

2006

Defending flash worms : contemporary detection schemes and a hierarchical model

Pele Y. Li
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Li, Pele Y., "Defending flash worms : contemporary detection schemes and a hierarchical model" (2006). *Master's Theses*. 2903.
DOI: <https://doi.org/10.31979/etd.29pn-zjj3>
https://scholarworks.sjsu.edu/etd_theses/2903

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

DEFENDING FLASH WORMS -
CONTEMPORARY DETECTION SCHEMES
AND A HIERARCHICAL MODEL

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Pele Y. Li

May 2006

UMI Number: 1436929

Copyright 2006 by
Li, Pele Y.

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1436929

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2006

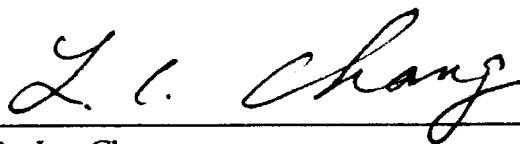
Pele Y. Li

ALL RIGHTS RESERVED

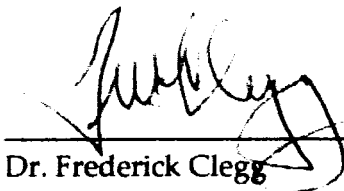
APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING



Dr. Xiao Su



Dr. Lee Chang



Dr. Frederick Clegg

APPROVED FOR THE UNIVERSITY



Abstract

DEFENDING FLASH WORMS CONTEMPORARY DETECTION SCHEMES AND A HIERARCHICAL MODEL

By Pele Li

Internet worms are one of the most harmful kinds of Internet attacks. The self-duplicating, self-propagating malicious code requires no human interaction and thus can spread quickly over a wide area.

The body of this thesis consists of four parts. First, this paper categorizes the characteristics of Internet worms. Second, it provides a comprehensive analysis of current worm detection and containment schemes. The detection schemes are then compared against worm characteristics. Third, this paper explores the design challenges of flash worms, a conceptual worm that has the potential to be the fastest worm, and possible solutions to these challenges. Lastly, a hybrid hierarchical model is proposed to catch both current worms and future flash worms.

Table of Contents

1. Introduction	1
1.1 Overview	2
1.2 Terminology	3
2. Internet Worms	4
2.1 Worm Target Finding Schemes	5
2.2 Worm Propagation Schemes	7
2.3 Worm Transmission Schemes	8
2.4 Worm Payload Formats	9
2.5 Existing Internet Worms	10
2.5.1 Code Red I & Code Red II	10
2.5.2 Slammer/Sapphire	11
2.5.3 Sasser	12
2.5.4 Witty Worm	13
2.6 Multiplatform and Multiexploit Worms	14
3. Defending Against Internet Worms	16
4. Worm Detection	17
4.1 Signature-Based	17
4.2 Anomaly-Based	20
4.2.1 Traffic Rate/Connection Count – TCP SYN	22
4.2.2 Failure Connection Rate – TCP RST & ICMP	23
4.2.3 Ratio of Success and Failure Connections	25
4.2.4 Destination-Source Correlation (DSC)	27
4.2.5 DarkNet/Unused Address Space	29

4.2.6 Honeypots	30
4.3 Combination Usage of Detection Schemes	33
4.4 Summary on Worm Detection	35
5. Containment	38
5.1 Slowing Down Infection	38
5.2 Blocking	39
5.2.1 Address Blocking	40
5.2.2 Content Blocking	41
5.3 Honeypot to Decoy	42
6. Fighting Worms in Different Scopes	44
6.1 Location of Defense	44
6.2 Scope of Defense	47
7. Flash Worms Detection and Simulation	51
7.1 Flash Worms	51
7.1.1 Challenges of Flash Worm Implementation	52
7.2 Flash Worm Detection and Challenges	54
7.3 Simulation	55
7.3.1 HoneyD	55
7.3.2 Simulation of Test Environment	56
7.3.3 Simulation of Flash Worms	57
7.4 Challenges of the Simulation	60
7.5 Hierarchical Detection	62
8. Conclusion	69
References	71

List of Tables

Table 1 Existing Internet Worm Implementation 14

Table 2 A Comparison of Worm Characteristics with Anomaly Detection
Methods..... 36

Table 3 A Comparison of Worm Characteristics with the Hybrid Hierarchical
Detection Model..... 67

List of Figures

Figure 1 Categorization of Worm Characteristics	5
Figure 2 Categorization of Internet Worm Defense	16
Figure 3 Connection Attempts	23
Figure 4 Illustration of Destination-Source Correlation Scheme	28
Figure 5 Honeyd Honeypots Used in Worm Detection and Containment	31
Figure 6 Different Scopes of Detection and Containment	47
Figure 7 Design of Flash Worm Infection Tree	58
Figure 8 Illustration of Cross Infection	59
Figure 9 Honeyd Simulated Network Environment	61
Figure 10 Hierarchical Detection Model	62
Figure 11 Hierarchical Detection Flow	63

1. Introduction

An Internet worm is a piece of malicious code that duplicates and propagates by itself, normally requires no human interaction, and, thus, can spread widely and quickly, often across countries. It is one of the most powerful Internet attacks. But being fully automated, a worm's behavior is usually repetitious and predictable, making it possible to be detected.

A worm's life consists of the following phases: target finding, transferring, activation, and infection. It has network activities during the first two phases and so network-based detection is able to catch it. This research categorizes worm characteristics for these two phases.

Many algorithms have been proposed in the past years to try to catch and stop the spread of Internet worms. Most research papers discuss efforts that are related to their proposed work, but none of these papers give a comprehensive classification of detection and containment systems. This thesis contains a survey and analysis of Internet Worm detection and containment systems. The research categorizes these systems based on the parameters used in each scheme. These categories are compared against worm characteristics, and the insufficiency of current systems is pointed out. Different locations and scopes of the systems are also explored in detail.

This research discusses the design of the fastest possible worm – the flash worm. The flash worm is not yet seen in the wild, but already has caught lots of attention. The reason is because this type of worm is very hard to detect, and the damage it may cause is severe. This paper presents the design issues of and solutions to the flash worm.

The HoneyD Honeypot system is proposed for simulating flash worms to test their performance. A network environment with background traffic is also simulated in the same system for the worm to run on. Various challenges of the simulation are also discussed.

This thesis proposes a hybrid hierarchical detection model. The model is designed to catch both flash worms and random scanning worms.

1.1 Overview

After the introduction, a terminology section is presented. Worm characteristics during target finding and worm transferring phases are identified in Section 2. The classification of detection algorithms is presented in Section 4 and the containment systems are in Section 5. The detection categories are compared against worm characteristics. This is summarized in Section 4.4. Different locations of implementation for detection and containment systems give different views and thus have different scopes. This is discussed at each

level in Section 6. Section 7 contains the introduction to and simulation of flash worms, as well as the simulated network environment for a flash worm to run on. The proposed detection algorithm is also introduced in Section 7.5. Section 8 concludes this thesis.

1.2 Terminology

False Alarm: Incorrect alert generated by the detection system.

False Positive: False positive is a false alarm that generates an alert when there is no actual attack or danger.

False Negative: False Negative means the detection system has missed an attack. It is a false alarm that does not generate an alert when under attack.

Threshold: The pre-defined condition that, if met, indicates the existence of suspicious traffic or a worm attack.

Viruses and Worms: Worms and Viruses are both malicious, but they are fundamentally different. A worm is a fully automated program that duplicates, propagates, and activates by itself. It normally transfers via network connections. A virus is a piece of code that attaches itself to other programs to propagate. It cannot propagate by itself and normally depends on certain host behavior to activate it.

2. Internet Worms

Since the Morris worm in 1988, Internet worms have caused the most extensive and widespread damage of all kinds of computer attacks. An Internet worm is defined as a piece of code that duplicates and propagates by itself. Usually, it does not require any human interaction and spreads via network connections.

The life of a worm after being released may include the following phases: target finding, worm transferring, worm activation, and infection. During the phase of target finding and worm transferring, the worm is active over the Internet, making it possible for network based intrusion detection systems (NIDS) to catch the worm. The activities in the later two phases are limited to local machines and are harder to detect by NIDSes. This paper categorizes the characteristics of worms in target finding and worm transfer phases into four categories: target finding scheme, propagation scheme, transmission scheme, and payload format. Each scheme is further divided into sub-categories (see Figure 1). Each one of these categories will be discussed in the following sections.

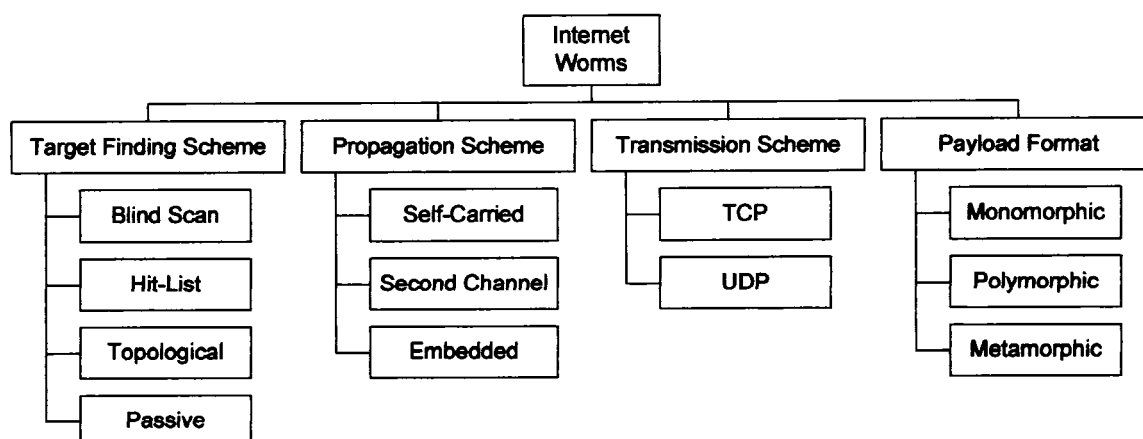


Figure 1 Categorization of Worm Characteristics

2.1 Worm Target Finding Schemes

The first step of a worm's life is to find targets. There are many different methods to find the next victim. One simple way is to use blind target scanning, which means the worm has no prior knowledge about the targets. The three types of blind target scanning are sequential, random, and permutation scanning. All these methods are based on chance and have a relatively high failure connection rate. Many worms use this method and many anomaly-based detection systems are designed to capture this type of worm. Blind scanning worms may be easier to implement and may spread fast, but are not very accurate. The miss rate can be very high. An improved version of a blind scanning scheme is to focus on local subnet scanning with information obtained from the current victim. Doing so can improve the hit rate of scanning.

The other way of finding targets is to use a pre-scanned list of vulnerable addresses called the hit-list. This way, the worm knows exactly where the target is. The hit-list can be generated stealthily before the release of a worm or obtained somewhere else. It can be contained inside the worm, or stored somewhere external for worms to query. The bigger the size of the hit-list, the harder it is to obtain and carry, but it is more accurate and may cause more damage. A pre-generated hit-list can be out of date since the Internet is changing all the time, but the speed and accuracy of the initial spread will improve greatly compared with a purely blind scan. Because of the high accuracy rate, the worm will cause very few anomalies on the network and, therefore, be hard to detect with customary anomaly-based NIDSes.

Staniford et al. (2002) have simulated a very fast spreading worm named the Warhol Worm that, with the combination technique of the hit list and permutation scanning, is able to infect most vulnerable systems in possibly less than 15 minutes. The hit list helps the initial spread and the permutation keeps the speed and hit rate high for a longer time than just using random scanning. An extended version of the Warhol Worm, which is equipped with a global size hit-list, is a flash worm. Staniford et al. (2004) simulated such kind of worms. The result of the simulation showed that a UDP flash worm can infect 95% of one

million vulnerable hosts in 510 milliseconds, while a TCP version of a flash worm can cause the same damage in 1.3 seconds.

