Modelling a Computer Worm Defense System

By

SENTHILKUMAR G CHEETANCHERI

B.E. Computer Science & Engineering. (Coimbatore Institute of Technology, Coimbatore, India) 1998

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Professor Karl N. Levitt (Chair)

Associate Professor Matthew A. Bishop

Doctor Jeff Rowe

Committee in Charge

2004

Modelling a Computer Worm Defense System

Copyright 2004
by
Senthilkumar G Cheetancheri

Contents

1 Introduction **2**
1.1 Contribution of this Thesis to the field . . . 3

2 Background **4**
2.1 Introduction . . . 4
2.2 Model of a worm . . . 5
2.3 Examples of worms . . . 8
2.3.1 Xerox Parc worm . . . 8
2.3.2 Morris worm . . . 8
2.3.3 Code Red and Nimda . . . 9
2.3.4 Slammer . . . 9
2.4 Summary . . . 11

3 The life cycle model of worm defense **12**
3.1 Prevention . . . 12
3.2 Prediction . . . 13
3.3 Detection . . . 13
3.4 Analysis . . . 14
3.5 Mitigation and response strategies . . . 15
3.6 Curing the infected hosts . . . 15
3.7 Vaccinating uninfected hosts . . . 15
3.8 Patching similar vulnerabilities . . . 16
3.9 Summary . . . 16

4 Worm Scanning Techniques **17**
4.1 Topological Scanning . . . 17
4.2 Permutation Scanning . . . 17
4.3 Local Sub-net Scanning . . . 18
4.4 Hit lists scanning . . . 18
4.5 Random Scanning . . . 19
4.6 Scanning Constraints . . . 19
4.7 Summary . . . 20

5 Future Worms **21**
5.1 Warhol Worms 21
5.2 Flash Worms 21
5.3 Stealth Worms 22
5.4 Polymorphic Worms 23
5.5 Miscellaneous Worms and Viruses 24
5.6 Summary 24

6 Models of Worm Defense **26**
6.1 Manual Vs. Automatic Mitigation 26
6.2 Means of Automatic Mitigation 27
6.3 Detection vs. Declaration 28
6.4 Prevention 29
6.5 Preventive tools 30
6.5.1 Kuang 30
6.5.2 NetKuang 31
6.5.3 NOOSE 31
6.5.4 TrendCenter - Predict the next worm 31
6.5.5 Other systems 32
6.6 Mitigation Models 32
6.6.1 Friends Model 32
6.6.2 Hierarchical Model 33
6.7 Summary 33

7 Worm Prediction - TrendCenter **34**
7.1 Introduction 34
7.2 Goals 34
7.3 Motivation 35
7.4 The Process 37
7.4.1 Sanitizing Intrusion Data 39
7.5 Discussion of the results 42
7.6 Issues 43
7.7 Future Directions 43
7.8 Summary and Conclusion 44

8 Friends Model of Worm Defense **46**
8.1 Introduction 46
8.2 The Model 46
8.2.1 Assumption 46
8.2.2 The Worm Race 47
8.3 Mathematical Models 49
8.4 Description of the Simulation 53
8.5 Discussion of the results 53

8.6 Limitations 55
8.7 Future Directions 56
8.8 Summary and Conclusion 56

9 The Hierarchical Worm Defense Model 57
9.1 The Model 57
9.2 Mathematical Models 58
9.3 Description of the Simulation 61
9.4 Discussion of the results 62
9.4.1 False Alarms 67
9.4.2 Stealth Worms 67
9.5 Future work 72
9.6 Summary and Conclusions 73

10 Summary and Conclusions 74

11 Future Work 77

A The AttackTrends Sanitizer 79

Bibliography 81

Acknowledgements

All credits to my adviser Prof. Karl N. Levitt for his unstinting moral support, guidance and encouragement in this research and life in general. He has been my leading light and a role model to look up to. The most humble person I have met. And the one who stood by me in times of crisis. My thanks are due to Dr. Jeff Rowe for his valuable suggestions and pep talks and to Prof. Matt Bishop for his constructive critique. I'd like to thank DARPA and the US Air Force for funding this research. My gratitude is due to Dr. Norman Matloff who helped me in the mathematical analysis of the models developed in this research and to Todd Heberlein for his numerous mails and write-ups that showed me how to write. I'm indebted to my friend Allen Ting who was always there in the lab to encourage me in my efforts and help retain my sanity with his kind words of courage during difficult times. Many Thanks to my friend Koen van Rompay, who helped in many ways during my stay in Davis and also proof read the first draft of this thesis. Thanks to my brother and wife for their love and support which I have come to take for granted. I'd be always thankful to my ever forgiving parents for tolerating all my tantrums and making me what I am with their loving guidance in spite of hardships.

Abstract

This thesis addresses the problem of computer worms in the modern Internet. A worm is a self-propagating computer program that is being increasingly and widely used to attack the Internet. 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 model of a computer worm and discusses in length the aspects involved in defending the Internet against a worm. It explores several techniques toward this end. It develops a life cycle model of worm defense, including prevention, prediction, detection and mitigation. It also discusses in detail about each of these techniques. It develops innovative models for each of these techniques and analyzes each one of them. Of primary interest are models that can automatically respond to a worm outbreak. Two such mitigating models, the 'friends' model' and the 'hierarchical model' have been developed and discussed in addition to a predicting model, 'TrendCenter'. It discusses the results of real time experiments conducted on the campus gateway for the 'TrendCenter' effort and the results of simulations of the mitigation models. It also discusses several preventive models that have been developed by the community. It concludes that worms are dangerous to the Internet but there are ways and means to mitigate their ill-effects.

Chapter 1

Introduction

These are days of networked computers. Their importance in world security in light of recent events related to terrorism is unprecedented. There is no need to belabor the potential havoc that a malicious hand can wreak on our lives upon gaining access to critical computer installations of the defense systems or the Internet, nor on the importance of securing such infrastructure from being compromised or misused.

One of the various ways in which computer systems can be compromised is by deploying a computer worm[1]. There have been instances in the past where worms have virtually brought the Internet to a grinding halt such as the classic "Internet worm" or the recent "Sapphire" worm.

This research primarily deals with stopping a worm on its tracks without human intervention. Several strategies have been proposed and analyzed with real world experiments where feasible and with simple simulations where a real world experiment is not possible owing to the nature of the subject.

This thesis starts off by providing a model of a simple worm and an extensive background about computer worms of the past, present and future in chapter 2. The chapter following that develops a life cycle model for the defense against worms. Chapter 4 presents various techniques used by worms to scan the Internet to find hosts susceptible

[1] Numerous word wars have been fought to define a worm. In this discussion we will consider a worm as any computer program which once started in a networked host can autonomously find and run on other machines.

to infection. Chapter 5 deals with hypothetical worms of the future. Chapter 6 analyzes various issues involved in manual as well as automatic models of worm defense. It also provides an overview of some of the tools and models that are currently available to help protect computer systems from attacks including computer worms and gives an introduction to the models of worm defense that are the focus of this thesis.

TrendCenter, a model to chart out the emerging threats and predict the next worm is presented in chapter 7. The *friends' model* and the *hierarchical model* of worm defense are discussed in the chapters following that. The last two chapters of this thesis present the conclusions and future directions of this research respectively.

1.1 Contribution of this Thesis to the field

This thesis provides a perspective of computer worms, explores the various worm technologies and popular 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 classification 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 security community.

It also develops and analyzes several indigenous and innovative techniques to address the problem of computer worms. It develops a predictive model, TrendCenter, and two mitigation models, the friends' model and the hierarchical 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 security.

Chapter 2

Background

2.1 Introduction

A computer worm is an extremely handy tool to do a particular task on several hosts. Unfortunately, it can also be used as a weapon. For example, consider a computing task that takes several days to finish on a single machine. It can be done much quicker if it can be broken down to several smaller and simpler sub-tasks that can be done in parallel on several machines. It is definitely painstaking to do it manually; particularly if we need to do it on a daily basis. We could design a parallel processing machine. But such a machine is usually very expensive and not very versatile. Instead of designing such a complex parallel processing machine, we can design a comparatively simple tool that can assign sub-tasks to capable idle machines and collect and compile the results. Such a tool is a worm.

When used properly the worm tries to hop on from one idle host to another carrying with it a sub-task in search of computing power to accomplish its tasks and return the results to the parent process that waits for the results in a different machine. A beautiful example is the worm program that Shoch and Hupp used at the Parc to make use of idle computing power of computers of several employees after regular office hours[22]. More on this worm in section 2.3.1.

The most important requirement for a worm to perform a sub-task on a idle machine is, obviously, permissions to execute programs on it. In cases where prior permissions

are granted on various machines, the task is simple. In cases where permissions are not granted the job owner may try to force his or her way into other people's computers.

One can force his or her way into a computer by exploiting any of the vulnerabilities that exist in it. When there is a wide-spread vulnerability on many hundreds of machines spread across the Internet, the Internet becomes a happy hunting ground for anyone that can exploit that particular vulnerability. A computer worm that can automatically hop on from one host to another to perform a task can also penetrate from host to host exploiting such a wide spread vulnerability with such penetration being its task.

Such an ability can transform from a very useful tool to a weapon that can cause wide-spread havoc and destruction. A very simple and recent example is the *Slammer worm* of 2003.

2.2 Model of a worm

Each computer worm is different in the sense that each of them uses different mechanisms to spread on the Internet. Some of them just spread on a particular targeted network as opposed to the Internet. There are several features or parameters that characterize a worm such as:

- Vulnerabilities exploited on victims
- Speed of spread
- Strategies of spread
- Payload of the worm
- Intended goals and unintended effects

Though each worm is a unique combination of all the above said parameters, a basic comprehensive structure can be assigned to any worm to understand its working. A worm consists of 4 components, Probe, Transporter, Worm Engine and Payload as shown in Fig.2.1.

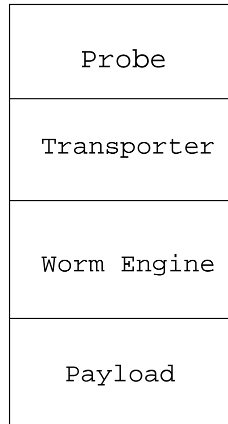

Figure 2.1: Components of a computer worm.

The probe is that part of the worm which looks for a susceptible host. This could be very simple query to a host inquiring if a particular service which the worm tries to exploit is running on it.

The transporter module of the worm is responsible for spreading the worm from one host to another. This contains the ability to establish a communication with the vulnerable host as reported by the 'probe' and exploit the vulnerability in it. Once the worm gains a foothold in the victim, this module transports the entire worm to the new host and starts up the worm engine. This is analogous to a boot strap loader in a operating system. A classic example of a transporter is the *Grappling Hook* used in the *Morris worm*[15].

The worm engine is the central command base of the worm. It dictates how the worm behaves. Once the transporter starts up the worm engine, the engine takes control of all the activities of the worm. It installs the worm in the new victim. It triggers the probe as and when required which is the first step in spreading the worm to another host. The worm engine can control the following traits of the worm giving the worm a unique character of its own.

- Frequency and timing of infection attempts
- Spreading strategy
- Limiting the number of worms in a single host

- Camouflage - to avoid detection eg. Frequent fork()s and opening a network port in *Morris worm.*
- Executing the payload of the worm

The above list is by no means exhaustive. Spreading strategy is a very interesting study in itself. Highly sophisticated worms can be built by appropriately selecting the next host to infect[28]. The current host itself could be chosen as the next host, thereby repeatedly infecting itself. This would cause the cpu to run several copies of the same worm program which competes with the other processes for cpu time. After a while, the other processes do not get enough cpu time and the performance of the host deteriorates severely and appears to stall. This doesn't serve the usual goal of a worm to spread far and wide and should generally be avoided in a worm.

The payload is the part of the worm that could contain malicious instructions. It could even remove all files in the hard disk if the worm has enough permissions for that. The worm engine has to execute the payload to achieve the intended malicious effects. The payload doesn't execute by itself. Ironically enough, all of the most damaging worms so far haven't contained any explicit malicious payloads.

In a C-like language a worm looks like:

```
worm()
{
    worm engine();
    probe();
    transport();
}

worm engine()
{
    install and initialize the worm();
    execute payload if any needs to be();
    choose the next host to probe();
    wait until desired time for the next probe();
}
```

2.3 Examples of worms

There have been numerous worms on the Internet. Every worm has its own unique characteristic. However, there are only a handful of worms that have caused any considerable concern in terms of disruption of services that are based on the Internet and the man-hours lost to fix the damages caused by them. Some of the most talked about worms are discussed in brief in the following sections. Each one of these worms have amazed security professionals for their novelty and their effects on the Internet and have been an eye opener to a new realm of the problem. A common feature amongst most worms that have caused the maximum damage is lack the of a malicious payload. They either gobbled all CPU time in their victims or network bandwidth saturation just by their rate of reproduction and spread.

2.3.1 Xerox Parc worm

Computer worms, as they are widely referred, were first reported by Shoch and Hupp at the Xerox Palo Alto Research Center(PARC) in the early 1980's after much ground work starting in the early 1970's. The worm programs were an experiment in the development of distributed computations: programs that span machine boundaries and also replicate themselves in idle machines during the night when nobody were using their machines. These worms would retreat as users reclaimed their machines in the morning. They were quite successfully used to support several real-time applications. An apparent corruption in the worm code during migration caused a cancerous growth of worms leaving about 100 machines dead because these worms tried to reproduce beyond what can be supported by the hosts. This incident at PARC helped show the dangerous potentials of a worm[22].

2.3.2 Morris worm

In the evening of November 2, 1988, the *Morris worm* was released upon the Internet. It invaded VAX and Sun-3 computers running versions of Berkeley UNIX. Within the space of hours this worm spread across the U.S, infecting thousands of computers and making many of them unusable due to the performance drain of the worm. Note again, this

worm didn't have a payload either.

The mode of attack consisted of locating vulnerable hosts (and accounts) to penetrate, exploiting security holes on them to pass across a copy of the worm and finally running the worm code. The worm obtained host addresses by examining the system tables */etc/hosts.equiv* and */.rhosts*, user files like *.forward* and *.rhosts*, dynamic routing information produced by the *netstat* program and randomly generating host addresses on local networks. It penetrated remote systems by exploiting the vulnerabilities in either the *finger daemon*, *sendmail*, or by *guessing passwords* of accounts and using the *rexec* and *rsh* remote command interpreter services to penetrate hosts that shared the account.

The worm also employed methods to cover its trails and adopted camouflage to evade detection. This was the first major incident which made computer scientists around the world to take a serious note of computer worms.[21, 24, 15]

2.3.3 Code Red and Nimda

Of the more recent worms, *Code Red I, II* and *Nimda* were the most reported. *Code Red* exploited a Microsoft IIS Web servers' vulnerability and is still active re-surging monthly, while *Nimda* did the same in addition to looking out for the back doors left behind by *Code Red II* and *Sadmind* worms.

The main difference amongst them was that Code Red I generated random IP addresses to infect while the other two used *local sub-net scanning* which was more effective.

Many firewalls allow mail to pass untouched, relying on the mail servers to remove malicious mails. Many mail servers remove them based on signatures. And signatures for unknown viruses and worms aren't generally available. Hence, such mail servers aren't effective in removing such malicious mails during the first few minutes to hours of an outbreak. This gave *Nimda* a reasonable time to spread. And Nimda's full functionality is still not known.[28]

2.3.4 Slammer

The Slammer worm[18], also known as the Sapphire Worm was the fastest computer worm in history. It began spreading throughout the Internet at about 5:30 UTC,

on January 25 2003, and doubled in spread every 8.5 seconds. It infected more than 90 percent of vulnerable hosts within 10 minutes. Although very high speed worms [27] were theoretically predicted about a year before the arrival of Slammer, it was the first live worm that came any closer to such predicted speeds.

