may not be detected. So that we can use alert message not just when worm detected, we use it for every suspect situations.

Now, we can define some abbreviations below which will be used in our model:

- **Group Size (G):** Number of children for hierarchical networks.

- **Threshold (τ):** It is the limit value. When network reachs the value, then defence unit will be active. ($0 < \tau < G$)

- **a:** The number of infected node.

- **c:** The proportion of alerted members.

- **M:** The total number of response members

- **Infection Rate (r):** Some worms spreads very fast and the other spreads slow. We use the term of infection rate as spread speed of worm.

- **Alert Message (A):** It is an alert message. Every cycle, every parent gets this message from their children. It is a number which is between 0 and 1. If detection unit detect a worm send this message as 1. But as we mentioned above, sometimes detection unit can not detect the worm however it observes some anomalies in the node. It may be a worm which does not known worm. In order to prevent network we consider this anomaly. And according to this anomaly, detection unit decide the alert message. If it is high, the alert message will be near 1. If it is not, it will be near the 0. If everything is normal, no worm and no any anomalies, it will be 0. ($0 \leq A \leq 1$)

- **Alert Level (AL):** It is total of received alert message for a node.

$$AL = \sum_{k=1}^{G} A \qquad (8.1)$$

If an uninfected node receives at least thresold(τ) alert level(AL) in the same timestep, then defence unit will be active and the node will be protected now.

8.2.2 Mathematical Model

In chapter 2 we mentioned about the worm spread. Shortly, we mentioned it again for our mathematical model. Worm infections can grow in an exponential pattern, rapidly at first and then slowing as a plateau value is reached. This is a typical kinetic model that can be described by a first-order equation:

$$N.da = (Na).K(1-a)dt$$

a: is the proportion of vulnerable machines that have been compromised
t: is the time,
K: is an initial compromise rate
T: is the constant time at which the growth began

It can then be rewritten in the form of a differential equation:

$$\frac{da}{dt} = Ka.(1 - a) \quad (8.2)$$

This describes the random constant spread rate of the worm. Solving the differential equation yields:

$$a = e^{K(t - \tau)} / (1 + e^{K(t - \tau)})$$

Rate *K* must be scaled to account for machines that have already been infected, yielding $e^{K(t - \tau)}$

This is an interesting equation. For early t ($t << T$), a grows exponentially. For large t ($t >> T$), a goes to 1 (all vulnerable hosts are compromised). This is

interesting because, it tells us that a worm like this can compromise all vulnerable machines on the Internet quickly.

For a given member, the expected number of co-operating nodes who remain in the normal, un-alerted state is:

$$F.(1-c)$$

The number of alerts a particular member sends in time dt is:

$$F.(1-c).a.dt$$

This implies that the total number of alerts system wide is given by multiplying the above term by M, the total number of response members. Since each member needs τ alerts before it can change its state, the number of members changing state in time dt is given by:

$$\frac{dcM}{dt} = \frac{F.(1-c).a}{\tau}$$

Rearranging the terms, we get the evolution rate of the number of alerted members in the following differential equation:

$$\frac{dc}{dt} = \frac{F.(1-c).a}{\tau} \quad (8.3)$$

The proportion of member already infected is obtained by altering (8.2) to include the fact that cooperatively alerted members will be able to block the worm. Two types of infection attempts are considered, local and global infection. Local infection is an infection that spreads from a host to another host without having to pass through any router. Global infection is an infection that needs to pass through a router. When an infected host tries to infect another one across a router, the infection

must pass through two filters, the local filter that blocks outgoing infections and the remote filter that blocks incoming infections. The probability that both of these are not alerted is $(1-c)^2$. Thus the global infection is

$$a.K(1-a).(1-c)^2$$

The local infection rate is same as the rate equation (8.2) because there are response devices between infection source and target. Since there are M response members, the probability of a host choosing a target behind the same router is $\frac{1}{M}$ and behind another router is $(1-\frac{1}{M})$ Combining these probabilities and the infection rates, equation (8.2) becomes:

$$\frac{da}{dt} = a.K(1-a).(1-c)^2.(1-\frac{1}{M}) + a.K(1-a).\frac{1}{M} \quad (8.4)$$

Thus we have a pair of differential equations which can be solved to get the number of infected and number of alerted members.