Many hosts on the Internet store information about other hosts on the network and possibly reveal their vulnerabilities. A worm can use this information to gain knowledge of the topology of the network and use that as the path of infection. This makes the attack more accurate since the scanning and infection may look like normal traffic.

If a worm does not aggressively seek the target but patiently waits, it is said to take the passive approach; that is, instead of voluntarily scanning the network, it waits for potential victims to approach the machine where the worm resides and then reply with a copy of the worm. It may also wait for certain user actions to find the next victim. This method is slow, but it is very stealthy and hard to detect.

2.2 Worm Propagation Schemes

After the next victim is found, a copy of the worm will be sent to the target. There are different schemes of worm propagation. In *Taxonomy of Computer Worms* (Weaver et al., 2004, p. 11-18), three propagation schemes are mentioned: self carried second-channel, and embedded scheme.

Self-carried worm propagation is straightforward; it transfers the worm payload in a packet by itself. Other worms deliver through a second channel; that is, after finding the target, the worm first goes into the target, then downloads the worm's payload from a previously infected machine or the Internet. This can be achieved by installing a back door, using RPC or other applications. A more deceitful worm may append the payload after, or replace, the legitimate traffic to hide itself. This embedded propagation scheme is very stealthy. No anomalous events will be triggered, and it is hard for anomaly-based detection systems to detect.

2.3 Worm Transmission Schemes

Based on the transmission scheme, there are TCP worms and UDP worms. The major difference between these two types of worms is that TCP worms are latency limited and UDP worms are bandwidth limited.

All TCP connections require a three-way handshake to establish connection before transmitting. Therefore, after a host sends out a TCP SYN packet to initiate a connection, it must wait until it receives a corresponding SYN/ACK or timeout packet from the other end before it can take any further actions. Compared with UDP worms, TCP worms need the additional round-

trip time and the two 40-byte packets to establish the connection. During this wait time, the thread or process is blocked and can't infect other hosts.

UDP is connectionless, so UDP worms do not require a connection to be established before the infection can begin. The implementation of the worm is normally self-carried and is included in the first packet sent to the target. Since there is no wait time required like for TCP worms, UDP worms normally spread very rapidly, and the limit is the bandwidth. UDP worms often have to compete with each other for network resources (Moore et al., 2003, p. 33-39).

2.4 Worm Payload Formats

The term "payload" used here means the actual worm code. Traditionally worms send the payload in a straightforward, unchanged fashion. By matching the worm payload with the signatures in a database, signature-based detection systems can identify them. Some worms make the payload size variable by padding the payload with garbage data, but the signature will not change; this is still a monomorphic worm.

Worm authors can make changes in the payload to make them appear innocent to evade detection systems. They may fragment worm payloads differently and re-assemble the pieces at the target. This type of worm is also classified as a monomorphic worm. The term "polymorphic worm" used here

describes those worms that change their payload dynamically by scrambling the program. Therefore, in every instance the worm looks different but functions exactly the same way. With the changing appearance of the worms, it is very hard for traditional signature-based detection systems to detect such worms.

If a worm can change not only its appearance but also its behavior, it is a metamorphic worm. If the worm also uses a complicated encryption scheme to hide its true purpose, then it will be even harder to defend against (Glazer, n.d.).

2.5 Existing Internet Worms

In this section, four recent existing Internet worms, Code Red, Sasser, Slammer, and Witty, are discussed based on their characteristics.

2.5.1 Code Red I & Code Red II

Code Red I was first seen in July 2001 affecting computers running Microsoft's Internet Information Server (IIS) web service. In the first 20-25 days after getting into the machine, Code Red I uses a blind scan scheme that scans port 80 on random IP addresses to find other vulnerable machines, and then it launches a Denial-of-Service (DoS) attack targeting a set of IP addresses. The infected websites will display: "HELLO! Welcome to [http://www.worm.com!](http://www.worm.com) Hacked By Chinese!"

Code Red II was released one month later. It is a variant of the original Code Red. Code Red II no longer launches a DoS attack against pre-defined IP address, instead it installs a backdoor into the infected systems. It still employs blind scan but focuses more on the local subnet, and targets mainly systems with Chinese language setting.

Code Red I sends its payload in monomorphic format and has a signature starting with "GET /default.ida?NNNNNNN". Code Red II has a similar signature but replaces N with X. Both versions of Code Red are self-carried and transfer via TCP connections.

2.5.2 Slammer/Sapphire

Slammer, also known as Sapphire, was one of the smallest worms seen. It was found in January 2003 targeting Microsoft SQL server 2000 or MSDE 2000.

The Slammer worm uses UDP port 1434 to exploit a buffer overflow in the MS SQL server. The code size is 376 bytes. Adding the UDP header makes the worm 404 bytes long in total (Moore et al., 2003, p. 33-39). It uses a blind scan scheme where random generated numbers are used as IP addresses in searching for vulnerable hosts. To initialize the random number generator, Slammer uses GetTickCount() function from Win32 API. Sometimes the random generator returns values that are broadcast addresses, such as a.b.c.255, and causes all the

hosts in that network to receive the worm packets, making the spread of the Slammer worm more rapid. Like most UDP worms, Slammer is self-carried and has a monomorphic payload.

Slammer does not write to the disk of the infected machines; it only overloads the victim's system and slows down the traffic (F-Secure Computer Virus Information Pages: Slammer, n.d.).

2.5.3 Sasser

Sasser was released in April 2004, targeting systems running Microsoft Windows XP or Windows 2000 that have not been patched for the vulnerability of LSASS (Local Security Authority Subsystem Service). Sasser exploits a buffer overflow vulnerability of LSASS to gain access to the remote systems and to spread further. Sasser transfers with a second channel via a TCP connection and uses a monomorphic payload.

If Sasser successfully infects a system, it will act as an FTP server listening on TCP port 5554. Sasser then generates 128 scanning threads (Sasser B uses processes instead of threads) to find vulnerable systems using random IP addresses. The worm probes and tries to connect to the next victims through TCP port 445, then attempts to connect to the victim's command shell available on TCP port 9996. Once the connection is successful, the victim will download

the worm code from the attacker using FTP. The sizes of Sasser A through E are 15-16 Kbytes. Sasser F and later versions are larger; Sasser F is 74KB (Sasser Worm Analysis – LURHQ, n.d.) and Sasser G is 58KB (Secunia - Virus Information - Sasser.G, n.d.).

After the Sasser worm enters the system, it makes a copy of itself, stores one copy in the Windows directory, and adds itself to the Registry. Transactions through the FTP server are logged to "C:\win.log".

2.5.4 Witty Worm

The Witty worm was released in March 2004, targeting buffer overflow vulnerability in several ISS (Internet Security Systems), including RealSecure Server Sensor, RealSecure Desktop, and BlackICE. Witty took advantage of a vulnerability in the ISS Protocol Analysis Module (PAM) used for ICQ instant messaging.

Witty is a self-carried, monomorphic, UDP worm that employs a blind target finding scheme. It sends out UDP packets to 20000 random generated IP addresses on random destination ports from source port 4000, with a random packet size ranging between 768-1307 bytes. The code size of Witty is only 637 bytes, and the rest of the payload is padded with data from the system's memory. This padding doesn't change the monomorphic format of Witty. The payload

contains the text “(^.^) insert witty message here (^.^)” and that is why it is named Witty. Witty randomly writes data onto the disk of infected machines (F-Secure Computer Virus Information Pages: Witty, n.d.).

It is harder to detect Witty worm than worms with fixed size packets targeting fixed destination port numbers because of its random characteristic. The size of Witty worms is larger than Slammer worms (some can be doubled), but they spread faster than Slammer. This proves that size is not always the bottleneck for the spreading of UDP worms (Shannon & Moore, 2004).

Table 1 Existing Internet Worm Implementation

	Target Finding Scheme	Propagation Scheme	Transmission Scheme	Payload Format
Code Red	Blind*	Self-Carried	TCP	Monomorphic
Slammer	Blind	Self-Carried	UDP	Monomorphic
Sasser	Blind	Second-Channel	TCP	Monomorphic
Witty	Blind	Self-Carried	UDP	Monomorphic

*Code Red II focusing on local subnet scan

2.6 Multiplatform and Multiexploit Worms

Up until now, most worms attack only one type of operating system, and often target a single vulnerability, so the network administrators only need to patch one type of system after receiving the worm alert. If a worm can perform

multiplatform attacks, it will be harder and more complicated to defend against.

And since there's more work to be done, the response time will be slower as well.

If a worm infects the victims by using multiple vulnerabilities, a single worm can cause greater damage at a faster rate. And the work of patching will also be harder.

3. Defending Against Internet Worms

Now that the nature of Internet worms is known, the next question is how to defend against them. This task breaks down into worm detection and worm containment. Detection and containment systems can be implemented at different locations in the network and thus give different scopes of defense. Worm defense is outlined in Figure 2 and each topic will be discussed in the remaining sections of this paper.

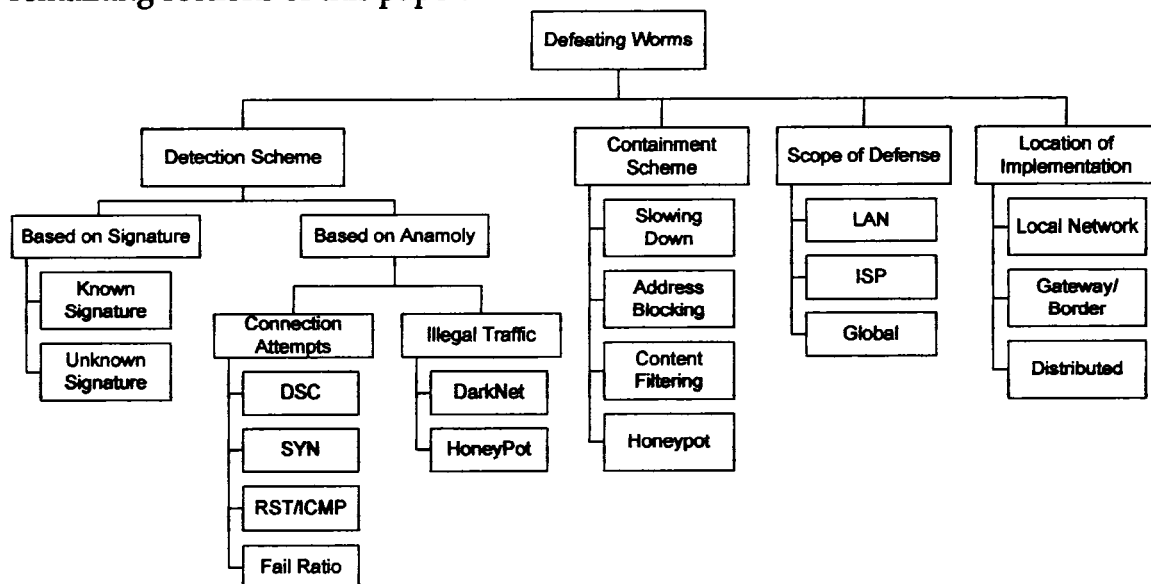


Figure 2 Categorization of Internet Worm Defense

4. Worm Detection

Based on the parameters used for detection, detection algorithms can be roughly divided into signature-based and anomaly-based schemes as seen in Figure 2. There are many proposed algorithms for both schemes; this section first introduces signature-based detection and then talks about anomaly-based detection.

4.1 Signature-Based

Signature-based detection is a traditional technique used for intrusion detection systems and is normally used for detecting known attacks. There are different definitions of “worm signature”. In this paper, our discussion will focus on content signature, which is often a string of characters that appear in the payload of the worm packets as a part of the attack.