Slammer exploited a buffer overflow vulnerability in computers on the Internet running Microsoft's SQL Server or MSDE 2000. This weakness in an underlying indexing service was discovered in July 2002. Microsoft released a patch for the vulnerability before it was announced. But many system administrators around the world didn't apply the patch for various reasons. The following were some of the reasons. The patch was more difficult to install than the original software itself. They were afraid that applying the patch might disturb their current server settings[1] and it was always not trivial to tune a software to the required settings and performance. Many just weren't ready to spend much time to fix this problem. Instead they were waiting for the next release to replace the entire software instead of applying patches. Some were just ignorant or lazy to apply patches.

Ironically, Slammer didn't even use any of the scanning techniques that were hypothesized in [28] to choose a worm's next victim. It was a worm that picked its next victim randomly [2]. It had a scanner of just 404 bytes including the UDP header in contrast to its predecessors Code Red that was 40KB and Nimda that was 60KB. The spread speed of Slammer was limited by the network bandwidth available to the victim. It was able to scan the network for susceptible hosts as fast as the compromised host could transmit packets and the network could deliver them. Since, a scan packet contains only 404 byte of data, an infected machine with a 100 Mb/s connection to the Internet could produce $\frac{100\times10^6}{404\times8} = 3\times10^4$ scans per second. The scanning was so aggressive that it quickly saturated the network and congested the network.

This was by far one of the most destructive worms whose ramifications rendered several ATMs unusable, canceled air flights and such. Note again, this worm didn't have a pay load. Fortunately, since the fateful day was a Sunday, the consequences weren't felt badly. And most services on the Internet were back to normal by the time the week started, at least in USA. However, the East Asian countries where the worm was first believed to be released did have some problems.

[1]This usually happens when one installs a patch

[2]Scanning techniques are discussed in chapter 4.

2.4 Summary

This chapter gave an introduction to a computer worm in general and developed a model for it. It described the various components that make up a worm. It also listed some of the features that could be used to characterize a worm. It discussed some of the most popular worms of the past. The Slammer worm was discussed in detail, explaining its aggressive scanning strategy and its self congestion, with the help of simple calculations.

Chapter 3

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. Fig. 3.1 gives a diagrammatic representation of the life cycle.

3.1 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.

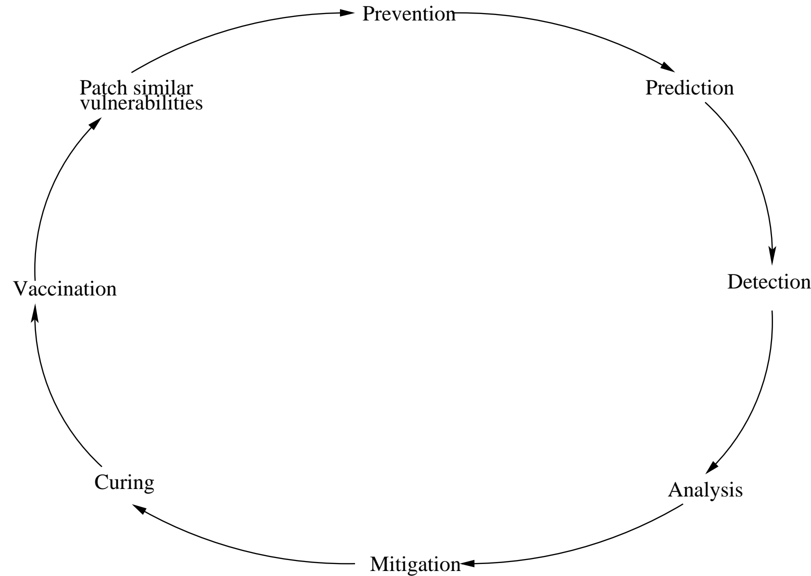

Figure 3.1: The various stages of the life cycle of worm defense.

3.2 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 'Friends' Model' [11] does this as a part of its mitigation strategy. The TrendCenter effort tries to predict the vulnerability that is most likely to be exploited in the near future. Chapter 7 of this thesis discusses TrendCenter in more detail.

3.3 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.

For example, the Morris worm[24] and slammer worm[18] consumed all cpu time at the host level and congested the network. There could be exceptions to the above characters. For example, the Slammer worm didn't use any TCP connections at all. Though the Morris worm and the Slammer worm, didn't have any malicious payload, they potentially could have had malicious payload that could have done a lot more damage. Other symptoms include but are not limited to disappearance of files and user accounts, ceasing of mail delivery, etc., depending on the payload of the worm. These could be detected using either manual or automatic means. But manual means don't promise to be effective to deal with such high speed worms [28].

Slow moving worms are harder to detect. They don't have any obvious effect on the network or the individual victims. These worms are an ideal means to plant time bombs, open back doors, etc., It needs a lot of correlation amongst various sites to suspect a worm-like activity. Fortunately, once we suspect something wrong is going on, we can stop it through manual means as we have enough time before all susceptible hosts are compromised by a slow worm.

3.4 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 blocking 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[28]. For example, Nimda is still not fully understood[28].

3.5 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. Two automatic response strategies, the *Friends Model* [11] and *Hierarchical control* [12] were developed during this research. These models are discussed in detail with results from simulations later in this thesis.

3.6 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.

3.7 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.

3.8 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.

3.9 Summary

This chapter developed a life cycle model of defense against computer worms. It showed how the fight against malicious computer worms is an on-going process without an end. It discussed in length, detection of worms. 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.

Chapter 4

Worm Scanning Techniques

The spread of active worms is generally limited by how quickly new potential hosts can be discovered[30]. Some authors [30, 28] discuss several scanning strategies like *hit-list scanning, permutation scanning, topological scanning, local sub-net scanning and Internet-sized hit-lists* that can increase a worm's virulence.

4.1 Topological Scanning

Topological scanning is a technique that uses information contained on the victim machine to select new targets. A popular example is an e-mail virus. It uses the e-mail addresses available in the address book of the victim host. Another classic example is the *Morris worm*, it made use of the entries in the *.rhosts* file to select new targets. This technique proved to work well.

4.2 Permutation Scanning

In a *permutation scan*, all worms share a common pseudo random permutation of the IP address space. Any machine infected during the hit-list[1] phase or local sub-

[1] See chapter 4.4. This phase is where the worm scans a given list of hosts.

net scanning starts scanning just after its point in the permutation, and scans through the permutation looking for vulnerable machines. Whenever it sees an already infected machine, it chooses a new, random start point and scans from there. Worms infected by permutation scanning would start at a random point.

This provides a semi-coordinated, comprehensive scan while maintaining the benefits of random probing and minimizing duplication of effort. After any particular copy of the worm sees several infected machines without discovering new vulnerable targets, the worm assumes that effectively complete infection has occurred and stops the scanning process. A timer could then induce the worms to wake up, change the starting point of this search to the next one in a pre-specified sequence, and begin scanning through the new permutation [30].

4.3 Local Sub-net Scanning

Local sub-net scanning has been used by the *Code Red II* and the *Nimda* worms.[30]. This involves scanning for vulnerable hosts in the class-C sub-net so as to infect all the hosts in a sub-net. This usually increases the number of infected machines quickly because once the worm has crossed the firewall or any kind of security system of an organization, it can easily quickly infect all the other vulnerable hosts behind that firewall as there are no host level firewalls usually within an organization.

4.4 Hit lists scanning

This is a scanning technique where the worm scans the network based on a list of hosts. This list is compiled by the attacker using slow and stealthy scans over a long period of time such as not to raise any suspicion or by even collecting information from publicly available resources. The period of time could easily be in the order of weeks or months to scan the entire Internet. This scanning technique can be used by extremely high speed worm like Flash worms[27]. However, by the time the attacker gathers information about all the vulnerable hosts slowly without showing up on any of the network monitors around the world, the vulnerability itself might be fixed.

4.5 Random Scanning

The worm itself generates random IP addresses to probe. This is a very simple and quite useful technique. For example, when a worm using *Topological scanning* finishes exploring all the hosts in the environment and is not able to get any new victims, it could use this technique to continue to spread. *Slammer*, the fastest spreading worm yet used this method. However, a bug in the Pseudo Random Number Generator of Slammer made the worm skip certain /16 blocks of addresses in a scanning cycle. It made it difficult for the Internet community to reassemble a list of compromised Internet addresses.

4.6 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* [18].

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 $100 \times 10^6/(400 \times 8) \approx 30,000$ 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.

4.7 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. It also presented simple calculations to estimate the scan rate of a worm.

Chapter 5

Future Worms

5.1 Warhol Worms

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. [28] calls such a worm a *Warhol Worm* and such a scanning technique *hit-list scanning*. It also talks about distributed control and update mechanisms by which such worms can be maintained and controlled on the Internet.

5.2 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. Reference [28] discusses more sophistications and concerns of this strategy.

References [27, 28] argues that a determined attacker could build a list of all or almost all servers with the relevant service open to the Internet in advance of the release of the worm. There are about 12M web servers on the Internet. The corresponding address list will be 48MB which can be sent over a T1 line in about 4 minutes and over an OC12 link in under a second **??**.

Such a worm, with an *Internet-sized hit-list*, starting from a site with high bandwidth links, could in principle infect all of the vulnerable machines on the Internet in under 30 seconds, and is termed a *Flash Worm*[27].

5.3 Stealth Worms

Though *Flash Worms* can attack with high speed such that no human mediated efforts could be of any use, we could devise automatic means of detecting and stopping them. But *stealth worms* spread much slower, evoke no peculiar communication pattern and spread in a fashion that makes detection hard[28]. Their goal is to spread to as many hosts as possible without being detected. However, once such a worm has been detected, manual means of mitigation are possible. These worms wouldn't cause any obvious damage to any system, for if they did, they would be detected easily. A more subtle use of such worms is:

- Open *back doors* for a future high speed worm. Even if a corresponding future worm is slow, it can spread quickly because it need not go through the entire process of infection mentioned in section 2.2 because they can use te back doors already opened by stealth worms.
- Plant *Trojan horses* in the infected systems to be triggered by a later incident.
- Install *'time bombs'* that all go off at the same time such that no salvation is possible.

Apparently innocuous, stealth worms are something to be extremely careful about. Though they spread slowly, immediate action is required upon discovering them to protect computer systems.

5.4 Polymorphic Worms

Any worm that changes its form or functionality as it propagates from machine to machine can be called a *Polymorphic Worm*, though there is no clear cut definition for it as yet[20, 32]. Generally, a *Polymorphic Worm* can be represented as an extension of figure 2.1 as shown in figure 5.1. The *Worm Engine* contains an additional module called

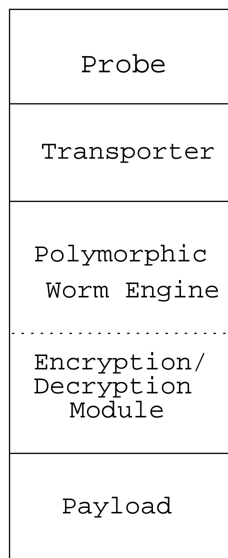

Figure 5.1: Components of a Polymorphic computer worm.

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. 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.

5.5 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[19]:

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.

5.6 Summary

This chapter gave a brief description of some of the hypothetical worms of the future. Some of the worms like Flash worms and Warhol worms could bring down the entire Internet in a matter of minutes. Such worms call for automatic mitigation techniques. The

other end of the spectrum is the Stealth worm, which spreads very slowly so as not to cause any noticeable surge in anomalous traffic that would attract the attention of system administrators. A polymorphic worm can change form with each copy, thereby evading signature based detectors. Finally, this chapter discussed e-mail worms.

Chapter 6

Models of Worm Defense

6.1 Manual Vs. Automatic Mitigation

There are several steps involved in our worm defense model. These include:

- Detecting that there is a worm outbreak.
- Using forensics to chart the *modus operandi* of the worm.
- Developing a patch or response for these exploits.
- Disseminating the patch or response.
- Applying the patch or response to various compromised and uncompromised hosts.

The difficulty of detecting a worm depends on the kind of worm. A very fast spreading worm shows up as spikes in traffic in various IDSs and can be detected by automatic means. A slow moving worm with subtle effects, for example, one that installs a time bomb or back doors to be used by later attacks, makes manual detection difficult.

An autopsy on the worm necessarily needs an expert's attention. However, we can infer certain trivial but useful information automatically. For example, we could determine the port to which an unusually high volume of traffic is sent and we could raise an alert or stop traffic automatically. However, not all worms are so simple, and blindly shutting off a

port could cause *denial of service* to legitimate users. In the later case, the cure could be worse than the malady.

A patch or the response to a worm could be distributed manually by physically delivering a CD to system administrators or by e-mail, or by posting it on the Internet, or on a newsgroup. Physical media are appropriate when the network might be choked due to the worm activity, and no patches are deliverable. A similar situation prevailed in the aftermath of the *Morris worm* outbreak in 1988. People were not able to download the patch from newsgroups because they had disconnected their computers from the network.

If the network is still usable, the response mechanisms could be delivered automatically. Also, these patches could be applied to the vulnerable hosts automatically. The issue of how these patches could be trusted is addressed in the chapter 8.

The above arguments have been summarized in the Table 6.1 for slow, medium and high speed worms for manual and automatic mitigation. A $\surd$, $\times$ and a blank indicates that the means is viable, unviable and undecidable respectively for the accomplishment of the concerned step for the mitigation of that particular worm kind.

	Slow worms		Medium speed worms		High Speed worms	
	Manual	Automatic	Manual	Automatic	Manual	Automatic
Detection	$\surd$		$\surd$	$\surd$	$\times$	$\surd$
Forensics	$\surd$	$\times$	$\surd$		$\times$	
Developing Patches	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\times$
Disseminating Patches	$\surd$	$\surd$		$\surd$	$\times$	$\surd$
Applying Patches	$\surd$	$\surd$		$\surd$	$\times$	$\surd$

Table 6.1: Summary of the arguments for manual vs. automatic mitigation techniques. $\surd$ - viable. $\times$ - unviable. Blank - dependent on the worm.

6.2 Means of Automatic Mitigation

Upon detection of the spread of a worm, an alert message or a trigger could be sent to a pre-built list of vulnerable, co-operating hosts. This trigger would put in place or invoke the necessary defensive mechanisms in machines that it could reach. The defensive mechanisms would include:

- Closing all unnecessary[1] out-going connections, perform egress filetering, so that the chances of worm spreading out from the infected machine is minimized.
- Filtering traffic with the worm signature at *routers* or *border gateways* (BGW) or any such place as deemed appropriate.
- Employing a *Sticky Honey-pot* by creating virtual machines using unused IP addresses in the local network[5]. This in effect is a DOS on the worm. This slows the spread of a worm and is an effective use of deception as a counter-offensive.

6.3 Detection vs. Declaration

A known worm's presence in a machine could be detected fairly easily. For example, the presence of a certain string of characters in the log file of a server indicates that a *Code Red* worm attempted to infect that server [10]. There are several commercial tools available to detect any unusual conditions on the network, and also to find and cure known worms and viruses. But no tool can detect an anomaly and declare with certainty that it is from a worm spreading unless the worm or anomaly are already known.