According to our model, hosts back off from fitering the traffic after a certain time period. The rate of back off is directly proportional to $(1-a)$ and also to the proportion of alerted members, c. Thus equation 8.3 becomes:

$$\frac{dc}{dt} = \frac{F.(1-c).a}{\tau} - \varepsilon.(1-a).c \quad (8.5)$$

where ε is a constant indicating how fast a host backs-off from filtering traffic.

Then in state t, there are a(G - a) exponential random variables in progress at once, since each of the (a) infected nodes is trying to infect each of the (G – a) uninfected nodes. Then the time to go from state t to t + 1 will be the minimum of a(G – a)

exponentially distributed random variables, and thus will itself be exponentially distributed, with mean $\frac{1}{a.(G-a)}$

For simplicity, we will consider the case $\tau = G - 1$. The total expected time to an alert, starting at the time the first member of the group becomes infected, is

$$\sum_{a=1}^{G-1} \frac{1}{a.(G-a)}$$

Using a standard approximation,

$$\int_{1}^{G-1} \frac{1}{x.(G-x)}.dx = \frac{1}{G} \int_{1}^{G-1} (\frac{1}{x} + \frac{1}{G-x}).dx = \frac{2}{G}\ln(G-1) + C$$

where C is the constant of integration. The latter quantity goes to C as $G \rightarrow \infty$

In other words, the first equality remains bounded as $G \rightarrow \infty$. This is a very interesting result, since it says that the mean time to alert is bounded no matter how big our group size is. This is verified in our simulations.

8.3 Architecture Of The Model

In order to simulate the our model, we use 3 units which names are infection unit, detection unit and defence unit.

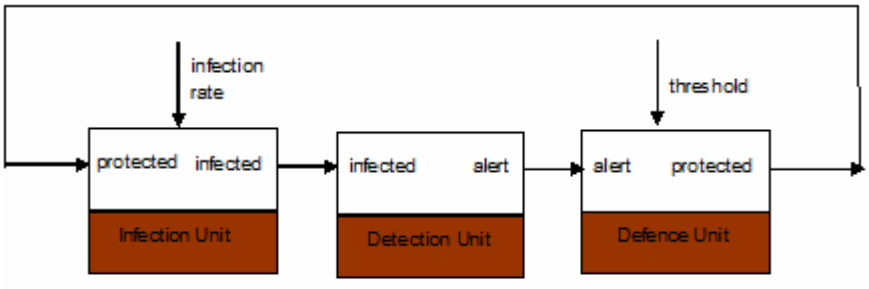

Figure 8.1 Architecture of the model

8.3.1 Infection Unit

It behavior a worm. The rate of infection is fixed at the beginning of the simulation. The exact number of machines that each infected machine tries to infect is determined by using a Poisson distribution, with the mean value as the rate of infection.

There are 3 parameters for this unit which described as below:

- **Infection rate (r):** Using this parameter, we control the spread of worm. It is a constant value so that, it define before the simulation.

  r: Double

- **Infected:** It is the output of this unit and it is a boolean array value. If it is true, then it will be infected for the node. The array length is equal to group size (G).

  Infected: Array [1..G] of Boolean

- **Protected:** It is the input of this unit and it is a boolean value. The input value, comes from the defence unit. If it is true, the infection unit will not infect the node because it is protected.

  Protected: Array [1..G] of Boolean

8.3.2 Detection Unit

There are 2 parameters for this unit which described as below:

- **Infected:** It is the input of this unit and it is a boolean array value. The input value, comes from infection unit. The array length is equal to group size (G).