No knowledge of normal traffic is required, but a signature database is needed for this type of detection system. This type of system doesn't care how the worm finds the target, how it propagates itself, or what transmission scheme it uses. The system looks at the payload and identifies whether or not it contains a worm. Since every packet will be examined, signature-based systems can catch worms that employ self-carried or second channel propagation schemes. The

embedded worms may not be detected because the payload can be different from worm to worm, depending on the embedding method used.

One big challenge of signature-based IDS is that every signature requires an entry in the database, and so a complete database might contain hundreds or even thousands of entries. Each packet is to be compared with all the entries in the database. This can be very resource consuming and doing so will slow down the throughput, thus making the IDS miss packets and possibly miss attacks as well. At the same time, this type of IDS is still vulnerable to unknown attacks.

Some algorithms were proposed to detect unknown attacks by generating signatures in real time. Hardware implementations were also proposed to improve the system throughput. Lockwood and Madhusudan (2004) introduced an algorithm to detect frequently-occurring strings in packets and use them as signatures for detection. In this system, a signature detection device (DET) sits between the router and subnets and monitors the traffic flow to detect Internet worms. It is implemented in hardware and the throughput was improved by parallelism and hashing. The EarlyBird system used a similar approach to find frequently occurring sub-strings in packets (Singh et al., 2003). Not only can these algorithms detect unknown worm signatures, they are also useful if, instead of sending the payload in a whole or fragment in the same fashion every time, the worm fragments the payload differently each time since it is to match

the substring of the payload. It may also catch worms that are embedded in legitimate packets for propagation.

No matter whether the signature is known or unknown, most signature-based detection algorithms target monomorphic worm payloads only, and have no defense against polymorphic worms that change the payload dynamically. Newsome, Karp, and Song (2005) proposed an algorithm to automatically generate signatures for polymorphic worms. They found that even though polymorphic worms change the payload dynamically, certain contents will not be changed. Such contents include protocol framing bytes (e.g., GET and HTTP protocol indicator), and the value used for return address or pointer to overwrite a jump target. Based on this characteristic of polymorphic worms, they divide signatures into tokens. The system generates tokens automatically and detects worms based on these tokens. An algorithm that detects polymorphic worms can detect monomorphic worms as well, but not the other way around, so it is a more thorough approach.

Unknown signature detection generates signatures from the traffic flow. Even though it takes time to generate signatures, compared with known signature-based detection system, it may be less efficient when facing known worms, but it can detect newly released worms and possibly catch other kinds of Internet attacks such as DDoS attacks. It may also help in detecting embedded

worms since the signature can be part of the packet payload instead of the whole content. Unknown signature detection systems often do not store all the signatures. As a previously generated signature ages, it will eventually be eliminated from the database so the database does not just grow bigger and bigger. This conserves the system resource from the processing time of comparing signatures and the storage space of the database.

4.2 Anomaly-Based

The signature of a new worm is unknown before it is seen, and it is difficult to draw conclusions based on a small number of packets. All fast spreading worms seen so far create large traffic volume and most of them employ blind scans. Lots of the scans target non-existing addresses and closed ports. Since most networks behave in a particular and consistent fashion most of the time, when there are lots of abnormal phenomena occurring on the network, it often means something is wrong.

Signature-based detection checks the payloads of the worms to generate and match signatures. Anomaly-based detection does not care about the payload format or content; instead, it checks the header of the packets to define the type of connection the packet belongs to. It observes the network traffic volume and the monitored hosts' behavior. The three most common purposes for a packet,

sent or received by a host, are: to initialize a connection, indicate a failed connection attempt and to send data via an already established connection. The system may also keep track of the traffic between source and/or destination addresses and try to find the scanner.

Anomaly-based detection systems detect abnormal behaviors and generate alarms. This technique often requires the definition of normal network behavior, and this requires a training period before the system can be effective in protecting the network. If an attack is crafted carefully, it may possibly train the system to take anomalies as normal or trigger false alarms. The big challenges of anomaly based detection systems are defining what a normal network behavior is, deciding the threshold to trigger the alarm, and preventing false alarms. The users of the network are normally human, and people are hard to predict. If the normal model is not defined carefully, there will be lots of false alarms and the detection system will degrade in performance.

As seen in Figure 2, anomaly-based algorithms are categorized based on connection attempts and illegal traffic. Under connection attempts, there are connection count/traffic rate, failure connection rate, success/failure ratio, and destination-source correlation. Under illegal traffic, there are motoring darknet and Honeypots. The following sub-sections will explained these categories in detail.

4.2.1 Traffic Rate/Connection Count – TCP SYN

Worms send out large numbers of scans to find victims. Keeping track of the outbound connection attempts is a traditional way to detect scanning worms. TCP/IP protocol stacks require a host to send out a TCP SYN packet to initiate a connection (shown in Figure 3, part A), and this is used as the parameter for connection count detection. The idea is if the number of SYN packets being sent by a certain host exceeds a threshold value within a period of time, the host is considered to be scanning.

This method is used in many older algorithms and some commercial IDSes, such as the older version of snort (n.d.). Tracking TCP SYN packets may be able to catch most of the active scanning worms with most of the scanning schemes, but it is very easy to cause false alarms. It is not widely used nowadays because it is not very accurate, nor is it efficient. Also, if the system logs TCP SYN only, then it is useless against UDP worms.

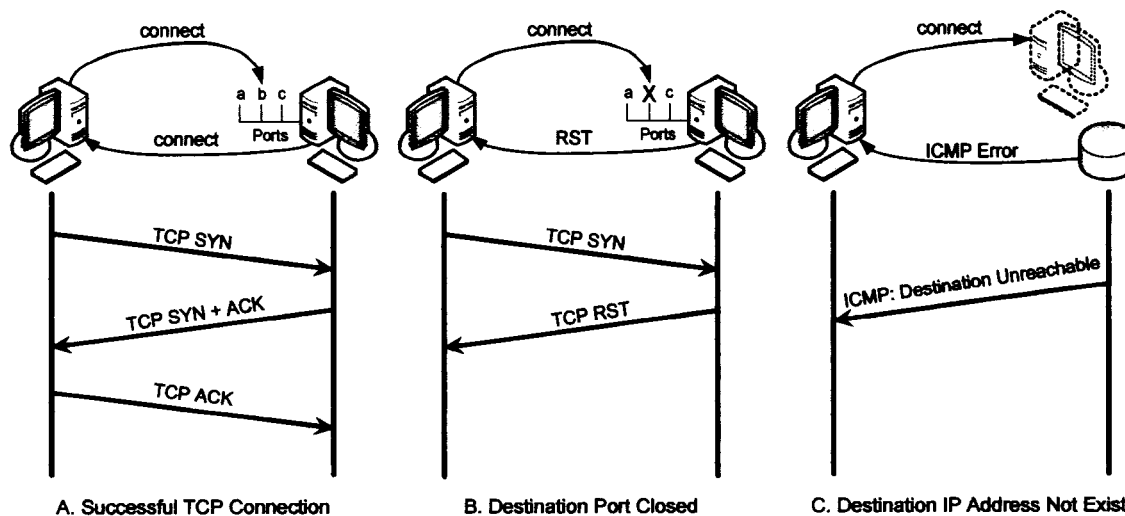


Figure 3 Connection Attempts

4.2.2 Failure Connection Rate – TCP RST & ICMP

Many worms implement blind target finding schemes. No matter if the worms use sequential scan, permutation techniques, or randomly generated IP address and/or random ports scan, the scan will include unused address spaces and closed ports, causing the connection attempt to fail. While ordinary users access the Internet mainly via domain names, they are less likely to encounter failed connection attempts.

Attempting to connect to a non-existing IP address or an existing address with the target port closed, the connection attempt is considered failed. According to TCP/IP protocol stacks, if the destination host does not exist, an ICMP host unreachable packet is returned (shown in Figure 3, part C). A TCP RST packet is returned when the connection targets an existing host with the

destination port closed (shown in Figure 3, part B). With these characteristics of TCP/IP protocols, keeping track of these error messages will work well against blind target finding schemes. The scan is blind, so the failure rate is higher than normal. In the case of hit list, topological, or passive-scanning worms, the failure rate will not be very useful because the scan and spread of worms have valid targets and will not cause these error messages.

Compared with the traditional method of detecting TCP SYN packets for connection count, it is more efficient and accurate to detect active scanning worms depending on failed connections. This approach may be useful for both TCP and UDP worms. ICMP error messages will be sent for both TCP and UDP connection attempts, but TCP RST works with TCP connections only. Another thing to consider is that some border routers or gateway systems block off ICMP error messages. In this situation, this method will be less useful.

Berk, Bakos, and Morris (2003) proposed a global detection algorithm based on ICMP destination unreachable error messages. Routers often generate ICMP error messages to notify the source that the target IP address does not exist on the network. By forwarding these messages to a central collection point, an alert can be generated when the number of such error messages reaches a certain threshold. The DAW (Distributed Anti-Worm) architecture (Chen & Tang, 2004)

identifies scanning sources by keeping track of both TCP SYN and RST packets, dealing with TCP worms only.

If the source address is forged in the packet header, then it will be very difficult to detect the scan source. This can be used as a feint attack by the worm's authors. If the system administrator actually traces back the fake information given in the header, it will be a waste of effort. The worm may also use this technique to trigger false alarms. Forging header information is used in many Internet attacks. Although this technique is not commonly seen in worm attacks yet, it is still an issue worth paying attention to. The Worm Early Warning System (WEW) proposed by Chan and Ranka (2004) considered this problem. This architecture with gateway and hash is introduced to not only detect the error messages from failure connection attempts, but also to verify whether the source address is legitimate or forged. The system only takes action on failed connection attempts that are sent from existing source addresses.

4.2.3 Ratio of Success and Failure Connections

Instead of counting the failure or successful connection attempts, some believe it is the ratio or correlation of succeeded and failed connections that matters. Connection counts, whether they succeed or fail, depend on the usage of the Internet and the size of the network to be effective. If the usage is too low

or the network is too small, using connection counts as a detection parameter may be less accurate. Thus, both success and failure connections should be taken into account. When the percentage of failed connections is large enough, this is said to be an anomaly and an alert will be generated. Similar to the previous method of detecting failure connection attempts, this method works well against blind scans, but not with other scanning techniques where the targets are specific and legitimate.

Jung et al. (2004) proposed the Threshold Random Walk (TRW) algorithm, derived from Sequential Hypothesis Testing. The algorithm says: for a given remote host R, if it tries to make a connection to a local host L, this attempt can be a success (marked as 0) or a failure (marked as 1). With a sequence of these results of connection attempts, the system can decide whether the remote host is a scanner based on the test of hypothesis. This algorithm requires very few packets (4-5 packets only) to draw a conclusion and does not require training the system. It focuses on detecting TCP traffic only.

Weaver et al. (2004) introduced the concept and importance of hard-LANs. They explained the three limitations of the TRW algorithm: it is offline, it requires infinite states, and it requires potentially unlimited memory access time. Because of these limitations, they proposed an improved algorithm that instead of identifying a connection as completed or failed, considers all new connections

as failures and changes the status to success when there is a response. It also keeps track of UDP connections. This “guilty until proven innocent” method keeps a counter for every IP address, starting with a miss. If the connection succeeds, the corresponding counter decrements. The counter increments if the connection is failed. When a counter is greater than 10 or other pre-defined value, the corresponding system is considered a scanner. This algorithm requires a fairly small amount of memory and is suitable for integration into switches or other low-cost networking devices.

Monitoring the connection status, no matter if it is a success, failure, or both, often requires keeping track of each host’s and/or each connection’s information. If the monitored network is large, this can be very resource consuming.