"How to ***declare*** a worm?" is a big question. Whenever there is any abnormal activity on the network, it could be a worm, a DDoS attack, a patch update to applications or just an innocent vagary of the Internet. The situation demands correlation of activities, symptoms and data at various places to determine if anomalous activity on the network is caused by a worm.

Once a *trigger* is released *"crying worm"*, all the alerted hosts would go into a defensive mode and start monitoring traffic, affecting normal throughput. Thus, there is a penalty in looking out for a worm. But the safety from a malicious worm outweigh the penalty. In the case of false positives, it is an unworthy price paid. A false negative is going to be very embarrassing and may be devastating. Thus, we have a situation of *perfect vs. imperfect detection* of a worm.

[1]The site managers or all participating hosts could collectively define "unnecessary".

6.4 Prevention

"Prevention is the better than remediation" applies here too as it does everywhere else. Some of the preventive steps include:

- Having the latest anti-virus software, sensible filtering(such as quarantining all executable content from e-mail and Web traffic), fire-walling, and rigourously applying patches.[29]

- Configuring the firewall properly, so that "all that is not explicitly allowed is forbidden."[30, 1]

- Including security by design[2], sensible defaults[3], [4], a diversity of operating systems in the network, usage of type-safe languages to write network services and quicker transition to IPv6. Due to its larger address space, the probability of a random scan finding a host using an IP address is small. The last can slow down the spread but not stop it completely.[31]

- Using techniques, which semantically prevent buffer overflows, such as safe-language dialects, buffer overflow analysis, non-executable stack and heap policies and Software Fault Isolation[5] are essential features of any defense. They prevent a common hole from developing.[30]

- Employing fine grained mandatory access controls. This ensures that every system call is authorized as a function of the running program, the user running the program and the context that invoked the program.[29]

- Evolving a protocol to isolate infected and compromised machines from the Internet and the internal network[30].

---

[2] UNIX wasn't designed with security as a goal.

[3] The DEBUG mode in sendmail exploited by the Morris Worm was turned on by default.

[4] Microsoft IIS is an amazingly vulnerable target for worms. But is still activated by default with Windows 2000 and it also provides a highly homogeneous target

[5] This technique creates a Java-like sandbox for dynamically loading arbitrary code in a language- neutral manner.

- We could use one or a combination of several security systems available to find vulnerabilities in the local computing environment. Some such systems are discussed in Section 6.5.

6.5 Preventive tools

This section gives a overview of some of the configuration and vulnerability analysis tools that can be used to identify security holes that could be exploited by worm authors. The original papers have further details.

6.5.1 Kuang

Kuang [8] is a rule based system that finds inconsistencies in the set of protection decisions made by the users and administrators of a UNIX system. It allows a system manager to perform a 'what-if' analysis of a protection configuration, and in this mode, it helps the manager make protection decisions. A typical use of Kuang is to see if access to the system is sufficient to become root. Given a set of initial privileges and an end goal, this system analysis the protection configuration and gives the sequence of steps by which the end goal can achieved, if it is possible.

The heart of the Kuang system is a set of rules that describe the Unix (BSD4.2) protection system from an attacker's point of view. Using these rules, the system perform a backward chaining search of the protection configuration.

This system does not find holes in the operating system. It finds security holes in the protection configuration that has been set up by the system managers at a site. This system can be easily modified to suit site-specific needs by modifying the rule set.

Its limitation include, the ability to run on single system only. It cannot find vulnerabilities at a network level. For example, it cannot find security holes arising out of trust relationships between hosts.

6.5.2 NetKuang

NetKuang[33] is an extension to Kuang. It runs on networks of computers using UNIX and can find vulnerabilities created by poor system configurations at a network level allowing attackers jump from one system to another. Vulnerabilities are discovered using a backwards goal-based search that is breadth-first on individual hosts and parallel when multiple hosts are checked.

Its limitations include finding only a single match, even though multiple paths may exist between the start and end privileges. The only way to discover multiple paths is the fix the known holes and run another search.

6.5.3 NOOSE

NOOSE(Networked Object-Oriented Security Examiner)[9] is a distributed vulnerability analysis system based on object modeling. It merges the functionality of host based and network-based scanners, storing the results into several object classes. It can collect vulnerabilities from a variety of sources, including outputs from other vulnerability analysis programs. It presents the vulnerability information as an integrated database, showing how vulnerabilities may be combined into chains across multiple accounts and systems.

The information is presented in an object-oriented “spreadsheet” format, allowing the security manager to explore vulnerabilities as desired. Once complete, the security manager can explore the vulnerability analysis interactively, showing both what a selected account can attack, and who can attack a selected account. It can also intelligently verify the installation of security patches, dynamically installing missing patches.

6.5.4 TrendCenter - Predict the next worm

TrendCenter[6] believes that stand-alone IDS approaches and ‘detect and respond’ models to counter computer worms are ineffective. It applies a ‘predict and prepare’ model. It collects sanitized information from several IDSes and correlates this information to predict the vulnerability that will be exploited in the next worm in the nearest future. This is discussed in greater detail in chapter 7.

6.5.5 Other systems

This section listed a few of the many systems or tools available to verify the integrity or effectiveness of the security measures of a computer system, stand-alone and networked. Some of the other such systems include COPS(Computer Oracle Password and Security System)[16], SATAN(Security Administrator's tool for analyzing networks)[13], tripwire[17], ISS**??** and Nessus**??** that scan a computer to look for a list of known vulnerabilities.

Websites such as DShield.org and incidents.org list what they consider the top 10 vulnerabilities across the Internet. These lists of vulnerabilities would be the ones of choice for a system administrators to fix in a network. In addition to these lists, these sites also give several top 10 lists like the top 10 IP addresses from which attacks are launched (if you wish to block traffic from such offenders) and the top 10 ports being probed (which gives an idea about which services are being targeted for attack).

6.6 Mitigation Models

Two models of worm mitigation are the Friends and Hierarchical models. The following sections give a brief introduction to them. Each of these two models are discussed in great detail in separate chapters 8 and 9 respectively.

6.6.1 Friends Model

This model is based on a co-operative effort of participating members according to a pre-arranged protocol. Each host has a friendly peer relationship with other hosts with a certain amount of trust. Several hosts together are headed by a router. Each router in turn maintains a peer relationship with the other routers. Participating members periodically exchange messages about the current threat scenario at their sites with their peers. The threat scenario at a particular site is a function of the amount of suspicious traffic seen at that site as well as at its peer sites. The alerts from friendly sites are weighted commensurate to the trust the friend enjoys. Each site takes actions based on the overall threat perceived. See chapter 8 for more details.

6.6.2 Hierarchical Model

In this model, vulnerable hosts form the leaves of a tree structure and the internal nodes are control structures immune to infection. There are sensors at the leaf level which raise alerts upon seeing suspicious traffic. These alerts are propagated up the tree structure. Once the number of alerts received by a control structure crosses a threshold, instructions are sent down the tree to the leaves to filter suspicious traffic. This phenomenon is carried on to higher levels of the tree. Chapter 9 delves deeper into this model.

6.7 Summary

This chapter discussed the aspects of manual and automatic mitigation of worm effects. It made a comparison of these two techniques against worms of low, medium and high speeds. It listed a few means of implementing automatic mitigation and discerned the difference between detection and declaration. It listed a few steps involved in prevention of a worm outbreak. A number of tools that help identifying configuration vulnerabilities were introduced and discussed. A worm prediction model, TrendCenter, and two worm defense models, the friends model and the hierarchical model were introduced.

Chapter 7

Worm Prediction - TrendCenter

7.1 Introduction

When 99 sites are attacked by a certain attack tool, will the 100th vulnerable site be notified before it is attacked? This is the kind of question that TrendCenter [1] tries to address. Thousands of attacks are reported each year and a majority of attacks are against known vulnerabilities and *zero-day*[2] attacks are very uncommon. A recent example is the Slammer worm, which exploited a buffer overflow vulnerability in Microsoft's SQL server. This vulnerability was known for about 6 months before the attack, and a patch was freely available on the Internet. This is one of the principal ideas behind TrendCenter.

TrendCenter digests large amounts of data collected from various participating IDSes and predicts the vulnerabilities that the next worm would exploit. It also provides its own top-10 lists like 'Top-10 vulnerabilities', 'Top-10 offending IP addresses', and other top-10 lists on its web-site http://AttackTrends.org.

7.2 Goals

TrendCenter has three basic goals:

[1] My task was to sanitize the data collected from IDSes.
[2] Zero-day attacks are those that exploit vulnerabilities that are not yet publicly known.

Measure the Threat Environment: Lots of vulnerabilities are reported periodically but only a few of them are exploited to any great extent. TrendCenter's goal is to identify the handful as precisely and as quickly as possible. Suppose we have 100 sites contributing data. After the first 5 have been hit by a particular attack, we want to alert the remaining 95 before they are attacked, with the least number of false positives and false negatives. One of the goals is to balance speed and accuracy of the identification of the attack.

Detect Emerging Threats: Current IDSs are designed to detect known attacks against known vulnerabilities. However, TrendCenter wants to detect emerging threats and unknown attacks against both known and unknown vulnerabilities. There is no definite method of doing this. One approach is to analyze scanning information. If a port is being scanned that is unrelated to any known vulnerability, then we could speculate that a new attack is circulating and look out for more details[3].

Disseminating the knowledge: Once we have a sense of the threat out in the Internet, we need to communicate that knowledge to the people who can actually change the environment. These people include the network administrators and security officers of various organizations and institutions as well as interested individuals. These people could tune firewalls, apply necessary patches or turn off unnecessary services. An effective way of doing this is discussed in section 7.4. The result would be a very specific set of action items like, 'patch vulnerability CVE-2003-2222 on host kanab, atlantis and padme. For details click <here>. For patch and patch installation directions click <here>'. Since most of the vulnerabilities exploited are already known and patches available, the action item would invariably be something like, 'run auto update tool on hosts kanab and padme'.

7.3 Motivation

In the early days before computer security was widely practised, actual attacks were extremely rare. Few vulnerabilities were known. Few important systems were con-

[3]The following is a verbatim quote of Todd Heberlein's, a pioneer of the TrendCenter idea, experience: "For example, about 2-3 years ago CERT announced a new vulnerability in FTP servers. When I looked back over the umbc.edu data (just a single site), I could clearly see a steady rise in probes to port 21 for 4 weeks prior to the CERT alert."

nected to networks and networks were themselves very small. Systems and processes' interaction were small and slow. Manual, usually insider attacks were the order of the day.

In such a scenario, stand alone intrusion detection systems were effective, though not efficient or sufficient, to tackle computer attacks. But today's environment is completely different. Every point mentioned in the previous paragraph has moved to the diametrically opposite pole today.

The current mean-time between attacks per system is in the order of minutes, if not in seconds. Thousands of vulnerabilities are known. Several critical infrastructure rely on the Internet. The Internet is very widely spread and almost everyone in the developed world has access to it. Network applications are complicated, and attacks are automated and originate from all parts of the world. It is impossible to maintain systems completely secure at all times manually due to sheer volume of work involved discounting the ingenuity required to achieve this goal. Whether we can do it automatically has to be explored. Configuration vulnerabilities creep in every day because of the complex interactions between various components of a security system are difficult to understand[8]. Manual attacks are rare and automated attacks are common. Automated attacks include worms, sweeps and viruses.

In such a changed environment, stand alone IDSes are anachronous. Moreover, IDSes can only detect an intrusion, they cannot predict an intrusion. TrendCenter believes that this kind of a *'Detect and Respond'* model is no longer useful. Nevertheless, there are several thousands of IDSes deployed at various places all over the world with a majority of them operating independently. Correlating the vast amounts of data thus collected could give us valuable information about future attacks.

TrendCenter advocates a paradigm shift from *'Detect and Respond'* to *'Predict and Prepare'*. It tries to correlate intrusion data from various IDSes being deployed all around the world, to predict the next worm. Huge amounts of data collected from IDSs is also available at http://incidents.org. But the data provided is highly sanitized and therefore is not very meaningful for TrendCenter's research to predict a worm.

These data consist of only the port number being probed. Unfortunately this does not help us identify the actual vulnerability being exploited. For example, there are many known vulnerabilities in web services. Just saying "port 80 was probed" is not sufficient.

We need to say, "Vulnerability XYZ in a PHP scripting engine is being attacked. Details about the vulnerability can be found <here>. Patches can be found <here>. Instructions can be found <here>."

Moreover, we are concerned about long-term commitment for data supply. Hence, we began our own effort instead of depending on third party data.

7.4 The Process

Figure 7.1 shows a schematic representation of the TrendCenter model. Components above the dotted horizontal line form the controlled AttackTrends.org's area. And those below that form the uncontrolled volunteer's area. Users will interact with AttackTrends through two points:

1. uploading sanitized log files.

2. downloading information about vulnerabilities being commonly exploited.

The process consists of volunteers submitting sanitized intrusion data from their network to the central database at http://www.AttackTrends.org. AttackTrends currently accepts data collected using SNORT[4][4].

The tool, *ats* (AttackTrends Sanitizer), to sanitize the raw data and upload the sanitized data, is also provided at http://www.AttackTrends.org for data submitters. *ats* uploads the sanitized files using *curl*. The *ats* program uses a configuration directory that keeps information about:

a. how to run the program, for example, the location of the log files, the username and password for uploading the files.

b. the past runs of the program.

Ideally, *ats* will be launched every day by a *cron* job.

The computer at the back-end periodically processes the database and suggests a vulnerability that the next worm is expected to exploit.

[4]SNORT is an open-source intrusion detection system.