  Infected: Array [1..G] of Boolean

- **Alert Message (A):** It is the output of this unit. If the unit detects a worm it send this message as 1. ($0 \leq A \leq 1$)

  A: Array [1..G] of Double

8.3.3 Defence Unit

It is the main unit which we will control. There are 3 parameters for this unit which described as below:

- **Threshold (τ):** When network reachs the value, then defence unit will be active. It is a constant value so that, it define before the simulation. When the *AL* reaches the threshold, the detection unit will be active. ($0 < \tau < G$)

  τ: Integer

- **Alert Message (A):** It is the input of this unit which comes from detection unit. It must be between 0 and 1.

  A: Array [1..G] of Double

- **Protected:** It is the output of this unit and it is a boolean value. The true means, the node will be protected.

  Protected: Array [1..G] of Boolean

8.4 Description of The Simulation

The simulation was done on a network modeled as a tree with 4 levels. Each level of the tree has 4 children. The simulation is started by randomly infecting a single leaf node. The rate of infection is fixed at the beginning of the simulation.

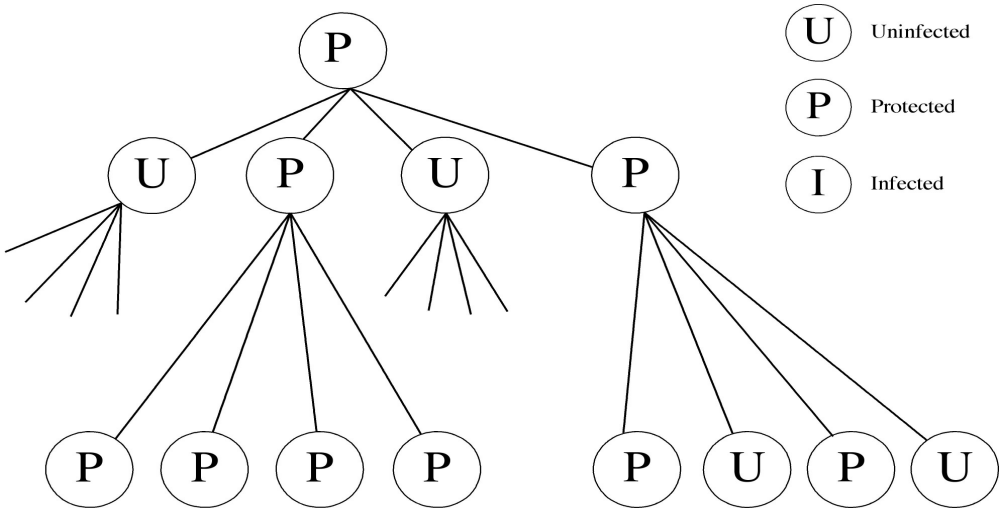

Figure 8.2 Our simulation's network

The exact number of machines that each infected machine tries to infect is determined by using a Poisson distribution, with the mean value as the rate of infection.

In each time slice, every infected machine tries to infect as many other machines as dictated by the Poisson distribution. Alerts are raised in the same time slice as an infection occurs. And each alert is propagated as high as possible in the tree in the same time slice.

Simulations were run with thresholds at 75% and 50% of the number of children. That is, if 75% of a node's children have raised alerts, the node takes action. The structure of network and thresholds were chosen so as to be comprehensible. However, more complex structures with different number of children at each level and different thresholds at each level could also be simulated.

8.5 Discussion of The Results

The basic results of two extreme cases where all parameters are identical except the rate of infection which is very high and very low are shown in Figure 8.3 and

Figure 8.4. In these two figures, for alert message we just use 0 and 1. It means that, the worm is not polymorphic, and its signature is known in advance or it is identified in real time. These two figures show that the number of infections before complete immunization could take place is almost the same for both the cases.