4.2.4 Destination-Source Correlation (DSC)

The Destination-Source Correlation (DSC) algorithm is a two-phase local worm detection algorithm aimed at detecting fast spreading scanning worms (Gu et al., 2004, p. 136-145), (Qin et al., 2004). The algorithm keeps track of SYN packets and UDP traffic of the source and destination. It is illustrated in Figure 4, where for every port, if a host inside the monitored network previously receives a packet on a certain port (e.g., port 25 in the illustration) and then starts sending

packets designated to the same port from which it previously received packets, a counter is incremented. When the counter reaches a certain threshold, an alert will be issued.

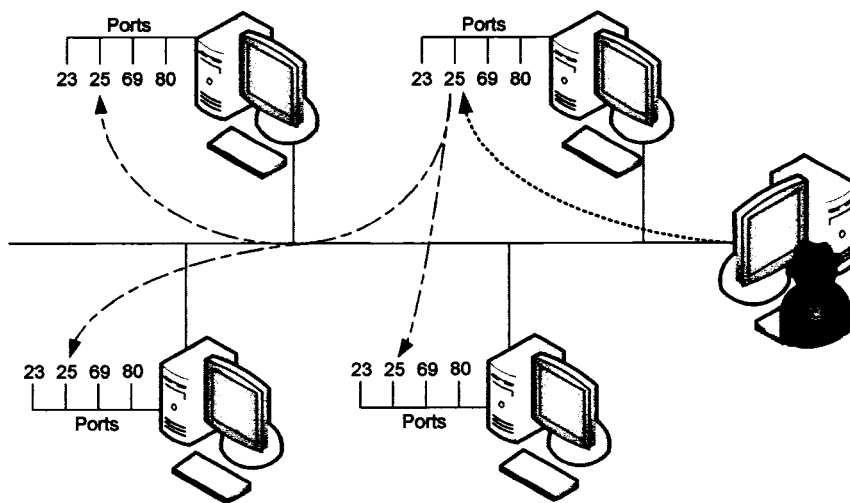


Figure 4 Illustration of Destination-Source Correlation Scheme

The DSC algorithm is able to detect almost all types of scans, as long as the scan is frequent enough (based to the threshold) and the infection of the worm is targeting the same port. It can detect aggressive scans including blind and topological scans. The effectiveness of passive scan depends on the incoming traffic rate since it relies on the interaction with the worm. It works for both TCP and UDP worms.

4.2.5 DarkNet/Unused Address Space

Worms using a blind scan generate random numbers for target addresses. There is a very high chance that these addresses are unused. Monitoring unused address space instead of used ones is another approach. This is a branch of anomaly-based detection since scanning or connection attempts toward non-existing addresses are abnormal behaviors in a regular network.

The monitored address space has to be big enough to have this method be useful. Chen, Gao, and Kwiat (2003) presented the Analytical Active Worm Propagation (AAWP) model, which simulates the propagation of random scanning worms. Using this model, they derived the size of address space needed to detect active worms. They suggested that an address space of 224 IP addresses is large enough to detect worms effectively, and an address space smaller than 220 addresses will be too small to obtain a realistic result of the spread of worms.

A scanner host is normally a host infected by worms. In other words, it is a victim itself. Wu et al. (2004) proposed an algorithm based on the number of victims. The victim is defined as: "the addresses from which a packet is sent to an inactive address" (Wu et al., 2004). This means if an IP address sends packets to an unused IP address, then this source IP address is considered a victim. To prevent false alarms, they combined this definition with the Two Scan Decision

Rule (TSDR), which means if the system captures two packets sending to unused IP addresses from the same host, the host is a victim. When the victim count reaches a certain threshold, the system will generate a worm alert.

Monitoring the unused IP address can find worms using a blind-target-finding scheme. But again, it is not very useful against a hit-list, topological or passive scans. Even though the system still needs to check all the packets in the traffic flow, the anomaly event should be less than monitoring used address space. There is less information to store in the database as well. This method works for both TCP and UDP worms since both connection schemes require IP addresses.

4.2.6 Honeypots

Honeypot technology can be used for anomaly-based worm detection. Honeypot is a vulnerable system on the network that does not provide any real services. In the book *Honeypots: Tracking Hacker*, the author, Spitzner (2002, p. 40), defines the Honeypot as "A Honeypot is [sic] security resource whose value lies in being probed, attacked, or compromised." Figure 5 illustrates the setup of Honeypot where it appears as a normal vulnerable machine on the same network, just like other servers and hosts to lure attackers.

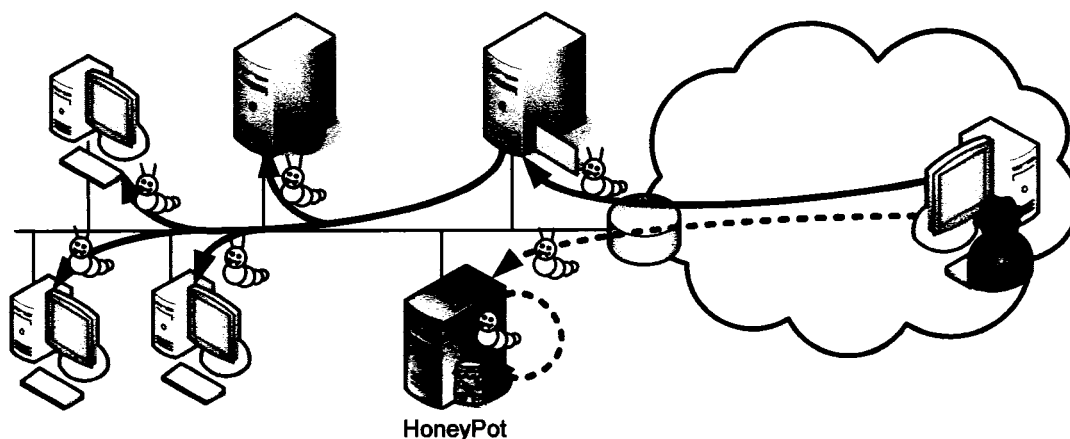


Figure 5 HoneyPots Used in Worm Detection and Containment

There are many different implementations based on the level of interaction the Honeypot provides. In a normal situation, no traffic is supposed to come toward the Honeypot; therefore, any traffic targeting the Honeypot is considered an anomaly, and may be an attack. Compared with other IDSes, Honeypot systems gather less but higher quality data because every piece of data gives information about probes or attacks.

HoneyPots can detect blind scan worms using the same approach that as monitor unused address spaces. HoneyPots are also able to defend against hit-list-scanning worms. If the hit list is generated automatically, the pre-scan may very likely include Honeypot systems because they appear as normal vulnerable hosts on the Internet. HoneyPots can be useful against topological worms if other working hosts on the network contain proper information about the Honeypot and let the worm find it. However, HoneyPots are not useful against

passive worms because Honey Pots only sit and wait but do not initiate connections. As long as they are properly configured, Honey Pots can detect both TCP and UDP worms.

The Virtual Honey Pot has been used for worm detection (Dagon et al., 2004), (Qin et al., 2004). In an emulator, they created a minimal Honey Pot that uses multihome to cover a large address space, called HoneyStat. It is used to gather information about worms as well as capturing worms. Their HoneyStat simulation runs on VMware GSX, if there are 64 virtual machines running windows and every window has 32 IP addresses, then a single node can have 2^{11} IP addresses. And the hardware requirement for such a system can be as low as 32MB RAM and 770MB virtual drives to capture worms. HoneyStat generates alerts based on the correlation of three types of event: memory, disk write, and network events. An example of a memory event is buffer overflow. An example of a network event is a downloading of some malicious code. And an example of a disk event is writing to registry keys or critical files.

HoneyD, a low interaction open source Honey Pot daemon that supports both UNIX and Windows platforms, can detect and log connections on any TCP or UDP ports. When a connection to HoneyD is established, HoneyD will emulate the configured personality or operating system and port behavior based on the configuration script. HoneyD can emulate any of the 437 existing

operating systems and any size of network address space with desired topology. Provos (2004), the author of HoneyD, took HoneyD and built a system on top of it for worm detection. It can detect intrusion as well as be configured to replay incoming packets to higher interaction Honeypots for analysis of unusual activities.

A Honeypot that uses scripts is more flexible than one with limited configuration settings. No matter what, a Honeypot has a narrower view, since it can only see the traffic coming towards addresses it simulates. Honeypots can also be used for containment. Honeypot will be discussed in more detail in section 5.3.

4.3 Combination Usage of Detection Schemes

An unknown signature-based detection system may defend against zero-day attacks as well as known worms. But it takes time to generate signatures, and since there are defined signatures already, why not just use them? A system equipped with a known worm signature database and an additional real-time signature generator may be more comprehensive; it would be even better if it also has the capability of detecting polymorphic worms.

A worm can be detected on the network during the phase of target finding and transmission. Different detection methods catch different types of worms,

and no one current algorithm is perfect. A hybrid system with the integration of both anomaly-based and signature-based types of detection techniques will give a broader and more complete view to a detection system. Ideally, a hybrid system should check for both the signature and network anomalies, and have a HoneyPot to aid in the detection and gathering of worm information.

Qin et al. (2004) combined HoneyStat with a modified version of DSC. Based on the DSC algorithm, the system monitors IP or MAC addresses to defend against worms using IP spoofing. HoneyStat is used to gather statistical data about the attack. Since DSC only captures scans with the same port and HoneyStat can capture scans with different ports, HoneyStat is also used to capture worms and can cover what DSC cannot see.

The EarlyBird system combined an unknown signature-based algorithm with a DSC-like anomaly-based algorithm for its detection system (Singh et al., 2003). An alarm is generated when packets with similar contents are: sent to a number of destination IP addresses; coming from a large number of source IP addresses; or sent from a number of hosts to a large number of hosts (source and destination IP address pairs). Staniford et al. (2004) also indicated that this type of system may be able to capture Flash worms.

4.4 Summary on Worm Detection

In this section, various methods and efforts in worm detection are discussed. The systems can be classified as signature based or anomaly based, and then further organized into several sub-categories based on the algorithms. Different detection schemes are useful against certain worm characteristics. This is summarized in Table 2.

As shown in Table 2, current algorithms have less chance to detect worms with certain schemes. Most anomaly-based systems focus on detecting blind scan worms, which is so far the most commonly seen technique for active worms. This is not the case for all worms. There is no algorithm that can detect passive scanning worms because these worms do not trigger any error messages and normally don't cause high traffic volume. Worms that use embedded propagation schemes are hard to detect because if the worm payload is appended to the packet content, then the signature, as a whole, may be different for each worm unless the system breaks the signatures into pieces and inspects them individually with consideration of the other pieces. Passive scans and embedded payloads are often used together in worm implementations. Doing so will make the worm very stealthy, but the spread is slow and causes less damage, making it less of a threat.

Table 2 A Comparison of Worm Characteristics with Anomaly Detection Methods

Method of Detection	Characteristic of Worms		Target Finding Scheme			Propagation Scheme			Transmission Scheme		Payload Format		
	Blind	Hit list	Topo-logical	Passive	Self-Carried	Second Channel	Embedded	TCP	UDP	Mono-morphic	Poly-morphic	Meta-morphic	
Signature	Known Signature	-	-	-	✓	✓	maybe	✓	✓	✓	×	×	
	Unknown Signature	-	-	-	✓	✓	maybe	✓	✓	✓	×	×	
	Token based Signature	-	-	-	✓	✓	✓	✓	✓	✓	✓	×	
Anomaly Based	Destination Source Correlation	✓	✓	×	×	✓	-	✓	✓	-	-	-	
	TCP SYN - Connection Count	✓	✓	✓	×	✓	-	✓	maybe	-	-	-	
	Failed Connection Attempts	✓	×	×	×	✓	-	✓	maybe	-	-	-	
	Ratio of Success/Failure Attempts	✓	×	×	×	✓	-	✓	maybe	-	-	-	
	Motoring DarkNet	✓	×	×	×	✓	-	✓	✓	-	-	-	
	HoneyPots	✓	✓	maybe	×	✓	-	✓	✓	-	-	-	
Hybrid	Modified DSC + HoneyStat *	✓	✓	✓	×	✓	-	✓	✓	-	-	-	
	Early Bird (unknown sig+DSC)	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	

There aren't too many systems that are able to catch worms using topological or hit-list scanning schemes because they cause fewer or no failure connections. But it is not impossible since they often still generate large traffic volume.