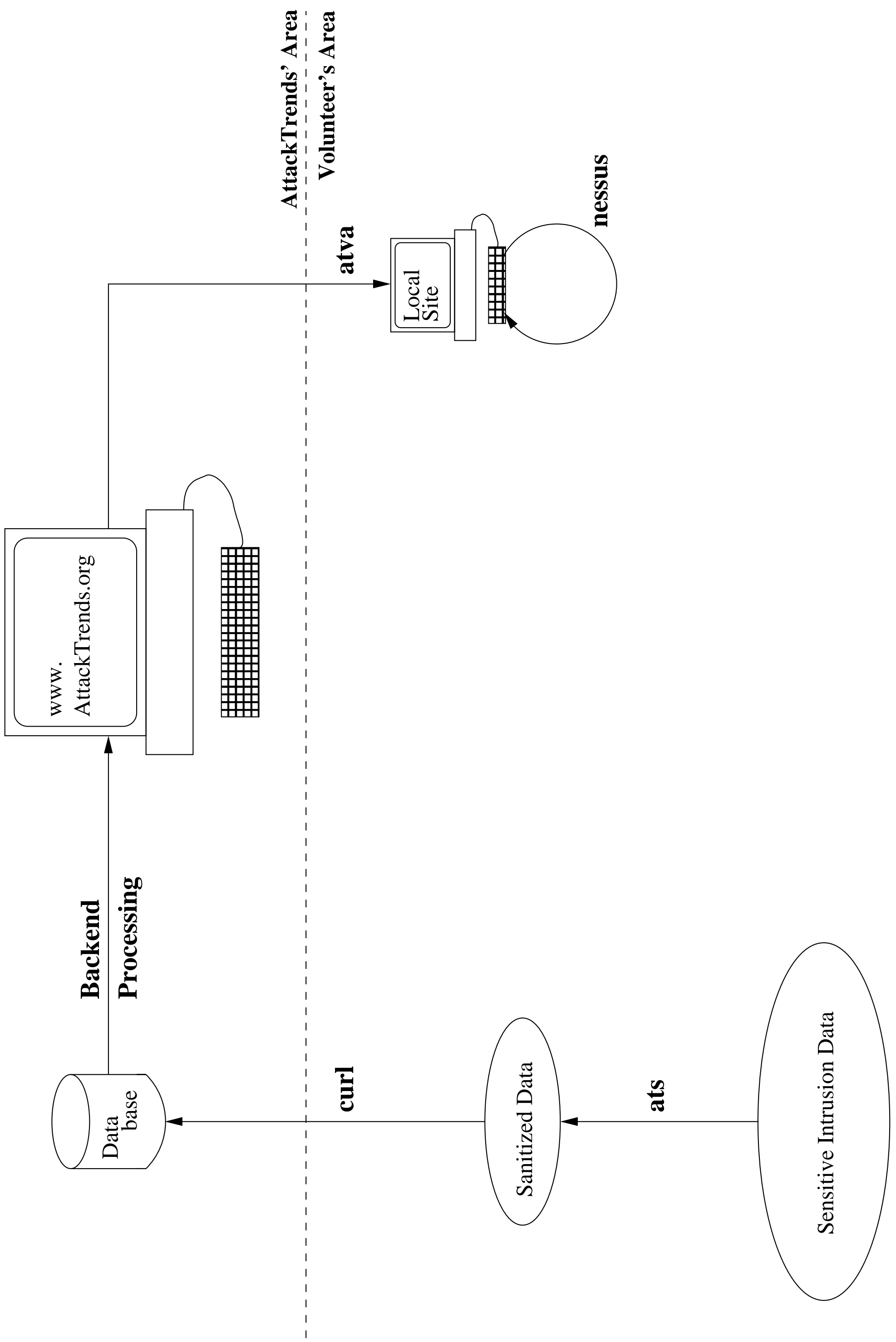

Figure 7.1: A schematic representation of TrendCenter

Optionally, subscribers can also install the *atva* (AttackTrends Vulnerability Analyzer) program. This program does the following:

- downloads the latest vulnerability information (via curl) from www.AttackTrends.org provided in XML format that can be processed by another computer program.
- determines which vulnerabilities are applicable to the subscriber's environment.
- automatically scans for those vulnerabilities in the subscriber's network using Nessus[5][3].
- provides relevant and specific information identifying which vulnerabilities on which systems will probably be exploited in the next 72 hours.

The relevant information includes the CVE-ID**??** of the vulnerability, the URL from which the patch can be downloaded, and the order of importance of applying each of the patches. A prioritized list helps optimize the system administrators time in keeping a network fairly secure.

For many, it might be difficult to manually sanitize the log files and transfer the sanitized files to the server and then run the vulnerability scanner daily. To address this problem, the entire process could be automated using a cron job that would:

- rotate the log files.
- run the sanitizer[6](*ats*).
- use curl to upload the sanitized file.
- run *atva* that gives a list of top patches to be applied.
- optionally even apply those important patches.

7.4.1 Sanitizing Intrusion Data

Everybody is sensitive with the data that go into and out of their network. Consequently, people are not eager to upload their Intrusion data. In particular people definitely

[5]Nessus is an open-source vulnerability scanner.
[6]Details about the sanitizer are given in the next section.

don't want anyone to know that their systems are attacking others, irrespective of the reasons. If it happens to be a commercial site like a bank, they don't want anyone to know that their systems are attacked either. Or they may not want to reveal their internal configuration.

Hence, a sanitizer tool must extract only the minimum amount of information required to make a meaningful analysis. Also no sensitive information should be collected from the raw intrusion data. *ats* was written using Perl, for its pattern matching strength.

`ats` collects the following information from each record of the SNORT log file:

- time the alert was logged into the system.
- the snort-ID for the vulnerability.
- source IP address.
- destination IP address.
- the destination port number.

Since the source IP address is a matter of high sensitivity if it is internal to the network, it can be optionally masked. The destination IP address is collected only for processing. It doesn't appear on the final sanitized data. The manual for the `ats` program is given in appendix A.

Figure. 7.2 shows two scenarios. The scenario on the left corresponds to one where the intrusion data is not sanitized as is currently done at some places [7]. The scenario on the right is the TrendCenter's way of doing it using `ats`.

This sanitation reduces the size of the file to be transferred to the AttackTrends' database dramatically by removing unnecessariy fields and by summarizing several lines of information into one. For example, a file with 583,656 lines was sanitized and summarized into 17,573 lines which is about 3% of the original events. This gives an indication of how many attacks are repeated. In terms of the file size, the original size was about 350MB which was sanitized and summarized to 630KB or 0.2% of the original size which further shrunk to 106KB or 0.03% of original size upon compression.

[7] This is the model at the US Air Force currently.

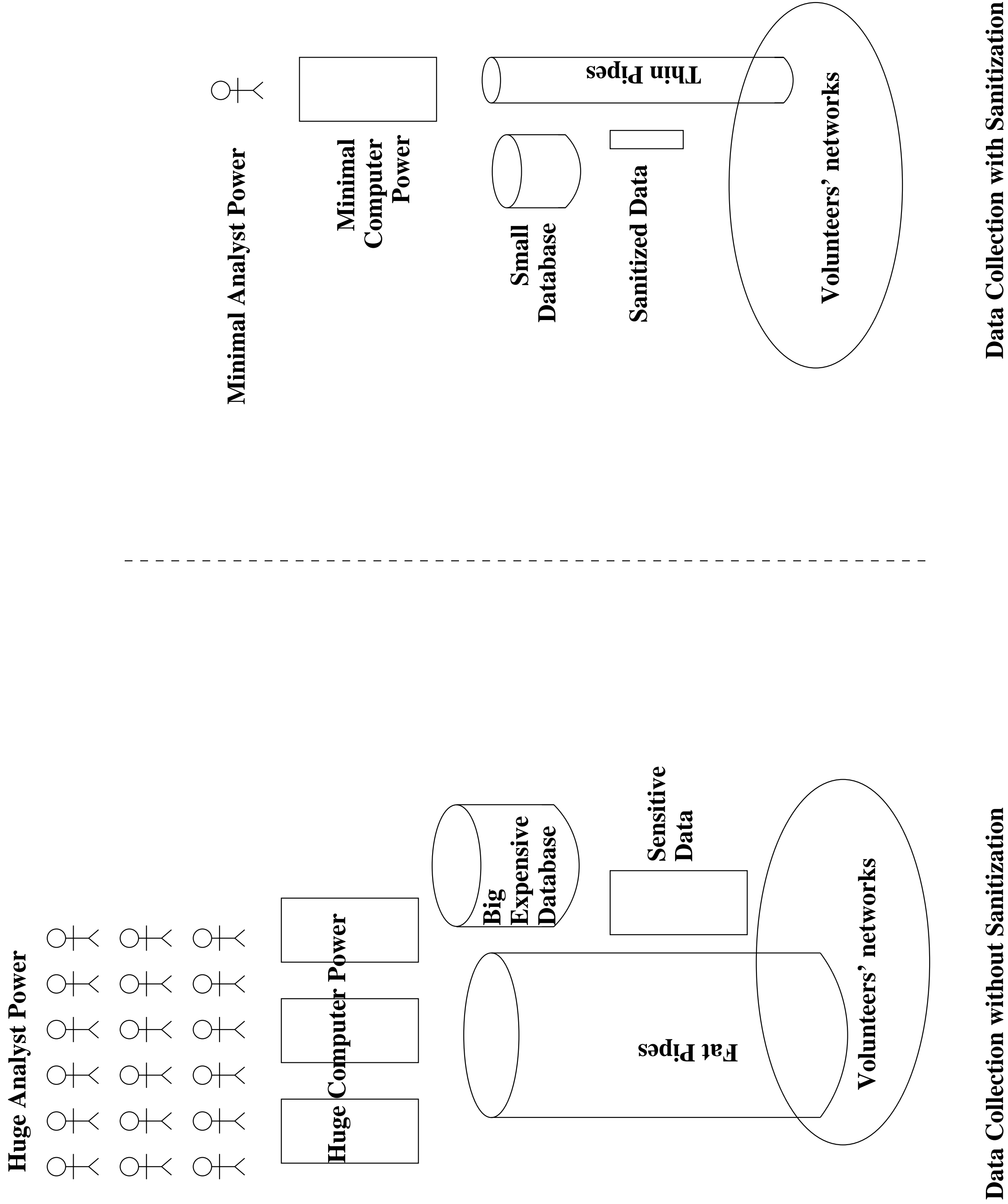

Figure 7.2: A diagrammatic representation of the advantages of sanitizing the intrusion data before uploading to AttackTrends.

Sanitation helps protect sensitive information and also makes it easy to transfer meaningful data to the database. 350MB of original raw data is not trivial to transfer for many. After sanitization the server doesn't have to handle huge amounts of data. And we need only a very small number of analysts at the back-end compared to the model showed in figure 7.2.

7.5 Discussion of the results

We deployed the SNORT sensor at the UCDavis campus backbone router and collected logs over a period of 3 weeks. We sanitized the data thus collected. We tried to look at the sanitized logs from several viewpoints. Some of them were the number of unique IP addresses launching attacks each day for the top 10 attacks and the number of attacks received by our campus each day.

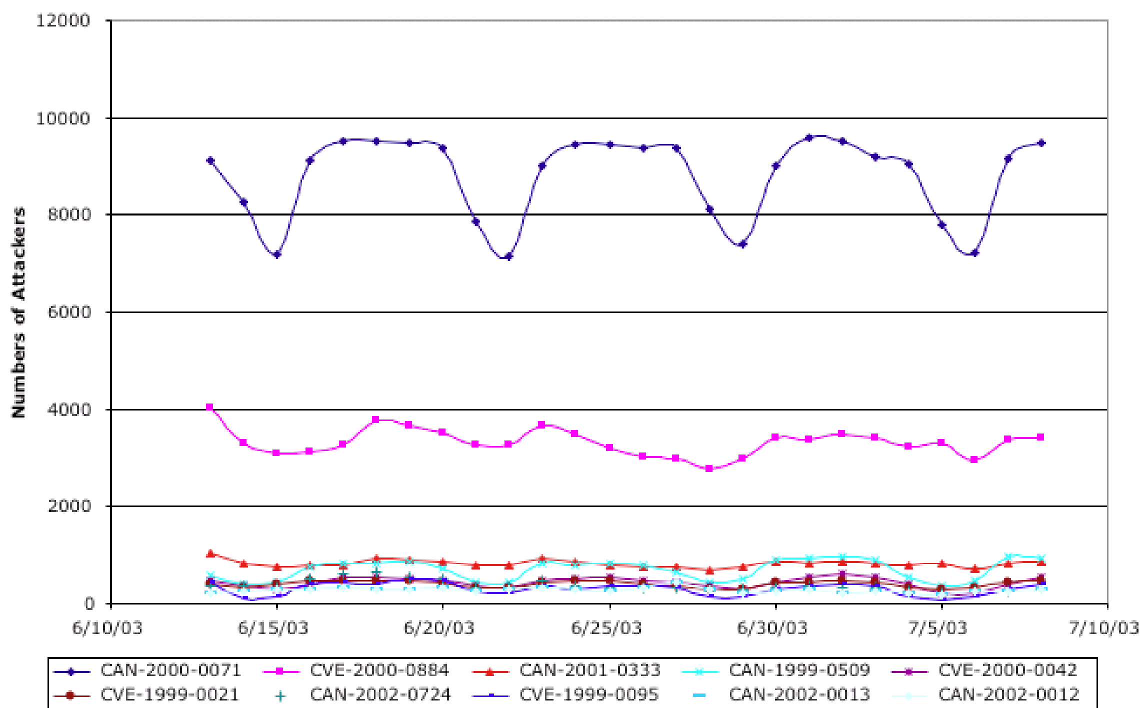

Figure 7.3: A snapshot of the numbers of the top 10 attacks received at the UCDavis campus.

The most obvious interesting trend was that the number attacks were fewer during the weekends. This is shown in Figure 7.3. This may be due to companies shutting down their machines during weekends. Incidentally, this practice also shuts down compromised machines in those networks. This is a strong indicator of a particular vulnerability on several hosts being exploited throughout the Internet. During business days, these compromised machines look for other victims even while their system administrators are unaware of the infection or compromise.

Another conclusion from Figure 7.3 is that the number of attacks that exploited the top vulnerability is much higher than the next top 9 attacks. This indicates that the vulnerability with ID CAN-2000-0071 needs to be patched immediately, followed by the others. Thus patching systems that have this vulnerability helps us secure a majority of hosts easily.

7.6 Issues

We do have some issues. The biggest issue is the quality of data we receive. It is an issue because the data submitters will be from a varied background, including malicious contributors. The sensors that volunteers employ might not have the latest signatures included. Thus they might not be able to capture some of the latest and most popular attacks. These problems will skew our results. How often the volunteers' sensors raise false alarms is unknown. Data with a large number of false alarms, will not give a correct analysis of the threat picture. Our back-end computer maps the snort alert ID to a CVE ID. Sometimes, the volunteers' logs may have snort IDs that don't match any of the CVE IDs. In those cases, the data we have is useless.

7.7 Future Directions

As a future project, the sensors could be improved to contain better and more signatures, and more CVE mappings could be developed. Sanitizers for logs generated by IDSs other than SNORT could be developed. The system could be improved to detect new attacks like systematic anomalies. These are anomalies that occur so regularly that the IDS

thinks that they are not anomalies. Deeper analysis including newer and better ways of interpreting the logs need to be developed. The TrendCenter idea could be packaged and sold by giving custom reports to paying subscribers. Dynamic interfaces supported by SQL in the server could provide flexible queries of incidents.

Presentation and interpretation of the statistics can be always improvised irrespective of the improvisation already made. For example, we could provide statistics like the top clustered IP address range of attackers. This would help in having fewer rules in the defending firewalls. Offenders could be aggressively ranked; for example based on recidivism. These could be graphically represented using a colour scale for varying degree of recidivism. This could be used to find correlation patterns. This data could be presented in various window sizes of a day, week and month. Graphical representation of certain incidents on the lines of `xtraceroute` would not be incongruous. Most relevant CERT contacts, cyber laws and cyber law enforcement contacts could be added to the web-site.

The contributors base needs to be expanded and is one of the most important issue. For example, out of 100 contributors, 99 could be home users and one could be a class B network administrator. But the data from this one class B network could completely dominate the analysis and provide a picture different from the actual threat environment on the Internet. Also, the type of contributor site - home, business, military, government, academic will affect analysis. Techniques should be developed to avoid skewing of results due to these differences in the nature of environment from where the data is collected.

7.8 Summary and Conclusion

This chapter described the TrendCenter project that provided an innovative approach to predict the occurrence of the next worm and the vulnerability that would most probably be exploited. TrendCenter aims to measure the threat environment, detect emerging threats and disseminate the knowledge by analyzing alert logs collected from various IDSs. TrendCenter optimizes the system administrator's time spent for securing his or her system by providing a priority list of patches to be applied to hosts in his or her environment. And it helps minimize cost of system maintenance in terms of man power as well as computing power. TrendCenter's governing philosophy is that standalone IDSes are

no longer useful by themselves but a collective summary of the alerts from many different IDSs may be useful in preparing for a future attack. The *'Detect and Respond'* is dead and *'Predict and Prepare'* strategy is the way of the day.