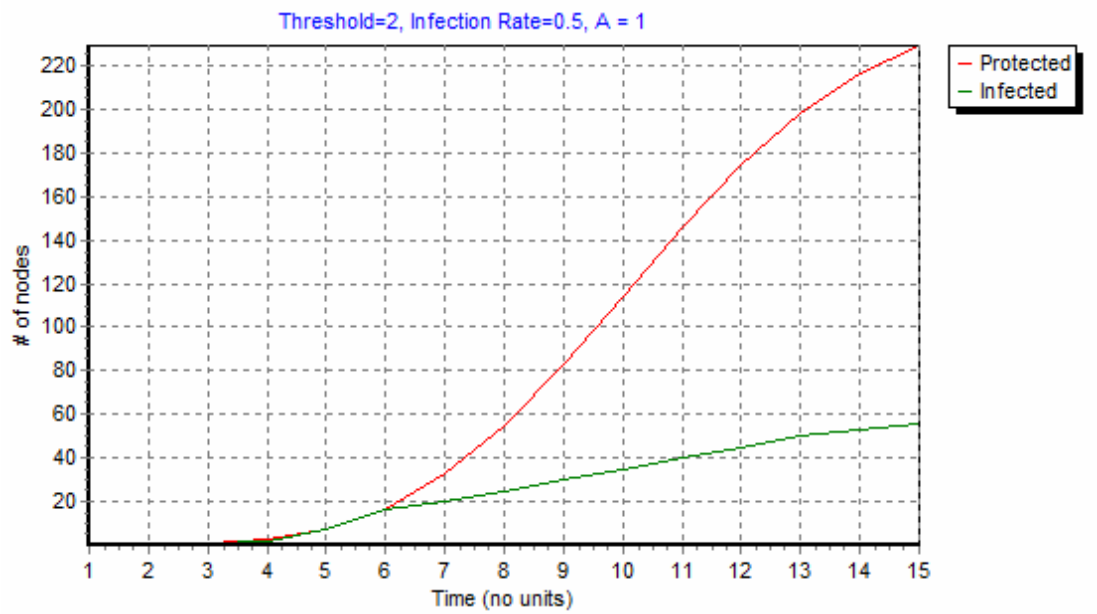

Figure 8.3 Response for a low rate of infection and for known worms

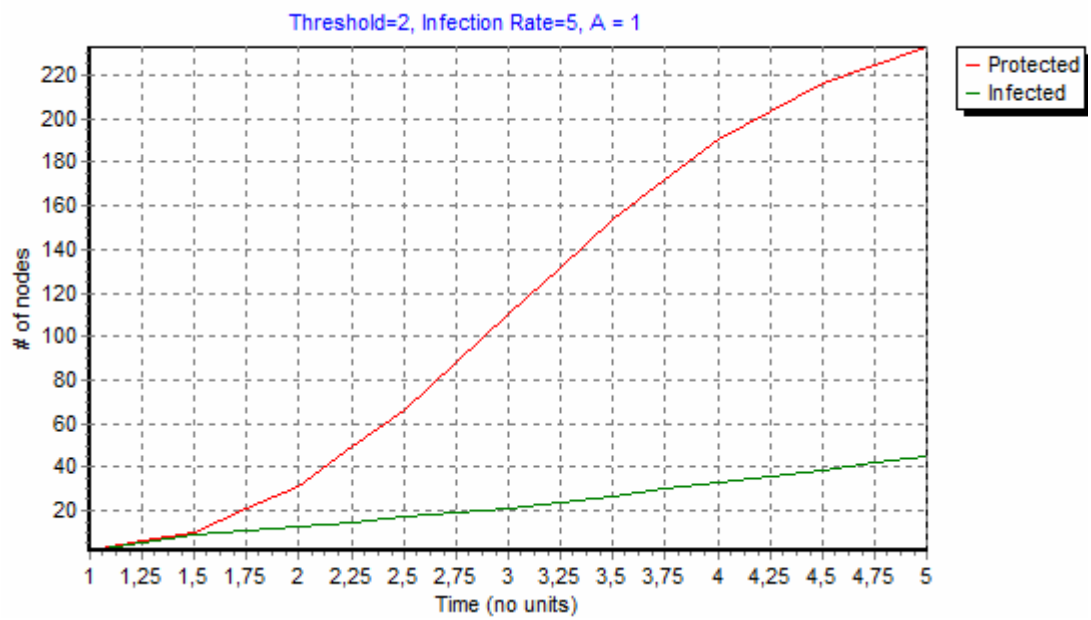

Figure 8.4 Response for a high rate of infection and known worms

In Figure 8.5, the worm may be polymorphic and its signature is unknown, but regarding the some anomalies, the detection unit send alert message as 0.8. As we can see the in the Figure 8.5, and although infected node more than protected, it is work. Due to the worm is unknown, detection unit would be sent just 0 for the alert message and there would be any protected nodes. But in Figure 8.5, detection unit observes the traffic and when it finds some anomalies in the node, it will send the alert message as between 0 and 1.