Only token-based signature detection is able to detect polymorphic worms. And if the worm fragments itself differently every time it attacks, only systems that are able to handle partial signatures can catch them. Metamorphic worms are hypothetical and, currently, there is not a way to detect them, but the solution should be found before they are seen. There has been a lot of work done in worm detection, but there are still more challenges to face in this field.

5. Containment

Detecting worms is important, but it is just as important to stop them from spreading. If a worm can be found ahead of the infection, say if the system detects the worm by its signature at the border gateway, then the system can try to block the worms and prevent any machine from being infected. But this is not the case most of the time. System administrators and users don't realize there is a worm attack until a victim is having some abnormal behavior and the damage has already been caused. Reacting quickly and minimizing the damage after the infection is as important as preventing and detecting worms.

At this point, the worm is found to exist. Those characteristics used for detection no longer matter. The containment systems aim to eliminate the worms. Many containment methods have been proposed in the past few years. These methods are summarized in Figure 2 and are categorized as the following: slowing down, blocking, and decoying worms.

5.1 Slowing Down Infection

The first approach is to slow down the spread of worms and give time for human reaction and intervention. Several methods were presented, such as using a feedback loop to delay suspicious traffic (Dantu, Cangussu & Yelimeli,

2004, p. 419-423), and using rate limiting techniques at different network levels to slow down the infection (Wong et al., 2004, p. 73-82).

These proposals suggest ways to slow down the speed of worm propagation, but worms normally spread at an extremely high speed. Within minutes, worms can spread through whole networks. Human reaction time is a lot slower than a well-designed computer system. A good worm containment system should be automated and should not only slow down an infection, but it should try to stop it.

5.2 Blocking

Automatically blocking off certain worm-like traffic is another method of containment. When a worm-like behavior is discovered, the source has to be isolated from the rest of the network to prevent more machines from getting infected. Blocking is often used together with the slowing down method. The system can first try to slow down the infection when the first level threshold is met to avoid false positives, and if the situation becomes worse and the second threshold is reached, then blocking will be utilized.

There are two major approaches for blocking. One is to block off packets with certain contents, and the other is to block traffic to and/or from certain addresses. Moore et al. (2003) simulated containment with both methods and the

result shows content blocking is more efficient and more effective than address blocking. For either blocking scheme, the challenge is to define when and whom to block to avoid false alarms.

5.2.1 Address Blocking

Address blocking means when a host is identified as a scanner or victim; any traffic from that host address is dropped. This technique is normally implemented at the border router/gateway, so the containment system is able to perform this task. The system will need to keep a black-list, which contains addresses to be blocked.

Several algorithms have been proposed for address block containment. One system mentioned in the previous section, in the success/failure connection ratio detection section, is designed to block the address when the miss and hit ratio is greater than 10 or other pre-defined value (Weaver et al., 2004, p. 70-76). The DAW architecture (Chen & Tang, 2004) mentioned before also implements address blocking. If certain hosts persistently have high failure rates, the address is blocked and the system waits for human intervention for unblocking or further analysis.

Address blocking has to be implemented very carefully to reduce false alarms. In the case of false positives, the non-infected hosts might get blocked off

and if the attackers use this loophole, they can trigger this network malfunction and bring damage to the organization.

5.2.2 Content Blocking

Content blocking is used in most signature-based detection systems. If a packet's content matches a worm signature, the packet will be dropped automatically. The system can also make decisions based on the type of packet obtained from the header information. Furthermore, certain error messages might not be let through to avoid giving information to a scan source.

In the containment algorithm proposed by Weaver et al. (2004), after the traffic anomaly reaches a certain threshold, the system will only allow packets from already established connections to go through. Scan-like packets such as TCP SYN to initialize connection and TCP RST that is not from a pre-established connection will be dropped.

Content blocking allows the legitimate traffic to pass while stopping infection-like traffic from going to its destination. It may cause less harm when false alarms arise.

5.3 Honeypot to Decoy

Honey pots were originally designed to lure the attackers as a non-existent host that appears to be valid. In the same way, Honey pots can be used as decoys to lure Internet worms. Worms aggressively find victims to infect; why not just let them find and infect some fake hosts and leave the real machines alone?

Figure 5 shows how Honey pots can be used to contain Internet worms. The top part of the illustration shows worms infecting one machine then propagating and infecting more hosts on the network (shown in solid lines). The bottom part of the figure illustrates the case when a worm infects a Honey pot machine; it stays there without infecting more hosts (shown in dotted lines). This is because the targets that the worm finds and tries to infect are all simulated by the Honey pot, which traps the worm inside.

In *A Virtual Honey pot Framework*, Provos (2004), noted that HoneyD not only can be used in worm detection, but also be configured to appear as a host having vulnerable applications to decoy and control the spread of worms. HoneyD can appear to occupy a large network address space, so it can be used to lure the worm into thinking it is infecting actual hosts, but it is actually attacking the Honey pot, thus slowing down the infection of real hosts on the network. These Honey pots should be installed and activated as soon as a worm is detected. The earlier they are activated the less damage will cause. In the research of

Provos, if the Honeypot starts 20 minutes after the beginning of the worm spread, with a deployment of about 262,000 virtual Honeypots, the system is able to stop the worm completely.

6. Fighting Worms in Different Scopes

Different detection and containment systems are designed for different places of the Internet with different views. A system installed at a LAN gateway has the scope of that local network and can monitor traffic coming in and going out of that network. A system implemented inside an ISP can monitor multiple LANs and has a broader view. And a distributed system may gather data across a wider coverage and monitor all the traffic on the Internet to obtain a global view.

This paper discusses worm detection and containment in the scope of LAN, ISP, and global as shown in Figure 2. These systems can be located inside the network, at a network border, or scattered on the Internet.

6.1 Location of Defense

A common point to place IDS is at the network boundary such as a gateway or border router. Sitting at the edge of a network, the IDS can inspect all traffic going in and out of the network to discover suspicious packets. This is useful if the system employs signature-based detection algorithms since the system needs to match the content of each packet with the signature database. It is also good for many anomaly-based systems because many detection algorithms are based on header information and the border routers check the

headers of every packet for routing purposes. Some anomaly-based systems can also be located inside the network, depending on the parameters used for detection. For example, Honeypots are normally implemented inside the network or at the DMZ and appear to be a regular host or server.

A detection algorithm implemented at the border router is good for defending purely random scanning worms and have less of a chance to catch worms that employ a local network preferred scan. Containment is normally better located at the border than inside the network because most of the algorithms try to block off or slow down certain traffic and it is easier to execute at the gateway of the network.

In their research on slowing down Internet worms using rate limiting, Wong et al. (2004) tested rate limiting at individual hosts, edge routers, or backbone routers in the simulation. The result shows that rate control at backbone routers is as effective as implementation at all the hosts that the backbone routers covered. However, it is more complex to install rate limit filters at every single host than only at the backbone routers. If it is in an enterprise environment, they suggest installing rate-limiting filters at both the edge router and some portion of the individual hosts to have better protection.

Instead of having the detection or containment system at one single location, distributed sensors and containment systems also have their advantage.

A distributed system can cover a greater portion of the network to have a broader view and possibly stop the worms faster and more effectively.

Stolfo (2004) and his Ph.D. students from Columbia University are working on the Worminator project that detects, reports, and defends against early attack events. This ongoing project is a collaboration of several academic institutions including Columbia University, GIT, FIT, MIT, and Syracuse University. They use the Antura sensor and the Columbia IDS PAYL (Packet Payload Anomaly Detector) sensors (Worminator Projector, n.d.) to detect stealthy scans and probes from outside of the firewalls. Analyzing the feedback of these sensors provides a greater picture of worm behavior.

In *Internet Quarantine: Requirements for Containing Self-Propagating Code*, Moore et al. (2003), simulated Internet worm containment in different percentages of customer Autonomy Systems (ASes) and different coverage of top ISP's ASes. The result shows that implementing in the ISP ASes is far more effective than the customer's ASes. In order to have the containment to be useful, the paths covered by the top 100 largest ASes have to be included. This means almost all Internet paths have to employ some containment system and require a wide cooperation among ISPs, which is very difficult to achieve.

6.2 Scope of Defense

The different locations of implementation give different scopes of worm detection and containment. Figure 6 illustrates the correlation of these different levels. Most IDSes are designed for local area or enterprise network detection and containment. The local area or enterprise network is a clearly defined entity, and it is normally controlled by one single organization, which has a central management when it comes to making decisions on the type of IDS to implement.

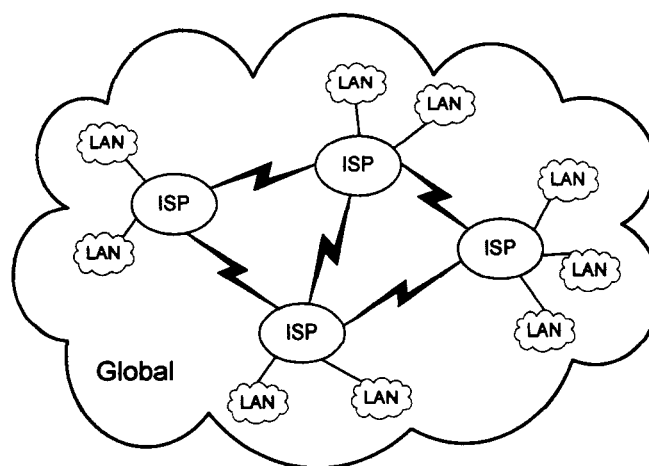


Figure 6 Different Scopes of Detection and Containment

A higher extent, that is well defined, is the scope of an ISP or Autonomy System (AS), which has multiple customer networks connected to it. After detecting worms from certain customer networks, the ISP can slow down or block off partial traffic from that network to prevent worms from spreading to

other customer networks. The detection might be more complex for signature-based algorithms because it has to deal with large amounts of traffic, so anomaly-based algorithms may be more feasible. Essentially, the larger the coverage is, the more accurate the normal model definition would be. Distributed Anti-Worm architecture (Chen & Tang, 2004) is work currently being done in this scope, which is implemented with anomaly-based detection and containment inside an ISP network.

Different parameters are used for different scopes of detection. As the scope grows bigger, the detection may be rougher, but the containment is more effective. Worms spread at a very fast speed. The damage of a worm outbreak is normally very broad, often across countries. The previous sections also described how many of the worm detecting algorithms are implemented based on monitoring larger size networks (2^{20} nodes or more). Moore et al. (2003) also showed that worm containment is only practical if a large fraction of the Internet unites and works together. All this tells us that a global scope is necessary in defending worms.

The idea of a Center for Disease Control (CDC) for global scope detection was brought up by Staniford, Paxson, and Weaver (2002). They believe that the CDC should have the following roles: identify outbreaks, rapidly analyze pathogens, fight infections, anticipate new vectors, proactively devise detectors

for new vectors, and resist future threats. This center should be deployed across the globe. There are real benefits to this approach when “one’s allies are awake and working while one sleeps.”