Chapter 8

Friends Model of Worm Defense

8.1 Introduction

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

Once a worm is detected, an alert message is spread to the set of participating hosts 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 depending on the detector's ability to reach other hosts and other factors discussed 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 Assumption

We assume the following for the sake of simplicity:

- The worm is not polymorphic, and its signature is known in advance or it is identified in real time.

8.2.2 The Worm Race

All participating hosts have a peer-to-peer and a hierarchical relationship with other hosts as in the real Internet. For eg., Border Gateways are higher in the hierarchy than routers while all routers and all BGWs have a peer-to-peer relationship amongst themselves respectively.

Each participating host has a list of trusted hosts[1] with associated trust-worthiness. By default, the trust is relative to the position in the hierarchy. The trust-worthiness of a host higher in the hierarchy is higher than that of a peer. This list need not include all the participants.

Once a host suspects a worm, it sends the suspected signature and an associated *perceived threat*, PT, to its friendly or trusted hosts. How a worm is suspected, detected or declared has been discussed in chapter 6. Such alerts received, the trust-worthiness of the alerting hosts and the actual number of suspicious packets seen by the host form the inputs to calculate PT.

$$\begin{aligned}\text{Perceived Threat, } PT \quad = \quad & f(\text{\# of alerts received, trust-worthiness of the alerting hosts,} \\ & \text{\# of malicious packets seen}) \qquad (8.1)\end{aligned}$$

f is a function that is decided by the participant to calculate PT. If the PT exceeds a certain pre-determined threshold and if any other criteria that the host has determined should be met, are satisfied, then that host takes actions. An example of a criteria that a host can require to be met is that any alert must be received from a certain number of different sources, each of which have a certain level of trust relationship with this host. This reduces the number of spurious alerts originating from malicious or *already compromised* hosts.

The list of friends is not static. It can be changed on a periodic basis based on the veracity of the previous alerts from each of the them automatically or by the site

[1]The terms 'trusted hosts' and 'friends' are used interchangeably.

administrator. The protocol to be followed while updating the friends lists will be the same as in updating routing tables to avoid race conditions and other inconsistencies that are possible when data at different locations have to be updated independently. Of particular interest would be the situation where one or more of the participants are left out of the friends lists of all the other participants due to race conditions. This situation should be avoided.

The actions taken are decided by the hosts individually unless there is an understanding with other hosts to take a collective decision about the actions to be undertaken. The iterative actions taken by each host include:

1. Assigning a threat level as perceived by the alert-receiver based on various parameters as mentioned above. This PT may be different from the PT seen by others.

2. Alerting its *friends.* Depending on the PT, a host chooses a different number of *friends* to alert.

3. Scanning the incoming and outgoing packets for the new signature. The intensity of scanning depends on the PT at the host. Based on the results of scanning, the PT at that host might increase or decrease. The intensity of scanning is the sampling rate of the traffic.

4. If the PT changes based on scanning, sending new alerts with the new PT. This acts like a control mechanism to dynamically increase or decrease the scanning intensity.

5. Reducing the bandwidth available to the general traffic and increasing the bandwidth to the alert messages. This prevents the worm traffic to occupy all available bandwidth that would prevent alert messages from propagating. This is possible because the hosts can control the traffic speed passing through it.

6. Blocking of traffic that is believed to be malicious.

7. Backing off from blocking the allegedly malicious traffic once it is found not to be so. This is achieved *automatically* since the intensity of the scanning decreases if there is a decrease in PT.

This spread of the alert messages may also be seen as a worm, but benign. Hence the topic *Worm Race*. One should also note that since we, the defenders, know the targets for our alerts, we can spread the benign worm much faster than the malicious worm after the initial latency of detecting a worm and extracting its signature.

The defending hosts might include backbone switches, routers, border gateways or gateways at universities. The higher in the network hierarchy that we deploy our scanning, the more costly is the process because of the high volume of traffic. But we can stop the worm at a much earlier stage and can also spread the alert message quickly as there are fewer targets for our alerts.

Note that if the worm has already crossed a host, a router for example, and entered the network below it, the hosts in the sub-net are open to attack. So, we need to have the detectors at various levels of the Internet hierarchy, not just at one level. For example, both at border gateways and routers; not just at border gateways. We should also not increase the redundancy too much lest we inflict a self DoS where a majority of the machines end up just scanning the traffic passing through it. We need to choose an optimal redundancy level to avoid such single point failures.

Currently this model proposes scanning at 2 levels, one at the BGW level and the other at the routers, one level of hierarchy below BGWs. Fig.8.1 shows the hierarchical relationship among the BGWs, routers and hosts.

While scanning incoming traffic prevents infection, scanning of out-going traffic prevents the spread of the worm and helps in quarantining the infected sub-net.

8.3 Mathematical Models

Mathematical models for the spread of worms are very similar to the epidemiological spread of diseases. In both the cases, the rate of spread of victims is proportional to both the number of victims and the number of susceptible hosts available for further infection. Thus, if a is the fraction of infected hosts, the rate of spread of infection is given by:

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

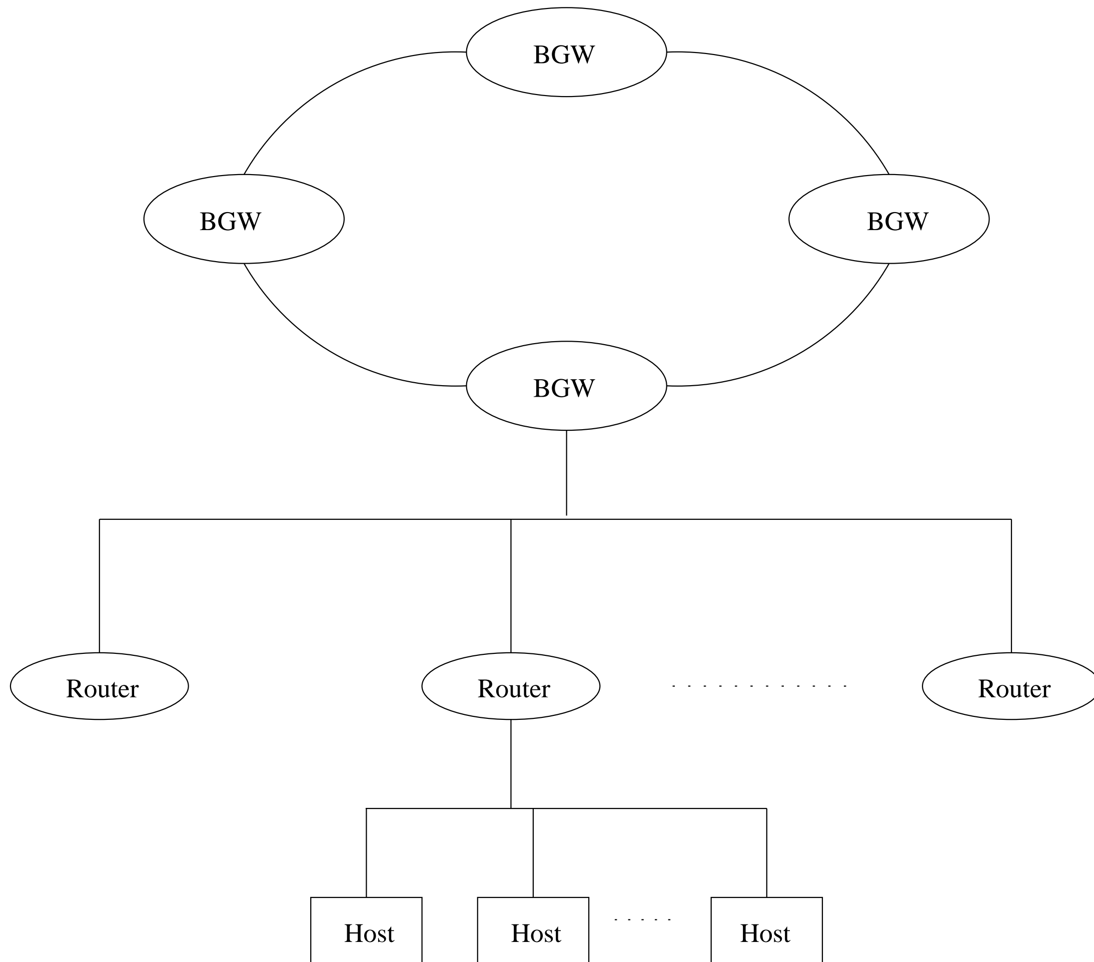

Figure 8.1: Hierarchical relationship among BGWs, Routers and Hosts. Ordinary hosts are one level below routers which are one level below BGWs.

Introducing a constant of proportionality, K, which here is the same as the rate at which each infected host can find and infect other hosts, we get,

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

Solving this differential equation, we get,

$$a = \frac{e^{K(t-T)}}{1+e^{K(t-T)}}$$

where T is a constant that fixes the time position of the incident. The equation produces the S-curve growth behaviour typically seen in population growth models and is known as

the *logistic* equation. This is the same equation that governs the rate of growth of epidemics in finite systems entities.

This is an interesting equation. For early $t (t \ll T)$, a grows exponentially. For large $t (t \gg T)$, 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 [28].

Mathematics of the Friends' Model The rate of spread of worm for the *Friends' model* described in this chapter has been developed in [14]. It is presented here in detail for the sake of completeness. The variables are defined as follows:

M : the total number of response members (eg. routers)

a : the number of infected members

c : the proportion of alerted members

F : the # of friends for each host

α : the # of alerts a host needs before it crosses the threshold

σ : the severity of the alert

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

$$F.(1 - c)$$

In our simple strategy, a cooperating member increases the severity of an alert in proportion to the number of infection attempts that it sees. Therefore, the number of alerts a particular member sends in time dt is:

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

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 α alerts before it can change its state, the number of members changing state in time dt is given by:

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

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

$$\frac{dc}{dt} = \frac{F.(1-c).\sigma a}{\alpha} \tag{8.3}$$

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

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

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 $1/M$ and behind another router is $1-1/M$. Combining these probabilities and the infection rates, equation (8.2) becomes:

$$\frac{da}{dt} = aK(1-a)(1-c)^2.\left(1-\frac{1}{M}\right) + aK(1-a).\frac{1}{M} \tag{8.4}$$

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

Dynamic response strategy with back off mechanism According to our model, hosts back off from filtering the traffic after a certain time period. The rate of back off is inversely proportional to PT meaning, it is directly proportional to $(1-a)$ and also to the proportion of alerted members, c. Thus equation 8.3 becomes

$$\frac{dc}{dt} = \frac{F.(1-c).\sigma a}{\alpha} - \epsilon(1-a)c \tag{8.5}$$

where ϵ is a constant indicating how fast a host backs-off from filtering traffic. Readers are referred to [14] for details.

8.4 Description of the Simulation

The worm defense model was simulated using the SWARM simulator. SWARM is a software package for multi-agent simulation of complex systems like population dynamics and simple social behaviour in biological organisms [2]. SWARM was chosen for its ease of programming and its active developmental support.

We consider a rectangular grid of x by y routers each having 8 neighbouring hosts connected to it. There are two kinds of routers and hosts. One kind that is infected with a worm and the other alerted by the white worm.

The simulation begins by starting the infection from random hosts. An arbitrary host is then marked as the initial detector of the infection. This host sends an alert to 8 of its friends. These friends are chosen randomly for the purposes of this simulation. The alerted hosts then calculate the threat that they perceive, PT, according to equation. 8.1. If the PT that each of these friends calculate exceeds a threshold and if certain conditions that are required to be satisfied by that host are met, then each of them also takes actions which may include alerting their friends in turn. Thus the alert message propagates from one host to another. This alert, the *white worm,*[2] spreads until all routers are alerted. Once all routers are alerted and start scanning traffic, the worm can no longer spread, thereby trapping the worm within those domains which are already infected. Once the worm stops spreading, the routers in whose sub-net there is no infection *back off* from scanning the traffic, while routers with infected sub-net continue to scan traffic. This reduces the number of routers that scan the traffic to the minimum required to stop the worm spread. This reduces the cost of blocking worms to the minimum required.

8.5 Discussion of the results

The curves for a typical scenario with 8 friends for each host are shown in Fig.8.2. Initially when the worm starts spreading, none of the hosts are aware of it. But once one host raises an alarm, a large number of the participants are alerted in a cascading fashion and traffic is monitored. The graph shows that the alert messages spread much faster than

[2]Term first seen in [31]

the worm and thereby triggers the filter rules at a large number of places before the worm could reach them. We can see that the maximum infection is restricted within 25% of the population. The drop in the infected population is due to the patching of machines against the worm. We can change various parameters such as the number of friends each host has, the spread speed of the worm, the network scanning technique the worm uses, etc., in the simulation to study the effectiveness of our model.

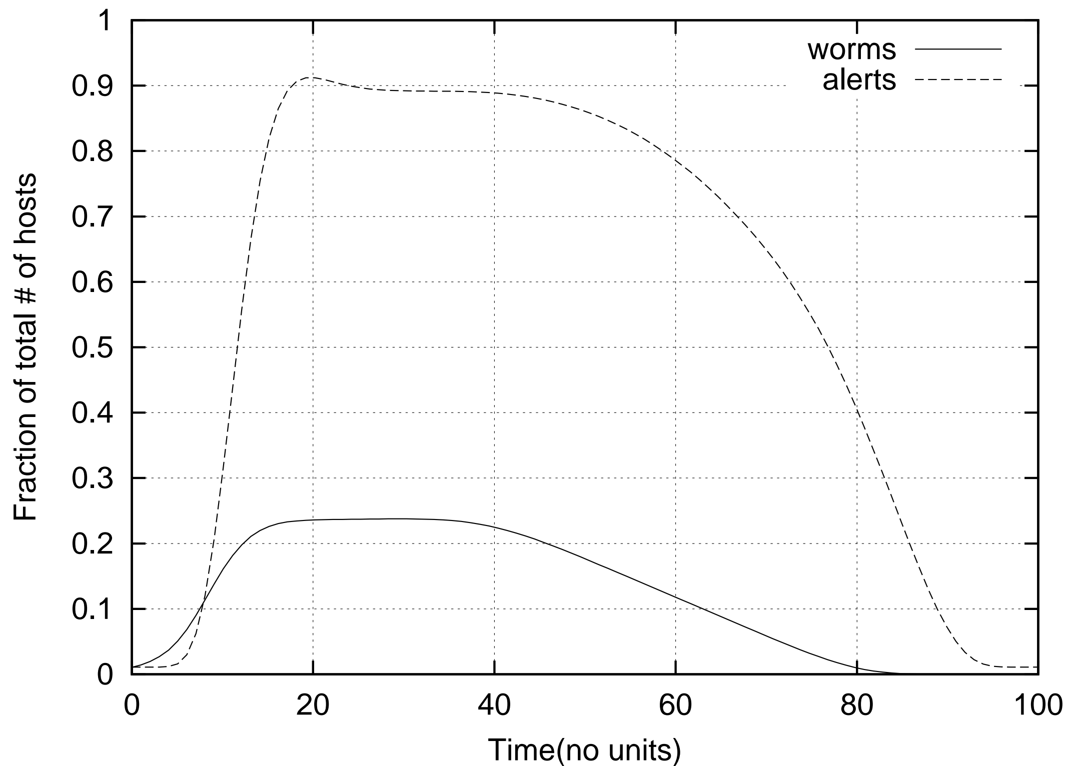

Figure 8.2: The graph shows how the alert messages catch up with the worm.

This work was further extended by the other members of the lab and presented in [14].

8.6 Limitations

This model could fail if the network is already saturated with malicious worm traffic. Reference [7] deals with such network connectivity problems. The techniques described in [7] can be used to analyze the robustness of network architectures against denial-of-service by link and node destruction.