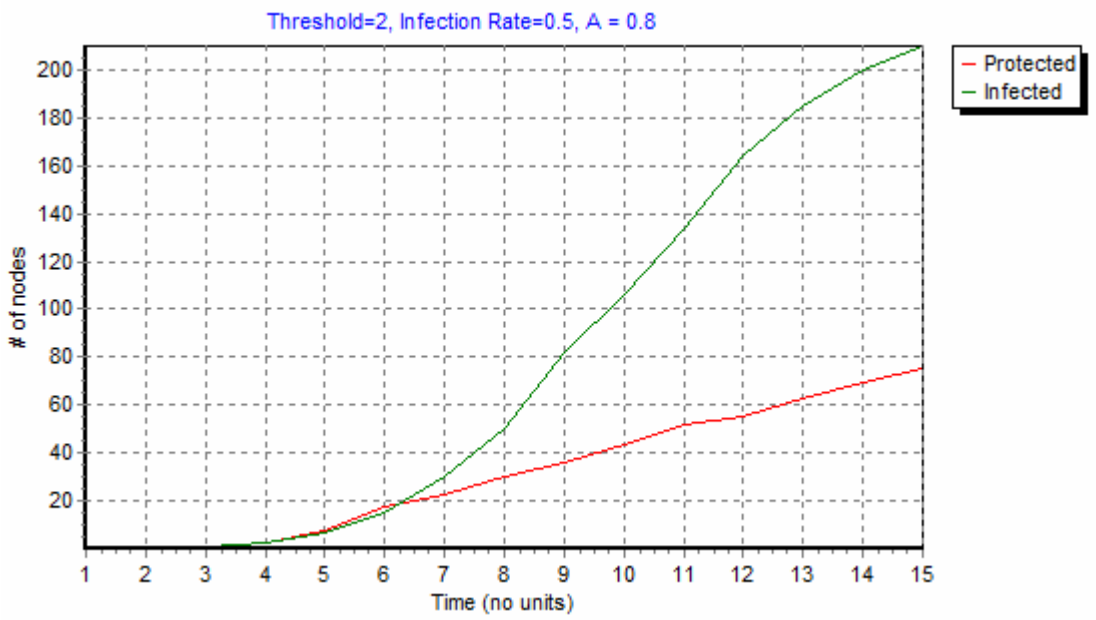

Figure 8.5 Response for unknown worms

Figure 8.6 shows that varies with threshold is the number of infected machines. A low threshold helps to save a lot of machines. Using threshold, we can define our tolerance.