Qin et al. (2004) suggested that CDC is not very practical if used by itself. One reason is privacy, where not all organizations are willing to share their data with others. Another reason stated in the paper is that the architecture of CDC requires a victim, one participant won’t hear about the worm outbreak until there’s a victim, and this victim could be any participant.

The detection system utilizing ICMP error messages (Berk, Bakos, & Morris, 2003) mentioned in the previous section is another system that tries to obtain a global scope. This global-scale worm detection and analysis system is based on ICMP destination unreachable error messages forwarded by routers. This method is not possible until there are enough participating routers. To deal with such large amounts of data, this central point will need a considerable amount of resources for processing. The global scope is essential, yet very hard to achieve mainly because: it requires wide cooperation among ISPs, organizations or countries; there is the privacy issue; and these entities might have conflicts with each other and make cooperation very difficult.

Detection and containment has to be implemented with a hierarchical approach. Most of the enterprises already have their own security systems to

protect their network. Other local area networks should do the same to defend against attacks at the lowest level. Detection is more detailed at this level, and if the networks can flag the ISP when worms are found, ISPs can then confirm this situation and start the containment procedure. ISPs have a great position in worm containment since they are the junction of data exchange. At the same time, the ISP can alert other ISPs of their findings about the worm and take necessary precautions. Since a worm outbreak can be a worldwide disaster, everyone should work together.

7. Flash Worms Detection and Simulation

Among all the Internet worms, the flash worm, the fastest possible worm, is the most feared. In order to test the performance of this worm, a simulation is necessary. Section 7.1 and 7.2 discuss the design, challenges, and solution for this conceptual worm. Section 7.3 and 7.4 discuss the simulation issues of the flash worm as well as a network environment. A hybrid hierarchical model is proposed and discussed in Section 7.5.

7.1 Flash Worms

A flash worm is a hypothetical Internet worm that uses a hit-list target-finding scheme. This hit list is supposed to be global in size, meaning it should include all the machines on the Internet that are vulnerable in certain areas.

The hit-list is created before the release of the worm. It can be generated with a stealthy scan or obtained from somewhere else, such as a vulnerable application security report. After determining all the vulnerable hosts, the worm author can then build an infection tree to efficiently send the worms to all the addresses on the list. The hit-list does not need to be delivered to every instance of the worm. Since it has a tree structure, every node only needs to infect its children. The children can then further infect their children and so on. This divide and conquer approach can reduce the size of the hit-list efficiently as the

infection goes on. The worm author just needs to choose hosts with higher bandwidth for the first few levels to ensure the speed of initial spread.

As previously discussed, flash worms can be spread at a very fast speed. Staniford et al. (2004) have simulated flash worms with a mathematical approach. The result of the simulation showed that a UDP flash worm could infect 95% of one million vulnerable hosts in 510 milliseconds, while a TCP version of flash worm can cause the same damage in 1.3 seconds.

Flash worms have not yet been delivered to the Internet, but they are already attracting lots of attention because it is believed that they will be the fastest possible worm. Many features have been suggested to make it faster, better, and more threatening. Since an infection tree is selected ahead of time, the worm author may be able to design the route of infection to evade the detection and/or containment systems, and create serious damage in a short time.

7.1.1 Challenges of Flash Worm Implementation

There is no perfect worm. Flash worms are very powerful, but have not yet been seen in the wild. There are challenges to writing a worm like this. Since the hit-list is pre-generated, it can go out of date easily because the Internet is changing all the time. It also has a relatively narrower point of view. It is

aggressive because of its accurate and speedy infection, but the target is limited by the hit list.

The list can also go out of date. First, a previously vulnerable host may be patched and can no longer be infected or it is no longer available for various reasons, such as the connection is down, it is powered off, etc. Second, a newly configured machine may be vulnerable but not included in the list, and the worm would not try to infect it. The latter case is less significant because it just misses a few hosts. This can be solved by combining a hit-list target-finding scheme with another type of scanning scheme; e.g., a blind target-finding scheme. Therefore, the worm can first spread according to the hit list then start a random scan on the second round to cover this flaw.

It is a bigger issue in the first case. Especially, if this later-becoming-invulnerable or unavailable host is close to the root of the infection tree, then a large portion of the hit list will remain untouched and seriously degrade the performance. One way to resolve this is by cross infection, where at each node of the infection tree, the instance of the worm not only spreads to its own children, but also to its sibling's children. This means the hit-lists passed to the children have to be redundant. The other method is to pass the whole hit list to every worm instance and use the permutation technique to have the portion of the hit

list of each worm overlap. The potential problem with these two solutions is that the size of the worm will be larger and consume more bandwidth.

7.2 Flash Worm Detection and Challenges

There is no ultimate solution to flash worm detection so far. Hit-list scanning is accurate, causing very little or no failure connection, while failure connection is the most popular parameter used in current anomaly-based detection schemes. The connection rate will be high, but as the previous section stated, this method is not very accurate, and can easily trigger false alarms.

There are detection algorithms that are possible to detect flash worms, but there are difficulties with these algorithms. One type of such systems is the HoneyPot technology, such as HoneyD or the EarlyBird system. The hit-list may be generated automatically and HoneyPots appear to be another vulnerable machine on the network and will be included on the list. Any traffic coming towards HoneyPot is considered an anomaly and is most likely an attack. Flash worms may be caught using HoneyPots.

The Destination-Source Correlation (DSC) algorithm might be able to catch flash worms as well because the victims need to send out many copies of a worm to different destinations, often targeting the same port. This repetitive behavior makes the worm detectable, but that is based on the assumption that

each node (of the infection tree) tries to infect a large number of children. If the infection tree is implemented more narrowly, meaning every node splits into fewer children, and the hit-list is not overlapping too much, meaning each host only gets visited once or twice, the DSC algorithm may not be useful because the anomaly may not reach the threshold and no alarm will sound.

7.3 Simulation

In order to test the performance of the flash worm, a simulation is needed. In order to test the worm, a network environment is necessary. This section includes the design of the simulation.

7.3.1 HoneyD

HoneyD, an open source HoneyPot system, was introduced in the previous sections. It can be used in worm detection, as well as slow down and stop the spread of worms. HoneyD is able to simulate a large network address space with simple configuration files. It can simulate different operating systems on different virtual hosts. It can also model port behaviors with scripts called personalities. These personalities, after being implemented in scripts, are mapped to the simulated IP addresses and port numbers in the configuration file. The configuration file consists of three parts. The first part is the network

routing information, which specifies the connection and topology of the simulated network. The second portion is the definition of various personality profiles. The third part contains the mapping of the simulated IP addresses to the personalities.

7.3.2 Simulation of Test Environment

The personality is normally triggered by incoming service requests or connection attempts, and then the corresponding script will start to take reaction. The reaction is normally some interaction with the incoming request that is from another machine, often the attacker. The script can also interact with another personality that is also run by HoneyD.

With the network environment set up, traffic running on this network is the second consideration. Since the simulation should be as realistic as possible, background traffic (other than worm traffic) is necessary. This can be done inside HoneyD with the scripts interacting with each other. In an ordinary network, certain ports have a higher usage rate than others; e.g., 80 (HTTP), 25 (SMTP). This can be implemented with some adjustable parameters to indicate bandwidth and network usage for certain ports. Adding this background traffic to the simulated network will make the environment more realistic.

7.3.3 Simulation of Flash Worms

The flash worm can be simulated with a script and run on HoneyD. A virtual host on HoneyD can be the initiator; in this simulation, it is the root of the infection tree or the attacker. This host will serve the first level of infection, and then the worm propagates on. The simulation is to use the hit list to test the distribution speed so the simulated flash worm does not do anything to the infected machine. The payload doesn't do any harm and contains only the hit-list.

There are two approaches to designing the infection tree. It is illustrated in Figure 7, where part A is the design of a narrower and deeper tree and part B is a design with a flatter and wider tree. k is the number of immediate descendents a node has, i is the level of the tree, and T_i is the total number of items being sent from level i , which is defined as:

$$T_i = T_{i-1} - k^i$$

T_i reduces as i increase. The higher k is, the faster T_i decreases, and fewer duplicates of the hit list will be needed. As a result, a flatter and wider tree reduces the duplication of the hit-list and the initial infection is faster, but every node tries to connect more targets and creates more traffic. This makes it easier to be detected by anomaly-based detection systems. A tree that is narrower and deeper creates fewer abnormal patterns to be detected. The initial infection

needs to carry a bigger hit list, making it slower, and because more duplication is needed, it has a higher chance of being detected by an unknown signature based detection system.

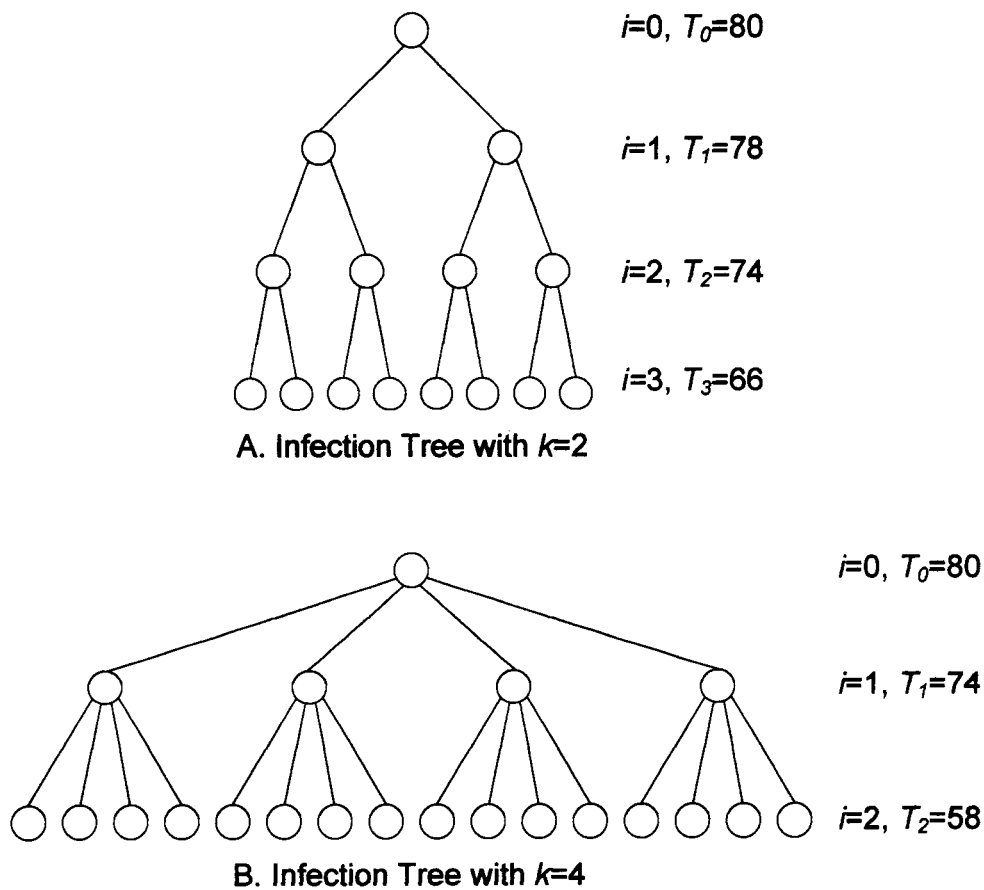


Figure 7 Design of Flash Worm Infection Tree

The simulation will use cross infection to prevent infection failures, but cross infection adds overhead to the propagation speed and bandwidth usage. Full-cross infection is too inefficient and so a half-cross approach is used. This is shown in Figure 8, where each node infects half of the children of its adjacent

siblings (shown in dotted lines). This serves the purpose of failure prevention, where an infection sub-tree will not be completely eliminated unless infection attempts of the sub-tree root and both of its adjacent siblings all fail. At the same time, the overhead is limited because every node is only visited twice.