If we treat our *friends' network* as a graph with hosts representing nodes and the links between them as edges, we could answer the question, 'how many links may get saturated before the alert messages can't reach others?'.

N : Total # of participating hosts

F : # of friends for each host

τ : # of independent hosts from which alerts are required before any action can be taken

There are F unsaturated links from each host to its friends when there is no worm. If this host has to take action, it should have at least τ links unsaturated to receive τ alerts. Thus, a maximum of $F - \tau$ links could be saturated. If more than $F - \tau$ nodes are saturated, then this particular node cannot receive the required number of alerts to take any action. Considering the links to be uni-directional, since there are N hosts in total, we could have a maximum of $N.(F - \tau)$ links saturated before the model can fail if failure is defines as any host has more than $F - \tau$ links saturated. Beyond that for each additional link that gets saturated, a maximum of one node gets cut-out from the network. In the worst case, if each additional link beyond $N.(F - \tau)$ that becomes saturated cuts off one node all nodes are isolated when $N.(F - \tau) + N$ links get saturated.

Another limitation of this model is that the signature of the worm is assumed to be available already or almost immediately after the worm is detected. But, it is very difficult to get the signatures of *zero-day* attacks within a very short time. Because, there are several techniques for very high-speed worms, this model could come a cropper against them. Fortunately, however, as discussed in chapter 7 *zero day* attacks are very far and few in between.

8.7 Future Directions

An important aspect, crucial for the success of this model, is the generation of worm signatures. A technique for the automatic generation of worm signatures should be developed. This model of defense should also be validated by experiments on isolated networks. As with other models, this model also requires a richer mathematical model.

8.8 Summary and Conclusion

This model uses a peer-to-peer strategy to control large scale worm attacks on the Internet. It shows that a controlled *white worm* propagating faster than the worm is an effective strategy to minimize the number of worm victims. This chapter provided a mathematical model for the spread of worms in general. It also gave expressions for the rate of spread of the infection as well as the white worm.

Simulations with smaller number of friends for each node suggest that a larger number of friends would suppress worms better. But simulations also showed that the number of alerts exchanged grows exponentially and the network performance degrades. Also, it is not scalable to the real world as one cannot have a large list of trusted friends. A large number of friends tends to give rise to a large number of false alarms. A certain amount of false alarms is inevitable. But the system suffers when faced with a large number of false alarms. However, when we use an optimized friends list this model is able to contain the worm to fewer hosts. In the simulation environment which had 5761 routers in a grid of 81×81 with 8 hosts connected to each router the optimized list turned out to have 8 friends.

Chapter 9

The Hierarchical Worm Defense Model

9.1 The Model

This model assumes a tree structure where the internal nodes of the tree are firewalls and leaves are servers vulnerable to worm attacks. See Figure 9.1. The firewalls are assumed to be immune to infections. It is also assumed that we have sensors at the vulnerable hosts that can detect an infection and report it. All nodes at a particular level of the hierarchy have equal number of children. Each level of the hierarchy has a certain threshold associated with it. Once the number of infection reports amongst a node's(firewall's) children reaches the threshold, the firewall turns on the filter rules protecting all of its children, and alerts its parent that the sub-tree below it is infected but now protected. This escalation of alerts from one level to the next higher level in the hierarchy and protection of sub-trees takes place successively as the threshold for infections is reached at each node.

False alerts are handled by the firewalls by associating a 'Time to Live'(TTL) with the latest alert that they receive. If the threshold is reached before the TTL expires the above actions are taken. If the TTL expires before the threshold is hit, all the alerts/infection reports are discarded and the firewalls 'back-off' from protecting its chil-

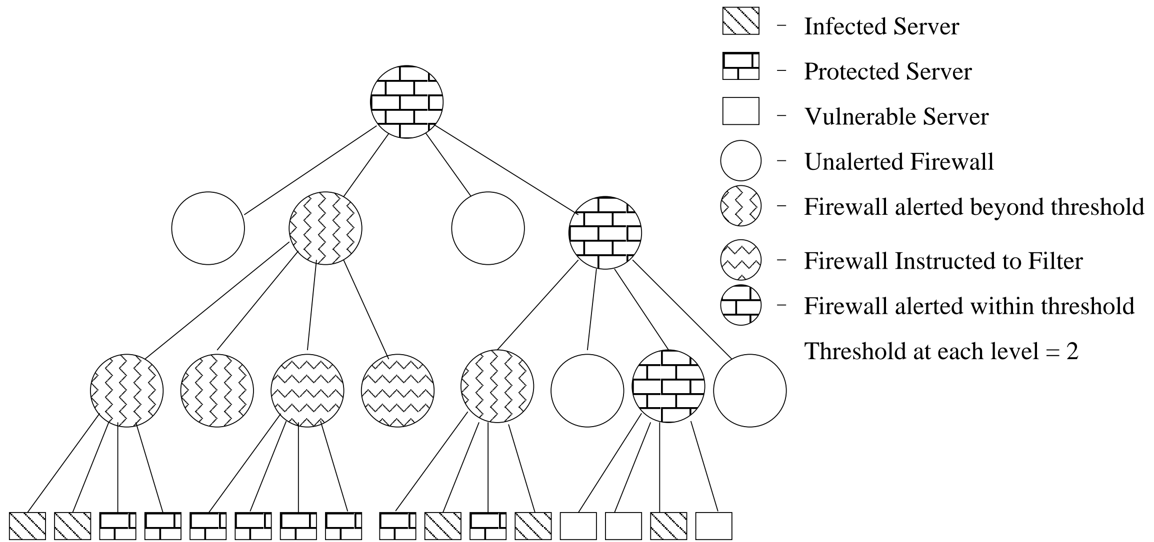

Figure 9.1: Hierarchical relationship among firewalls and vulnerable hosts in Hierarchical Model.

dren. This is done because any action taken to contain a worm involves a cost, and actions undertaken when there is no worm are of no use.

9.2 Mathematical Models

Simple mathematical models of worm spread were already described in section 8.3. In this section we will derive some of the interesting quantities of the hierarchical model of mitigation mathematically. As always, the goal of this whole project is to minimize the number of infections amongst our participating hosts.

The first quantity of interest is the probability with which infections are minimized. For this to happen, the root node should trigger the protection of its children with the minimum number of hosts being infected.

Probability to minimize infection For the sake of this derivation let us consider that the leaves are at level 0 of the tree, the parents of the leaves are at level 1 and so on.

$$
\begin{aligned}
\text{\# of children per level} &= \lambda \\
\text{\# of levels} &= n \\
\text{\# of leaves} &= \lambda^{n-1} = m_0 \\
\text{\# of nodes one level above leaves} &= \lambda^{n-2} = m_1 \\
\text{P(an infection choosing one particular domain)} &= 1/\lambda^{n-2} = 1/m_1 \\
\text{P(of } i \text{ infections choosing the same particular domain)} &= 1/\lambda^{n-i} = 1/m_1^i \\
\text{\# of ways this one domain can be chosen} &= C_1^{m_1} = m_1 \\
\text{Threshold} &= \tau - 1 \\
\text{Minimum \# of infections needed at the leaves level to alert the root} &= \tau^{n-1}
\end{aligned}
$$

For a node to protect its children, it should have received τ alerts from its children. That is the number of infections $i = \tau$.

$$
\begin{aligned}
\text{P(some non-leaf node at level 1 to protect to its children)} &= C_1^{m_1}.1/m_1^\tau \\
&= m_1/m_1^\tau \\
&= 1/m_1^{\tau-1}
\end{aligned}
$$

Similarly,

P(some non-leaf node at level 2 to protect to its children)

$$
\begin{aligned}
&= \left(\frac{1}{m_2^{\tau-1}}\right).\text{P}(\tau \text{ nodes being alerted at level 1}) \\
&= \left(\frac{1}{m_2^{\tau-1}}\right) \cdot \left(\frac{1}{m_1^{\tau-1}}\right)^\tau
\end{aligned}
$$

P(some non-leaf node at level 3 to protect to its children)

$$= \left(\frac{1}{m_3^{\tau-1}}\right).\text{P}(\tau \text{ nodes being alerted at level 2})$$

$$= \left(\frac{1}{m_3^{\tau-1}}\right) \cdot \left[\left(\frac{1}{m_2^{\tau-1}}\right) \cdot \left(\frac{1}{m_1^{\tau-1}}\right)^{\tau}\right]^{\tau}$$

$$= \left(\frac{1}{m_3^{\tau-1}}\right)\left(\frac{1}{m_2^{\tau-1}}\right)^{\tau}\left(\frac{1}{m_1^{\tau-1}}\right)^{\tau^2}$$

$$\vdots$$

P(some non-leaf node at level $(n-1)$ to protect to its children)

$$= \left(\frac{1}{m_{n-1}^{\tau-1}}\right)^{\tau^0}\left(\frac{1}{m_{n-2}^{\tau-1}}\right)^{\tau^1}\left(\frac{1}{m_{n-3}^{\tau-1}}\right)^{\tau^2}\cdots\left(\frac{1}{m_1^{\tau-1}}\right)^{\tau^{n-2}}$$

$$= \prod_{i=1}^{n-1}\left(\frac{1}{m_{n-i}^{\tau-1}}\right)^{\tau^{i-1}}$$

But the only node at level $n-1$ is the root node. So, the above expression gives us the probability that the root will be alerted with the minimum number of infections at the leaves.

Time to alert root The next interesting mathematical result is the time it takes to achieve complete protection. That is the time it takes to alert the root so that it can trigger protection of its children.

For the sake of this derivation we will consider a small sub-tree with just 2 levels. The top level contains the root node and the leaves all form a group. Each infected leaf node tries to infect all non-infected leaf nodes in its group. For each infected/non-infected leaf node pair, the time until infection of the non-infected node by the infected one is exponentially distributed. By scaling of time, we can take the mean of this distribution to be 1.0. It is assumed that all infecting processes operate stochastically independently.

i : # of infected nodes in a group

h : the threshold for alerts

g : the group size

Then in state i, there are $i(g-i)$ exponential random variables in progress at once, since each of the i infected nodes is trying to infect each of the $g-i$ uninfected nodes. Then the time to go from state i to $i+1$ will be the minimum of $i(g-i)$ exponentially distributed random variables, and thus will itself be exponentially distributed[1], with mean $1.0/[i(g-i)]$.

For simplicity, we will consider the case $h = g-1$; the more general case is handled similarly. The total expected time to an alert, starting at the time the first member of the group becomes infected, is

$$\sum_{i=1}^{g-1} \frac{1}{i(g-i)} \tag{9.1}$$

Using a standard approximation, (9.1) is approximately equal to

$$\int_1^{g-1} \frac{1}{x(g-x)}\, dx = \frac{1}{g} \int_1^{g-1} (\frac{1}{x} + \frac{1}{g-x})\, dx = \frac{2}{g} ln(g-1) + C \tag{9.2}$$

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

In other words, (9.1) 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.

9.3 Description of the Simulation

The Hierarchical Model simulation was implemented in Perl. 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. 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. The worm scenarios were simulated with a large TTL value, 1000.

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

[1] Note that the Markov property is being used here implicitly.

occurs. And each alert is propagated as high as possible in the hierarchy in the same time slice.

In the case of false alarms, the rate of false alarms is fixed at the beginning of the simulation. Unlike worms where only infected machines can infect others, there is no relationship between one false alarm and the next. The machine that raises a false alarm is determined randomly. All false alarm scenarios were simulated with TTL window sizes varying from 10 to 400 time units.

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.

9.4 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 Fig.9.2 and Fig.9.3. These two figures show that the number of infections before complete immunization could take place is almost the same for both the cases.

The simulation was done for different rates of infection with 2 different levels of thresholds and the results are shown in Fig.9.4. As we can see in Fig.9.4, the number of infections before complete immunization takes place is almost the same for varied rates of infection. But the time it takes is different and favorable too as seen in Fig.9.5. For high rates of infection, maximum possible protection is achieved quicker than for slower rates of spread. This is a direct result of the dependence of the alert propagation on the infection rate. And we also see that the thresholds don't determine the time taken as it takes almost the same time to achieve complete immunization for 2 different threshold values. The only thing that varies with threshold is the number of infected machines.

Thus the lessons learnt are :

- For any rate of infection, the number of victims is the same.

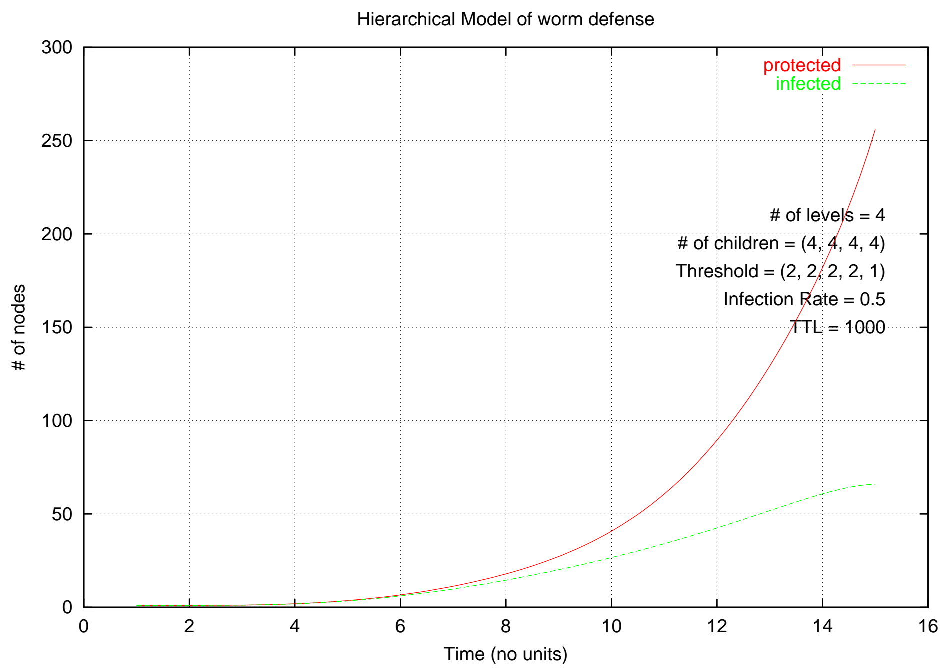

Figure 9.2: Response for a low rate of infection.

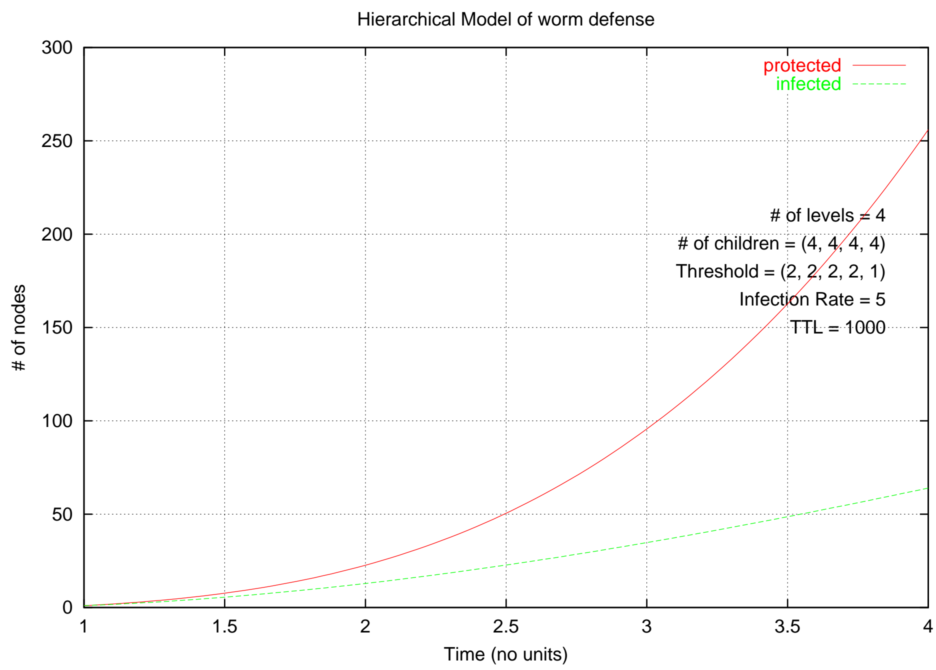

Figure 9.3: Response for a high rate of infection.