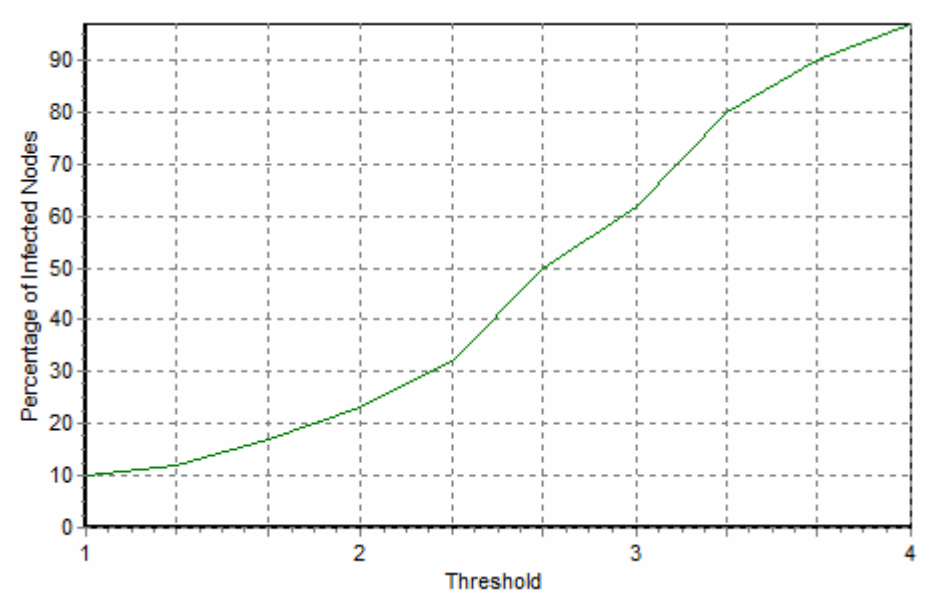

Figure 8.6 Percentage of machines infected for different threshold values

8.6 Summary and Conclusions

This chapter provided a mathematical analyses of the group based model of worm defence and showed that the simulation results.

From this model, and the simulations, we can determine the thresholds required for a given tolerance of lost machines. With the threshold levels and detection unit thus determined, we can effectively inhibit the spread of worms .

CHAPTER NINE
CONCLUSIONS AND FUTURE WORK

9.1 Research Contributions

In this thesis, a research has been done characterizing and analyzing computer worms and a defence system model against the computer worms has been developed. Research has been made in the following areas:

1. Worms and their scan techniques

2. Analyzing of the worm detection methods

3. Analyzing of the worm defence methods

4. Modeling a defence system against the computer worms

9.1.1. Worms and Their Scan Techniques

This thesis gave a stepwise introduction to a worm. It traced the genesis, evolution and the state of the art of worm technology. A brief history of worms was provided and an extensive background about them including their history and taxonomy. It developed a comprehensive model of a simple worm and described the components that make up a worm. In chapter 4, discussed possible future worms. It provided a detailed account of various worm scanning techniques. Network bandwidth and latency constraints faced by the worm during spreading were explained in detail. These scanning techniques and constraints were explained.

9.1.2 Analyzing of The Worm Detection Methods

The goal of our detection strategies is to detect nearly any type of worm with as little effort as possible. To do this, we focuses on the features common to most worm types and build strategies to detect these characteristics. There are different methods of worm detection. These methods are traffic analysis, the use of honeypots, dark network monitors, and the employment of signature-based detection systems. It analyzed these various techniques and mentinoed about their advantages and disadvantages.

9.1.3 Analyzing of The Worm Defence Methods

There are various stages of the life cycle of worm defense. The life cycles contains following steps: Prevention, prediction, detection, analysis, mitigation, curing, vaccination and patch similar vulnerabilities. It showed how the fight against computer worms is an on-going process without an end. It emphasized the need to keep systems patched up and up to date and the importance of preventing a worm incursion rather than trying to catch up with it.

There are 2 defence strategies, active and passive. Firewall, network defences and proxy-based defences are passive defence, active worm defence is active defence strategies.

Firewall systems are a popular network security device. When properly configured, a firewall can enforce the security policies of a network and become an effective tool in the defense against worms. However, even with their widespread deployment, Code Red and Nimda were able to deeply penetrate many networks that were otherwise protected. Altough firewall is effective tool in the defense against worms, it is not the final solution for network security.

9.1.4 Modeling A Defence System Against The Computer Worms

In Chapter 8, the group based defence is modeled. This model of defence is based on the willing co-operation of a set of hosts on a pre-arranged protocol. We developed mathematical models for the simplest of the scenarios. Then, we went on to develop simulations to study more complex scenarios of worm mitigation. Group based model of worm defence is discussed with the simulations results.

From this model, and the simulations, we can determine the thresholds required for a given tolerance of lost machines. With the threshold levels and detection unit thus determined, we can effectively inhibit the spread of worms .

9.2 Conclusion and Future Work

Computer worms are a self-propagating computer program that is being increasingly and widely used to attack the Internet. Because they spread extremely fast, usually install malicious code and it could access confidential information. This thesis concludes that worms could be very dangerous to the Internet. But there are several techniques to mitigate the ill-effects of worms as illustrated in this thesis.