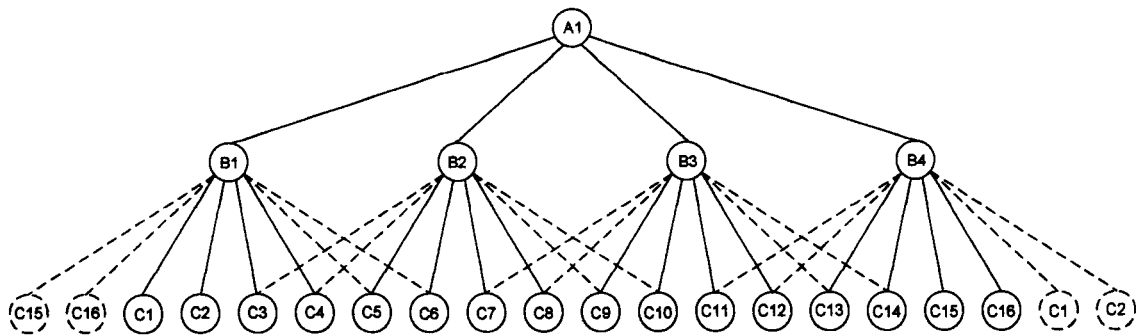


Figure 8 Illustration of Cross Infection

A personality can be defined and implemented with a script to serve a flash worm. Upon receiving incoming traffic at a certain port, which is pre-selected for the worm simulation, the script will inspect the hit list, divide it, and further send it to the next targets according to the list derived from the infection tree.

7.4 Challenges of the Simulation

One big concern with this simulation is the system resources. Many processes (one for each script and each instance of worm) will be run on the

simulated HoneyD system. The larger the simulated network and the hit-list are, the longer time it takes to process the simulation. This will degrade the simulation and make it harder to obtain a realistic result.

To solve this problem, multiple HoneyD machines can be used and connected with Ethernet. This is illustrated in Figure 9, where three HoneyD machines are used and each one of them imitates a certain portion of the simulated network to distribute the processing requirement. The simulated network is shown in dotted lines and the actual ones are shown in solid lines.

Even with this distribution approach, the result still may not be realistic. So different network behaviors are tested on these HoneyD machines and the speed of these behaviors is compared with an implementation for a regular network. For example, the time it takes for certain numbers of clients to transfer a file of a certain size from one host for ordinary Ethernet can be compared with the same scenario on HoneyD.

The processing power and system specification should be taken into account in this comparison. A scale can be defined and the result from the HoneyD simulation is mapped with the scale to find the time it takes on a real network.

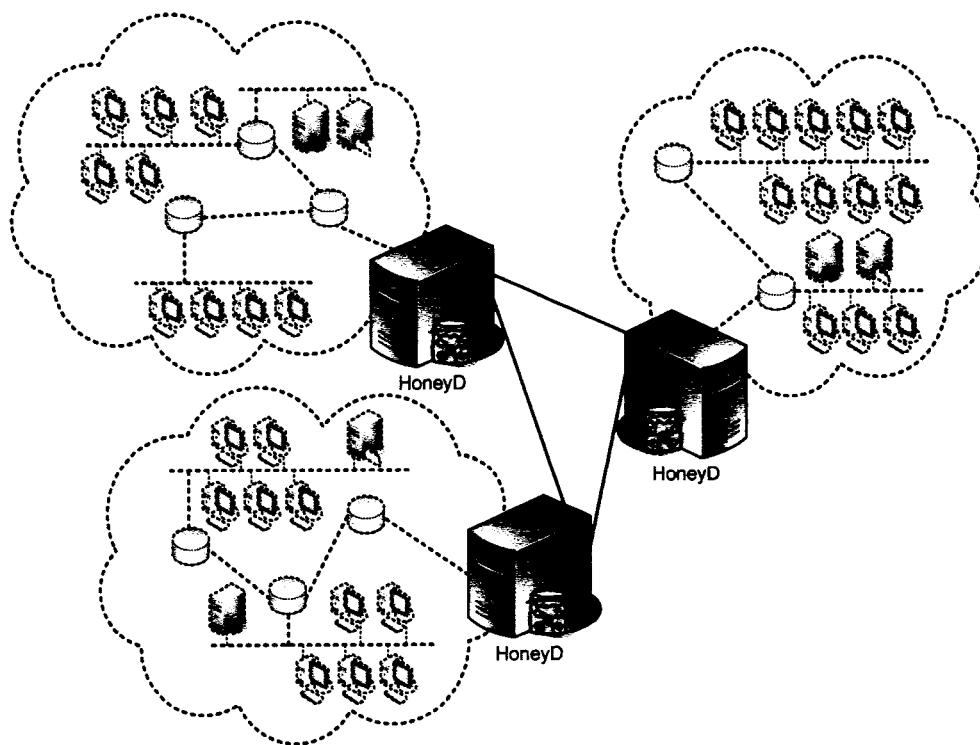


Figure 9 HoneyD Simulated Network Environment

Flash worms carry the hit list in their payloads and send the lists to the next victims. The greater the depth of the infection tree, the more copies of the list will be generated. Since the worms are running as processes and the hit list is stored in the memory, memory can run out before the bottom of the infection tree is reached because its simulation is running on PCs.

This simulation will be an ongoing project, and a team of three computer engineering graduate students will enhance HoneyD and develop this simulation according to the design.

7.5 Hierarchical Detection

With the environment and active worms running, the detection algorithm can be tested. As mentioned in the previous section, worm detection should be done in a hierarchical approach using a combination of different schemes to be more effective and have a broader view. This paper proposes such a detection algorithm using the combination of failed connection rate scheme and a hierarchical DSC scheme to detect flash worms as well as other types of worms. The detection model is illustrated in Figure 10 and the detection flow is shown in Figure 11.

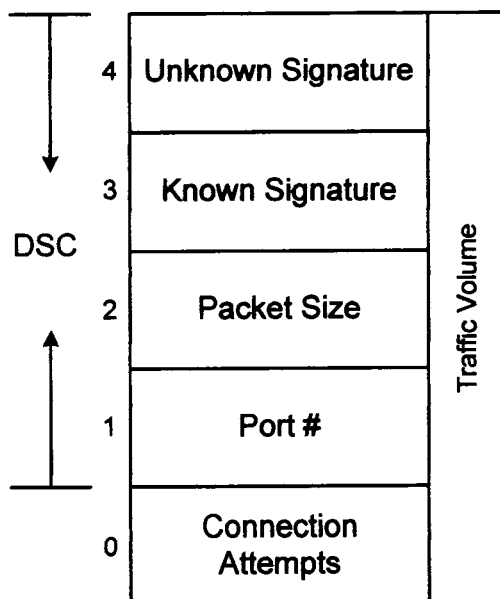


Figure 10 Hierarchical Detection Model

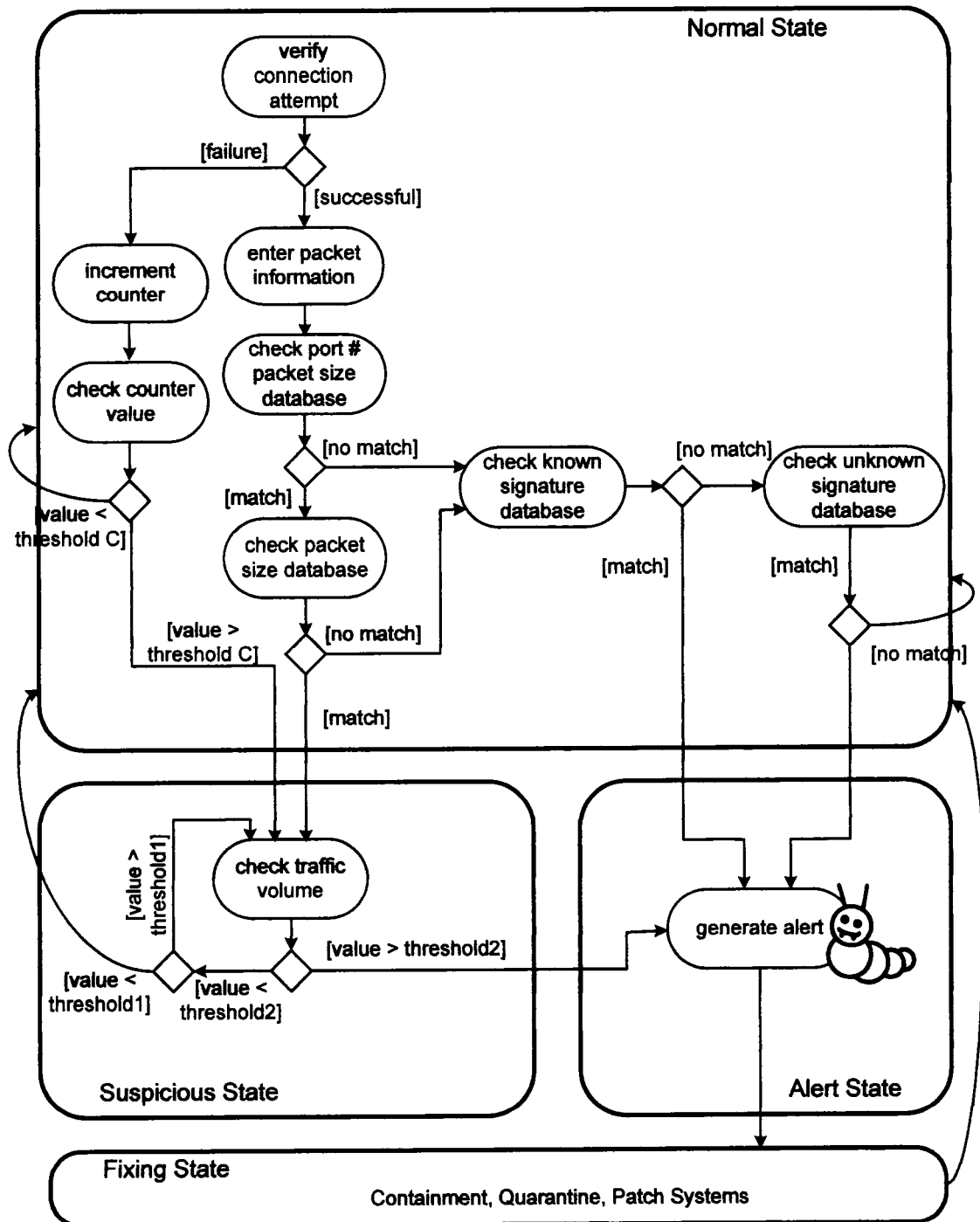


Figure 11 Hierarchical Detection Flow

The first level of the system uses a connection attempt as a detection parameter. This is designed for detecting worms using blind target-finding schemes. Based on the total number of connection attempts, the system keeps track of both ICMP-destination-unreachable and TCP RST messages. If the percentage of failed connections exceeds the redefined threshold C , then a worm possibly exists. The system will go into a suspicious state and if the traffic volume exceeds a certain threshold then it will go into an alert state. If the threshold C is not met, then the packet is ignored, and the system stays in a normal state. The successful connection attempts are passed to the DSC part of the system for further inspection.

The DSC part is mainly designed to detect flash worms and consists of four levels. The system first records the packet size and port number and puts this information into the database. The entries in the database age and are eliminated as time goes by. The first level of DSC uses port number for detection. This is also the original approach of DSC. The algorithm says that if a host gets a packet on port i then start sending packets destined for port i to other hosts, it becomes suspicious. If the condition of the first level is met, then the system will consider the second level, the size of the packets. If the condition is false, meaning the packet is not designated to the same port, it is sent to level 3 to be compared with the known signature database.