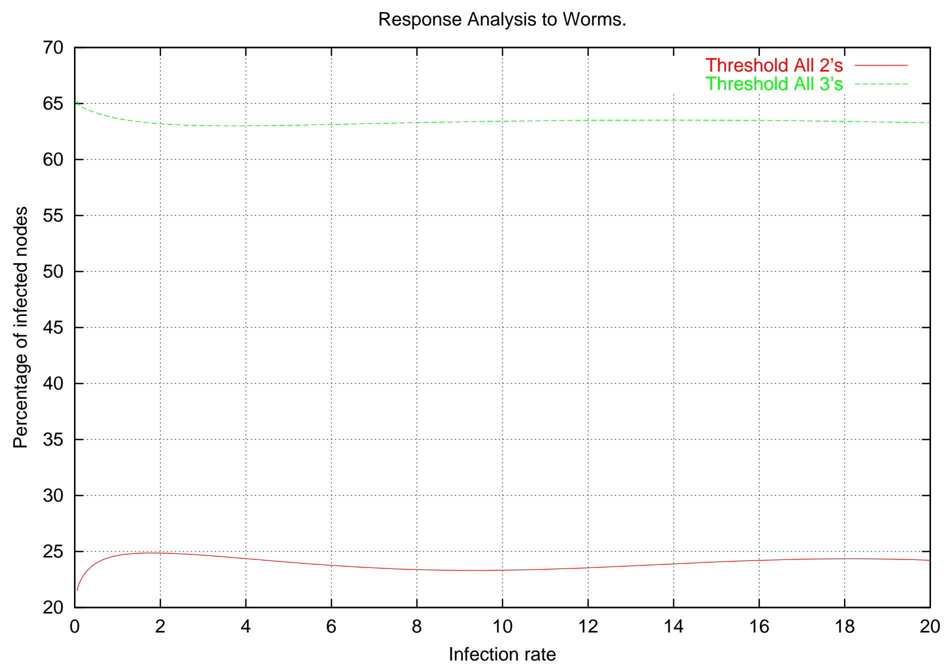

Figure 9.4: Percentage of machines infected for different rates of infection.

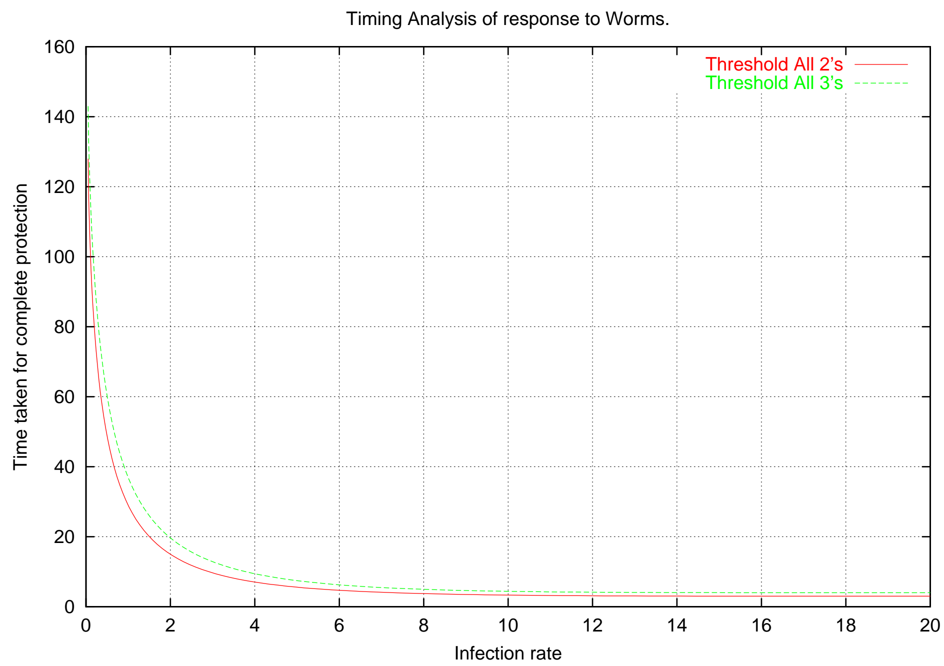

Figure 9.5: The time taken for complete protection is inversely proportional to the infection rate.

- The time taken for complete protection is inversely proportional to the infection rate.
- It is only the threshold levels that makes or breaks the network. A low threshold helps to save a lot of machines.

It is obvious that we can't forecast an unknown worm's infection rate. But we can define our tolerance. So, we now have a model which can tell us what should be the threshold, for a given tolerance.

For example, if we want to save 90% of the Internet in the event of a worm attack, we need to find out from simulations what should be the thresholds at various levels of the network hierarchy and set the firewall rules accordingly. Setting the right thresholds at various levels of the hierarchy would achieve our goal of saving 90% of the Internet for us slowly or quickly, depending on the infection rate of the worm.

9.4.1 False Alarms

During false alarms the number of machines given protection does not rise steadily as in case of real worms. Rather, the number of machines protected keeps oscillating as the systems backs off if there are no alerts within the TTL, as seen in Fig.9.6. This oscillation is an indication of a series of false alarms. It can also be considered as an indication of a Stealth Worm spreading very slowly. This is discussed in the next sub-section.

Fig.9.7 shows the average number of machines that are given protection in response to false alarms for various TTLs and various false alarm rates. This protection involves a price and can be considered as a self inflicted DoS attack. As we can see, if the TTLs are low enough, we pay a much lower price. But we would not be able to capture Stealth worms as is seen in figure 9.9. At the same time, a high TTL would raise false positives even for low rates of false alarms and raise costs unnecessarily.

9.4.2 Stealth Worms

We can see the same oscillating pattern in figure 9.8 and figure 9.9 which were recorded for a Stealth worm simulation with a TTL of 60. Figure 9.8 shows a case where the stealth worm is suppressed because of a low threshold of the defense system.

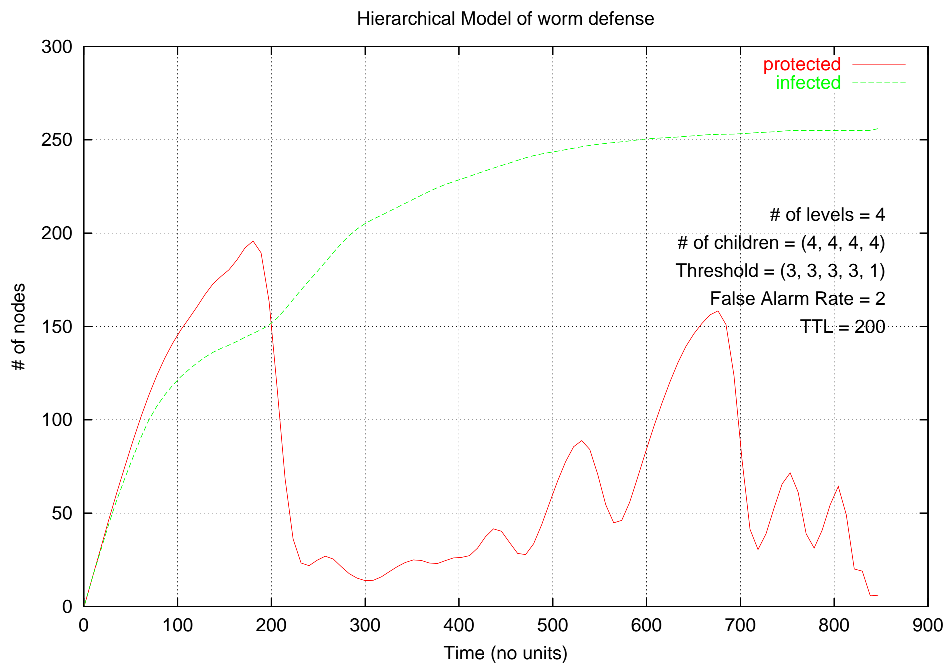

Figure 9.6: The number of machines that are protected keeps oscillating in case of false alarms.

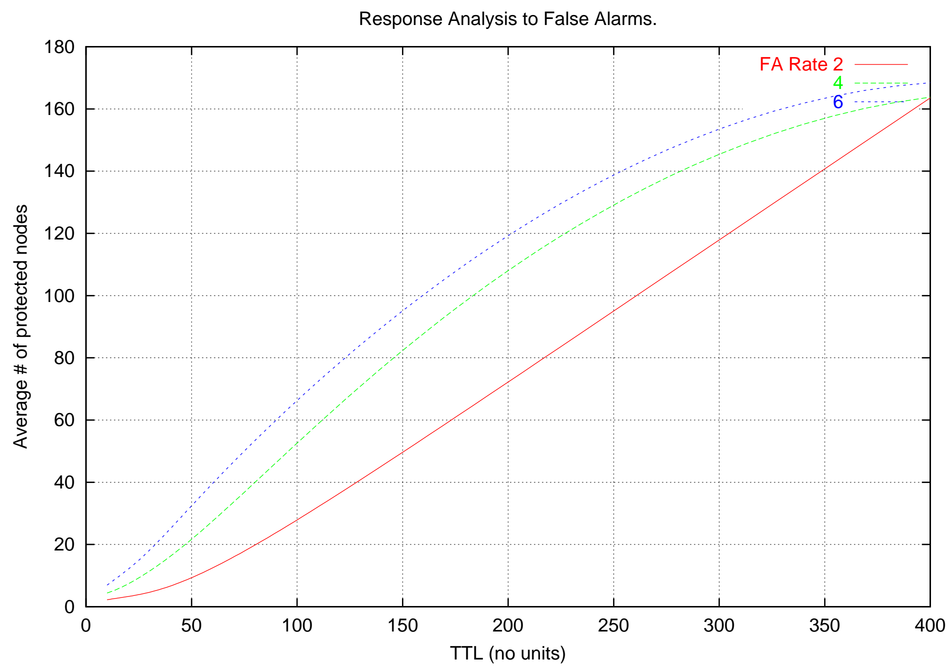

Figure 9.7: Average number of machine protected for different False Alarm rates for varying TTLs.

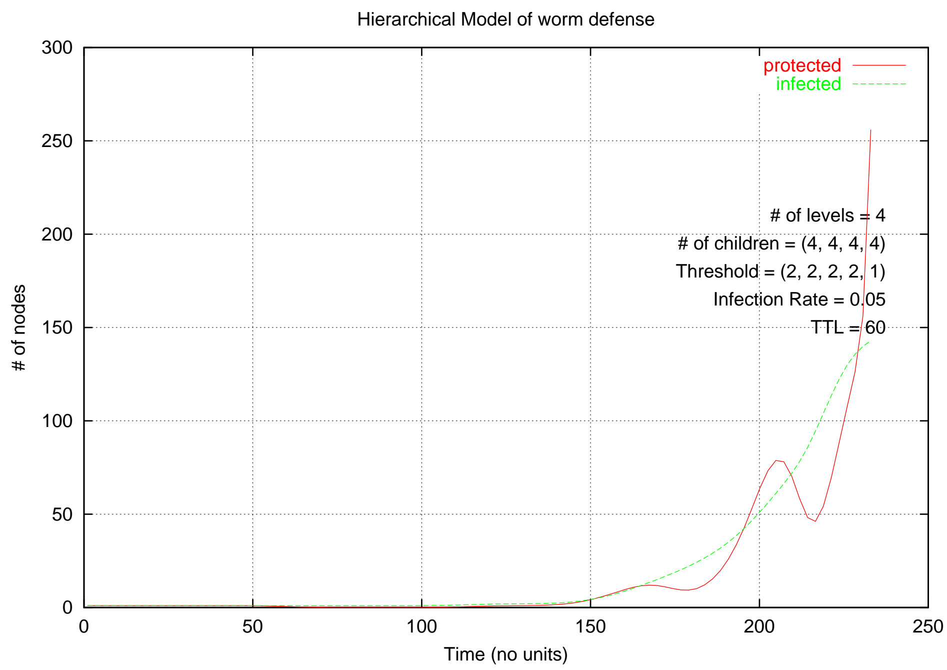

Figure 9.8: A stealth worm overpowered with a low threshold.

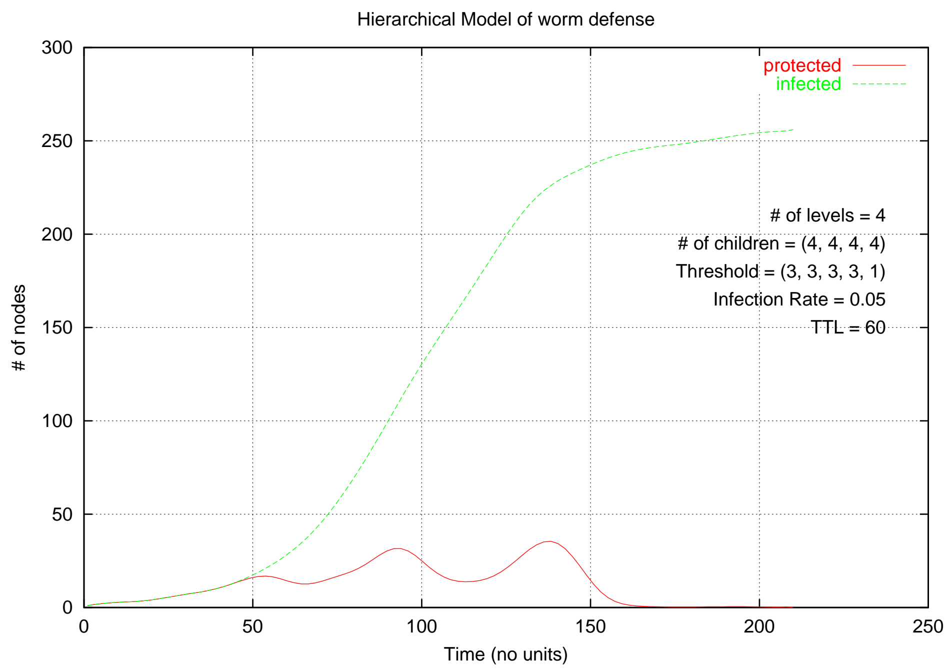

Figure 9.9: A Stealth worm sneaking in due to a high threshold.

Fig.9.9 shows a scenario where the protection mechanism is able to sense that something is wrong. But due to the high threshold, the rate of alerts received is just low enough that the TTL expires frequently and our system backs-off reasoning it as a false alarm. This suggests that we need a higher TTL or a lower threshold, which we addressed above. But a high TTL means a high cost due to false alarms which may be unacceptable.

Since an oscillating curve means a series of false alarms or a stealth worm which is slow spreading, we can afford the luxury of human intervention.