In the future work, we would like to design worm defence system. An effective practical worm defense system is important. The following important questions will be answered: How can firewalls cooperate with each other to block worm traffic? How do firewalls treat traffic differently from the normal pattern? How can the system defeat the malicious firewalls?

REFERENCES

Bace, R., & Mell, P. (2001). *NIST Special Publication on Intrusion Detection Systems*. Retrieved June 18, 2007 from http://download.at.kde.org/infosys/security/snort/docs/nist-ids.pdf

Chapman, D. B. (1992). *Network (In)Security Through IP Packet Filtering*. Retrieved July 29, 2007 from http://www.greatcircle.com/pkt_filtering.html

Cheetancheri, S. G. (2004). *Modelling a Computer Worm Defense System. Ms. Thesis*. University of California Davis in Computer Science.

Lee, K. (2006). *Polymorphic Worm Detection by Instruction Distribution*. Retrieved March 19, 2007 from http://www.isit.or.jp/lab2/kouryu/2006/postech/postech_jkim_hpc/postech_hpc_kihunLee.ppt

Mullen, T. (2002). *The Right to Defend*. Retrieved July 29, 2007 from http://www.securityfocus.com/columnists/98

Mullen, T. (2002). *Defending your right to defend: Considerations of an automated strike-back technology*. Retrieved July 31, 2007 from http://www.hammerofgod.com/strikeback.txt

Munson, H. G. (1991). *928 F.2D 504: United States of America v. Robert Tappan Morris*. Retrieved January 24, 2007 from http://floridalawfirm.com/iplaw/worm2.html

Nazario, J., Anderson, J., Wash, R., & Connelly, C. (2001). *The Future of Internet Worms*. Retrieved January 29, 2007 from http://www.blackhat.com/presentations/bh-usa-01/JoseNazario/bh-usa-01-Joes-Nazario.pdf

Nazario, J. (2003). *Defense and Detection Strategies against Internet Worms*. (1th ed.). Norwood: Artech House.

Oberman, R. K. (1998). A-2: The *W.COM Worm affecting VAX VMS Systems*. Retrieved August 5, 2007 from http://www.ciac.org/ciac/bulletins/a-02.shtml

Spitzner, L. (2003). *Honeypots Definitions and Value of Honeypots*. Retrieved June 10, 2007 from http://www.tracking-hackers.com/papers/honeypots.html

Staniford, S., Paxson, V., & Weaver, N. (2002). *How to Own the Internet in Your Spare Time*. Retrieved February 19, 2007 from http://www.cs.pitt.edu/~mosse/3510/vern-usenix02.pdf

Wack, J., Cutler, K., & Pole, J. (2001). *Guidelines on Firewalls and Firewall Policy: Recommendations of the National Institute of Standards and Technology*. Retrieved April 29, 2007 from http://csrc.nist.gov/publications/nistpubs/800-41/sp800-41.pdf

Wack, J. (2002). *Keeping Your Site Comfortably Secure: An Introduction to Internet Firewalls*. Retrieved July 27, 2007 from http://www.windowsecurity.com/whitepapers/Keeping_Your_Site_Comfortably_Secure__Introduction_to_Firewalls.html

Weaver, N. C. (2001*). Warhol Worms: The Potential for Very Fast Internet Plagues*. Retrieved March 11, 2007 from http://www.cs.berkeley.edu/nweaver/warhol.html

Wikipedia. (2007). *Computer worm*. Retrieved January 24, 2007 from http://en.wikipedia.org/wiki/Computer_worm

Xia, J., Vangala, S., Wu, J. & Gao, L. (2006). Effective Worm Detection for Various Scan Techniques. *Journal of Computer Security, 14*, 359-387. Retrieved February 19, 2007, from ACM database.

Zalewski, M. (2000). I *Don't Think I Really Love You, or Writing Internet Worms for Fun and Profit*. Retrieved February 17, 2007 from http://lcamtuf.coredump.cx/worm.txt