Level 2 is to check the packet size, based on the DSC scheme. The packet size is to be compared with the packet size of the other connections. This information can be found in the packet size database. The comparison has a tolerance of plus or minus four bytes. This is because when the flash worm divides a hit list with an odd number, some children will get one more item on the hit list than other children. These hit-list items are most likely to be IP addresses, and one IP address is four bytes. At this level, if the packet size is the same (with the consideration of plus and minus four bytes tolerance) as the previous receiving packet size or the same as the size of the packets from other connections, it is said to be suspicious, and the system changes state. If the packet size is different, then the packet is sent to level 3 to be compared with the known signature databases.

Level 3 uses the known signature to detect worms, which is the traditional signature-based detection method. If the content of the worm matches any item in the known signature database, a worm alert will be generated and the system enters alert state. If no items are matched, the packet is sent to level 4 for the last inspection. Level 4 consults the unknown signature database to detect worms. The signatures are generated from frequently occurring sub-strings that break down into tokens. The signatures age as time goes by to reduce the size of the database. If the packet content matches the signatures in this database, a worm

alert will be generated; if not, the traffic is considered innocent and the system stays in normal state.

When the suspicious state is reached, the system will consult with the traffic volume to determine whether to trigger the worm alert or not. If the traffic volume is higher than the predefined threshold 2 (indicates high traffic volume) an alert will be generated, and the system enters alert state. Otherwise, the system will stay in a suspicious state until the traffic volume goes down and get below threshold 1 (indicates normal traffic volume), and then goes back to a normal state. When the alert state is reached, the system will take the necessary steps for containment and quarantine, or inform the system administrators to patch the system in the fixing state (the details of the fixing state is out of the scope of this thesis). If the crisis is resolved, the system will again go back to the normal state and prepare for the next attack.

This model is able to detect more worm characteristics than other systems proposed in the past. The coverage is shown in Table 3. This model is not able to catch the hypothetical metamorphic worms because it exploits different vulnerabilities and every instance of the worm may target on different port numbers. The model might be able to catch passive scanning worms, depending on the implementation of the worm and the incoming traffic volume of the victim. If built on the Honeypot system, the model can be extended and have the

Table 3 A Comparison of Worm Characteristics with the Hybrid Hierarchical Detection Model

Method of Detection	Characteristic of Worms	Target Finding Scheme				Propagation Scheme			Transmission Scheme		Payload Format		
		Blind	Hit list	Topo-logical	Passive	Self-Carried	Second Channel	Embedded	TCP	UDP	Mono-morphic	Poly-morphic	Meta-morphic
Signature	Known Signature	-	-	-	-	✓	✓	maybe	✓	✓	✓	✗	✗
	Unknown Signature	-	-	-	-	✓	✓	maybe	✓	✓	✓	✗	✗
	Token based Signature	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✗
Anomaly Based	Destination Source Correlation	✓	✓	✓	✗	✓	-	-	✓	✓	-	-	-
	TCP SYN - Connection Count	✓	✓	✓	✗	✓	-	-	✓	maybe	-	-	-
	Failed Connection Attempts	✓	✗	✗	✗	✓	-	-	✓	maybe	-	-	-
	Ratio of Success/Failure Attempts	✓	✗	✗	✗	✓	-	-	✓	maybe	-	-	-
	Monitoring DarkNet	✓	✓	✗	✗	✓	-	-	✓	✓	-	-	-
Hybrid	Honeypots	✓	✓	maybe	✗	✓	-	-	✓	✓	-	-	-
	Modified DSC + HoneyStat	✓	✓	✓	✗	✓	-	-	✓	✓	-	-	-
	Early Blind (unknown sig+DSC)	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗	✗
The Hybrid Hierarchical Detection Model		✓	✓	✓	maybe	✓	✓	✓	✓	✓	✓	✓	✗

capability of containing worms, making the system complete (as shown in Figure 11). Containment is not in the scope of this thesis.

The system is especially designed to catch flash worms with the hierarchical DSC algorithm and the traffic volume. Even though flash worms cause very little connection failure, they still generate high traffic volume, and still follow certain patterns, such as using the same port number and having the same packet size, making it detectable by this model.

The system checks detailed packet information and is better to be used for a local area network. This system should be implemented at the gateway of an enterprises network and will have the scope of LAN.

8. Conclusion

This thesis has identified existing and hypothetical worm characteristics during the target finding and propagation phases of a worm's life cycle. They are classified based on target finding, propagation, transmission schemes, and payload format.

Current detection algorithms were organized in this thesis based on the categories of signature-based, anomaly-based or hybrid approaches. These categories have been evaluated against worm characteristics.

This research has categorized current containment schemes based on the method they use to control the spread of worms. Implementations at different network locations and the system scopes have been explored as well.

An ideal system should use combination schemes to have a more comprehensive coverage. New attacking technologies are being developed every day and the threat constantly exists. This research has pointed out the remaining challenges and future work to be done based on the insufficiency of current algorithms.

This thesis discussed flash worms, said to be the speediest hypothetical worm that uses a hit-list-target-finding scheme. The discussion contains a simulation of flash worms, including how to design the infection tree to make the hit-list efficient and prevent failure infection. HoneyD was proposed to be used

for simulation of the flash worm as well as the network environment. The network environment simulation provides traffic running in the background with adjustable parameters such as the bandwidth and port usage. This is used to test the performance of the simulated flash worm and the efficiency and effectiveness of the detection algorithm. Challenges and possible solutions of the simulation are explored in the report.

Flash worms are very hard to detect, but not impossible. This thesis proposed a hybrid hierarchical detection model. The model is intended to catch both flash worms and blind scanning worms. The system uses failure connection and advanced DSC schemes with the consideration of traffic volume. In comparison with other systems, the proposed model has a broader coverage of worm characteristics. The system is designed to be located at the gateway of a network and has the scope of LAN.

References

- Berk, V., Bakos, G., & Morris, R. (2003). Designing a framework for active worm detection on global networks [Electronic version]. *1st IEEE International Workshop on Information Assurance*, 13-23.
- Chen, S., & Ranka, S. (2004). An Internet-Worm Early Warning System [Electronic version]. *IEEE Global Telecommunications Conference 2004*, 4, 2261-2265.
- Chen, S., & Tang, Y. (2004). Slowing Down Internet Worms [Electronic version]. *The 24th IEEE International Conference on Distributed Computing Systems*.
- Chen, Z., Gao, L., & Kwiat, K. (2003). Modeling the Spread of Active Worms [Electronic version]. *IEEE Computer and Communications Societies Annual Joint Conference*. 3, 1890-1900.
- Dagon, D., Qin, X., Gu, G., Lee, W., Grizzard, J., Levine, J., & Owen, H. (2004). Honeystat: Local worm detection using honeypots [Electronic version]. *Proceedings of the 7th Symposium on Recent Advances in Intrusion Detection*.
- Dantu, R., Cangussu, J., & Yelimeli, A. (2004). Dynamic control of worm propagation [Electronic version]. *International Conference on Information Technology: Coding and Computing*. 1, 419-423.
- F-Secure Computer Virus Information Pages: Slammer. (n.d.). Retrieved May, 2005, from <http://www.f-secure.com/v-descs/mssqlm.shtml>

- F-Secure Computer Virus Information Pages: Witty. (n.d.). Retrieved May, 2005, from <http://www.f-secure.com/v-descs/witty.shtml>
- Glazer, D. G. (n.d.). Computer worms. Retrieved May, 2005, from <http://www.research.umbc.edu/~dgorin1/is432/worms.htm>
- Gu, G., Sharif, M., Qin, X., Dagon, D., Wenke, L., & Riley, G. (2004). Worm Detection, Early Warning and Response Based on Local Victim Information [Electronic version]. *20th Annual Computer Security Applications Conference*, 136-145.
- Jung, J., Paxson, V., Berger, A. W., & Balakrishnan, H. (2004). Fast Portscan Detection Using Sequential Hypothesis Testing [Electronic version]. *2004 IEEE Symposium on Security and Privacy*.
- Madhusudan, B., & Lockwood, J. (2004). Design of a system for real-time worm detection [Electronic version]. *Proceedings of the 12th IEEE Annual Symposium on High Performance Interconnects*, 77-83.
- Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., & Weaver, N. (2003). Inside the Slammer Worm [Electronic version]. *IEEE Security & Privacy Magazine*, 1 (4), 33-39.
- Moore, D., Shannon, C., Voelker, G. M., & Savage, S. (2003). Internet Quarantine: Requirements for Containing Self-Propagating Code [Electronic version]. *IEEE INFOCOM*.

- Newsome, J., Karp, B., & Song, D. (2005). Polygraph: Automatically Generating Signatures for Polymorphic Worms [Electronic version]. *2005 IEEE Symposium on Security and Privacy*, 226-241.
- Provos, N. (2004). A virtual honeypot framework. [Electronic version]. *Proceedings of the 13th USENIX Security Symposium*.
- Provos, N. (n.d.). Honeyd Virtual Honeypot. [Computer Software] retrieved August, 2005, from <http://honeyd.org>
- Qin, X., Dagon, D., Gu, G., Lee, W., Warfield, M., & Allor, P. (2004). Worm detection using local networks [Electronic version]. *Technical report, College of Computing, Georgia Tech*.
- Sasser Worm Analysis – LURHQ. (n.d.). Retrieved May, 2005, from <http://www.lurhq.com/sasser.html>
- Secunia - Virus Information - Sasser.G. (n.d.). Retrieved May, 2005, from http://secunia.com/virus_information/11515/sasser.g
- Shannon, C., & Moore, D. (2004). The Spread of the Witty Worm [Electronic version]. *IEEE Security & Privacy Magazine*, 2 (4).
- Singh, S., Estan, C., Varghese, G., & Savage, S. (2003). The EarlyBird System for Real-time Detection of Unknown Worms [Electronic version]. *Tech Report CS2003-0761, University of California, San Diego*.
- Snort. (n.d.). Retrieved May, 2005, from <http://www.snort.org>

- Spitzner, L. (2002). *Honeypot: Tracking Hackers*. Boston: Addison-Wesley.
- Staniford, S., Moore, D., Paxson, V., & Weaver, N. (2004). The Top Speed of Flash Worms [Electronic version]. *Proceedings of the 2004 ACM Workshop on Rapid Malcode*, 33-42.
- Staniford, S., Paxson, V., & Weaver, N. (2002). How to Own the Internet in Your Spare Time [Electronic version]. *Proceedings of the 11th Usenix Security Symposium*, 149-167.
- Stolfo, S. J. (2004). Worm and Attack Early Warning [Electronic version]. *IEEE Security & Privacy Magazine*, 2 (3), 73-75.
- Weaver, N., Ellis, D., Staniford, S., & Paxson, V. (2004). Worms vs. perimeters - the case for hard-LANs [Electronic version]. *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects*, 70-76.
- Weaver, N., Paxson, V., Staniford, S., & Cunningham, R. (2003). A Taxonomy of Internet Worms [Electronic version]. *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 11-18.
- Weaver, N., Staniford, S., & Paxson, V. (2004). Very fast containment of scanning worms [Electronic version]. *Proceedings of the 13th USENIX Security Symposium*, 29-44.

Wong, C., Wang, C., Song, D., Bielski, S. M, & Ganger, G. R. (2004) Dynamic quarantine of internet worms [Electronic version]. *2004 International Conference on Dependable Systems and Networks*, 73-82.

Worminator Projector. (n.d.). Retrieved April, 2005, from <http://worminator.cs.columbia.edu/public/index.jsp>

Wu, J., Vangala, S., Gao, L., & Kwiat, K. (2004). An efficient architecture and algorithm for detecting worms with various scan techniques [Electronic version]. *Network and Distributed System Security Symposium*.