So, we need to arrive at a compromise saying that we will look out for Stealth worms which only spread above a certain speed. In any case, slower worms will get exposed as more and more machines are infected, as this will increase the overall rate of infection. So, the TTL also dictates how many machines we will have to sacrifice before our defense mechanism takes over. We may even lose all machines before we respond if we choose a very small TTL.

9.5 Future work

An extension of this model to handle false-alarms would be the following. The TTL is increased for the next alert for it could be a 'Stealth worm' that is spreading. As we know, Stealth worms spread very slowly. The quantum of increase should neither be too large that we fall prey to false alarms nor should it be too small that 'Stealth worms' propagate. However, there is no definite algorithm to determine this quantum.

This TTL could be reset to a desired realistic value once it reaches a maximum limit or after the alerts have been thoroughly examined manually by experts and have been declared to be false alarms.

A richer mathematical model needs to be developed to verify the simulations. Developing a prototype to verify these models on isolated networks with real worms will be the next step in implementing this model. The load on the network due to the alert messages is another facet that needs to be studied.

Possible related work includes a study of correlation of data from various sites and development of a model to identify if a malicious activity is caused by a worm. The required data could be collected from various intrusion detection systems deployed at different trusted

locations. GrIDS[26] is a good place to start.

Theoretical analysis on the effect of the propagating worm taking up bandwidth to the point where the alert messages cannot get through is in order. This could be done by treating the network as a graph and answering the question, "How many links may get saturated before the alerts cannot reach others?" Reference[7] would form a good starting point for this research.

There are a myriad of ways to make this model richer. Possibilities include automatic detection of worms with a minimum of uncertainty as possible, automatic generation of worm signatures, and so on. A hybrid model could be built by merging both the hierarchical model and friends model. Techniques for co-operative automatic recovery of compromised hosts could be developed.

9.6 Summary and Conclusions

This model uses a hierarchical relationship between co-operating hosts to mitigate the spread of a worm. Each leaf node is a vulnerable host and each non-leaf node is an invulnerable control structure which can process alerts sent by sensors at leaves and each non-leaf can issue appropriate instructions to its children.

This chapter also provided a mathematical analyses of the hierarchical model of worm defense and showed that the time to alert the root of the hierarchy is bounded.

From this model, and the simulations, we can determine the thresholds required at various levels of the hierarchy for a given tolerance of lost machines. We can also determine the TTL to stop a Stealth worm of a given speed by looking at the usual rate of false alarms in an environment. With the threshold levels and TTLs thus determined, we can effectively inhibit the spread of worms without losing much due to false alarms.

Chapter 10

Summary and Conclusions

This thesis gave a step-wise 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 some of the popular worms were analysed in detail with respect to the techniques they used to select their victims and how they spread. An overview of both good and malicious uses of worms was provided. It developed a comprehensive model of a simple worm and described the components that make up a worm. Simple calculations modeled the scan rate of worms.

This thesis then developed a life cycle model of defense against computer worms. It showed that the fight against malicious worms is a cyclic process. It discussed various aspects of detection of worms and emphasized the need to keep systems patched and up-to-date. It advocated that preventing a worm incursion is better than trying to remidiate one.

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 in the context of worms of the past and of the future and examples were provided. It observed that worms like the Flash worms, Warhol worms and the Slammer worms could spread throughout the entire Internet in a matter of few minutes. It noted the subtle dangers of Stealth worms. It briefly introduced e-mail viruses, stubborn, wireless and coffee shop viruses.

We listed down the various aspects of automatic and manual worm mitigation

techniques and it tabulated the applicability of each of these two techniques against worms of different speeds. A few means of implementing automatic means of mitigation and differentiated detection and declaration of a worm outbreak were discussed. It also described in length the steps involved in prevention of a worm outbreak. A number of tools that help identifying configuration vulnerabilities were introduced and discussed.

It developed an innovative model of prediction of worm outbreaks and the vulnerability that a next worm in the future would most probably exploit called TrendCenter. This model aims to measure the threat environment, detect emerging threats and disseminate the knowledge by analyzing alert logs collected from various IDSs. This model optimizes the system administrator's time spent for securing one's system by providing a priority list of patches to be applied to hosts in one's environment. And it helps minimize cost of system maintenance in terms of man power as well as computing power. TrendCenter's governing philosophy is that standalone IDSes are no longer useful by themselves but a collective summary of the alerts from many different IDSs may be useful in preparing for a future attack. The *'Detect and Respond'* is dead and *'Predict and Prepare'* strategy is the way of the day.

The thesis went on to describe two models of automatic worm mitigation. The first one to be dealt was the friends model. This model uses a peer-to-peer strategy to control large scale worm attacks on the Internet. It shows that a controlled *white worm* propagating faster than the worm is an effective strategy to minimize the number of worm victims. It showed that with 8 friends for each node the system controls the worm best for the simulation scenario of 5761 routers with 8 hosts connected to each router. It argued that a larger number of friends is not practical. It also expected that the inevitable false alarms raised by a large number of friends would become too big for the system to perform with any reasonable accuracy. Also the number of alerts exchanged amongst friends becomes too huge to the point of choking the network and thereby degrading the performance.

The next mitigating model discussed was the hierarchical model of worm defense. This model uses a hierarchical relationship between co-operating hosts to mitigate a worm spread. Each leaf node is a vulnerable host and each non-leaf node is an invulnerable control structure which can process alerts sent up by sensors at leaf level and can issue appropriate instructions to its children.

Using this model and the simulations we can determine the thresholds required at various levels of the hierarchy for a given tolerance of lost machines to handle worms. We can also determine the TTL to stop a Stealth worm. Thus, with the threshold levels and TTLs thus determined, we can effectively stop worms without losing much due to false alarms.

This thesis concludes that worms could be very dangerous to the Internet and could have extremely adverse ramifications to the point of threatening lives in a large scale. But there is hope; there are several techniques to mitigate the ill-effects of worms as illustrated in this thesis.

Chapter 11

Future Work

Since this is a fairly recent topic of research, there is a wide range of projects that can be done in the future extending the models developed in this research and presented in this thesis. The future directions for each of the models developed in this thesis were presented at the end of each chapter. They will be summarized here for completeness in addition to several projects that can be done on this subject as a whole integrating several aspects discussed in this thesis.

One such project is to refine the model of a worm developed in this thesis to a more accurate model which can be used in model based systems for worm detection. A ready reckoner of various worms, real and hypothetical, with their signatures, spread strategy and other unique characteristics of each worm would be a very handy tool in the hands of computer forensic experts and researchers.

Honeypot techniques can be extended to develop worm prediction models as future project. Automatic signature generation of worms is a very crucial step in the success of automatic worm mitigation techniques. But we don't have such a capability. So, this would a very important future project to undertake.

A testbed that can be used to test such models of defense against worms would be a great help to research on worms. And so would be a standard template to generate worms at will for this experimental testbed. This might be used for malicious purposes if it gets into wrong hands. Hence, this template should generate worms that exploits vulnerabilities

that were known to exist long time ago and known to be fixed now. Another way out would be to artificially incorporate vulnerabilities into the testbed and generate worms that exploit this vulnerability. Either way, it should be easy to specify a vulnerability, incorporate that into the testbed and generate a worm that exploits the vulnerability.

Developing techniques that can declare a worm with the least uncertainty is an important future project. A survey of all the tools available for finding vulnerabilities with their strengths and weakness would be a very useful reference guide for system administrators to easily choose the tool most suited for their environment. This would help them to strengthen their systems and keep them patch-free with minimum effort.

Addressing the limitations of each of the models mentioned in their respective chapters is another issue that merits further exploration.

As for the TrendCenter effort, its sensor foundation can be improved. Newer and more signatures can be added to the rules database. Sanitizers for logs generated by IDSs other than SNORT could be developed. The system could be improved to detect new attacks like systematic anomalies. Dynamic query systems can be added to the model for flexible queries of incidents. A better representation of the statistics would be very helpful. Some of the ideas are to provide cluster of IP addresses; this would reduce the number of rules in the firewalls, and ranking offenders based on a recidivism index. Providing the most relevant CERT contacts, cyber laws and cyber law enforcement agencies would add value to the service provided. Expanding the contributors database is an important aspect to avoid skewing of analysis due to difference in the environment of the data contributors.

Both friends model and the hierarchical model need richer mathematical models and experimental validation of the simulations. These could be tested on isolated networks testbed. Both these models need studies of the effect of network loads on these models in the face of a worm outbreak. "How effectively would these models be able to exchange messages in face of a very fast worm like Slammer?" is a question that remains to be addressed. Both the friends and the hierarchical models could be combined to develop a hybrid model which might be more effective.

A fitting tribute to these models would be to integrate TrendCenter and one of the mitigating models and make them robust enough to face the real Internet scenarios.

Appendix A

The AttackTrends Sanitizer

NAME

`ats` - Attack Trends Sanitizer Perl script for Snort fast format alert files

SYNOPSIS

```
ats [-debug] [-sd MMDD] [-ed MMDD] [-s <ip or part of ip>] <alertfile>
```

DESCRIPTION

This script parses fast format alert file generated by snort. Takes in an `<alertfile>` as the argument and returns the parsed output in `<alertfile>.parsed`. Each record in The output file contains the following data:

```
<# of unique alerts> <month/day> <src-ip> <sid> <dst-port>
```

The last line of the file contains:

```
# of uniq targets = <# of uniq targets>
```

`<month/day>` indicates the month and day this attack was reported.

`<src-ip>` indicates the IP address which launched this attack.

<`sid`> indicates the Snort ID this attack has been assigned.

<`dst-port`> indicates the port number this attack targeted.

NOTE

If there are 'n' consecutive identical alerts, they are counted as 1. For example, if one attacker attacks one target with the same attack consecutively 500 times, it is counted as only one attack. On the other hand, if the attacker attacks 2 targets alternatively for 500 times each we would count them as 2 different attacks each launched 500 times.

OPTIONS

`-debug` When included prints out some helpful debug info.

`-s` The alerts that have their source IP address belonging to the given IP class are suppressed and are not processed beyond the first parse. This option can be repeated to suppress multiple IP addresses.

`-sd` Only those alerts generated on or after this date are processed beyond the first parse.

`-ed` Only those alerts generated on or before this date are processed beyond the first parse.

AUTHOR

Senthilkumar G. Cheetancheri cheetanc at cs dot ucdavis dot edu

Bibliography

[1] “http://www.kryptocrew.de/snakebyte/e/StoppingWorms.txt”. Internet.

[2] “http://www.swarm.org”. Internet.

[3] “Nessus Vulnerability Scanner”. Internet. http://www.nessus.org.

[4] “SNORT IDS”. Internet. http://www.snort.org.

[5] “http://www.hackbusters.net/LaBrea/”. Internet, 2001.

[6] “TrendCenter”. Internet, July 2003. http://AttackTrends.com.

[7] Tuomas Aura, Matt Bishop, and Dean Sniegowski. “Analyzing Single-Server Network Inhibition”. In *Proceedings of the 13th Computer Security Foundations Workshop*, pages 108–117, July 2000.

[8] Robert W. Baldwin. “Kuang: Rule based security checking.”. Documentation in ftp://ftp.cert.org/pub/tools/cops/1.04/cops.104.tar. MIT, Lab for Computer Science Programming Systems Research Group.

[9] B.G.Barnett. “NOOSE - Networked Object-Oriented Security Examiner”. In *Proceedings of the 14th Systems Administration Conference, USENIX Association*, Nov 2000.

[10] Cert. “Cert Advisory CA-2001-19 ”Code Red” Worm Exploiting Buffer Overflow In Iis Indexing Service DLL”. Internet, January 2002. http://www.cert.org/advisories/CA-2001-19.html.

[11] C.G.Senthilkumar. “Worms: How to stop them? - A proposal for Master’s thesis.”. University of California, Davis. http://wwwcsif.cs.ucdavis.edu/c̃heetanc, July 2002.

[12] C.G.Senthilkumar and Karl Levitt. “Hierarchically Controlled Co-operative Response Strategies for Internet Scale Attacks”. University of California, Davis. http://wwwcsif.cs.ucdavis.edu/c̃heetanc., January 2003.

[13] D.Farmer and W.Venema. “Security Administrator’s tool for analyzing networks”. http://www.fish.com//zen/satan/satan.html.

[14] D.Noijiri, J.Rowe, and K.Levitt. “Cooperative Response Strategies for Large Sacle Attack Mitigation”. DISCEX, 2002.

[15] Mark W. Eichin and Jon A. Rochlis. “With Microscope and Tweezers: An analysis of the Internet Virus of November 1988”. In *Proceedings of the symposium on Research in Security and Privacy*, May 1988. Oakland, CA.

[16] Dan Farmer and Eugene H. Spafford. “The cops Security Checker System”. USENIX, 1990. Summer.

[17] Gene Kim and Eugene H. Spafford. “The design of a system integrity monitor:Tripwire”. Technical Report CSD-TR-93-071, Purdue University, West Lafayette, IN, USA, November 1993.

[18] David Moore et al. “Inside the Slammer Worm”. In *IEEE Security and Privacy*, August 2003.

[19] Carey Nachenberg. “Computer Parasitology”. Symantec AntiVirus Research Center.

[20] Carey Nachenberg. “Understanding and Managing Polymorphic Viruses”. Symantec AntiVirus Research Center.

[21] Don Seeley. “A Tour of the Worm”. In *Proceedings of 1989 Winter USENIX Conference*, pages 287–304, Berkeley, CA, February 1989. Usenix Association, USENIX.

[22] John F. Shoch and Jon A. Hupp. “The “Worm” Programs - Early Experience with a Distributed Computation”. *Communications of the ACM*, 25(3):172–180, March 1982.

[23] Eugene H. Spafford. "The Internet Worm Program: An Analysis". Technical Report CSD-TR-823, Purdue University, West Lafayette, IN, USA, December 1988.

[24] Eugene H. Spafford. "The Internet Worm: Crisis and aftermath". *Communications of the ACM*, 32(6):678–687, June 1989.

[25] Eugene H. Spafford. "The Internet Worm Incident". Technical Report CSD-TR-933, Purdue University, West Lafayette, IN, USA, September 1991.

[26] S.Staniford-Chen et al. "GrIDS – A Graph-Based Intrusion Detection System for Large Networks". In *The 19th National Information Systems Security Conference*, volume 2, pages 361–370, October 1996.

[27] Stuart Staniford, Gary Grim, and Roelof Jonkman. "Flash Worms: Thirty Seconds to Infect the Internet". Silion Defense - Security Information, August 2001.

[28] Stuart Staniford, Vern Paxson, and Nicholas Weaver. "How to Own the Internet in Your Spare Time". In *Proceedings of 2002 Summer USENIX Conference*, Berkeley, CA, August 2002. Usenix Association, USENIX.

[29] Nicholas Weaver. "Future Defenses: Technologies to Stop the Unknown Attack". Internet, February 2002. http://online.securityfocus.com/infocus/1547.

[30] Nicholas Weaver. "Potential Strategies for High Speed Active Worms: A Worst Case Analysis". UC Berkeley, March 2002.

[31] Nicholas Weaver. "Warhol Worms: The Potential for Very Fast Internet Plagues". UC Berkeley, February 2002.

[32] Tarkan Yetiser. "Polymorphic Viruses. Implementation, Detection and Protection.". VDS Advanced Research Group, January 1993.

[33] Dan Zerkle and Karl Levitt. Netkuang - a multi-host configuration vulnerability checker. USENIX, 1996.