2008

A Survey of Worm Detection and Containment

Pele Li, *San Jose State University*
M. Salour, *San Jose State University*
Xiao Su, *San Jose State University*

A Survey of Internet Worm Detection and Containment

Pele Li, Mehdi Salour, and Xiao Su

Computer Engineering Department

San Jose State University

One Washington Square San Jose, CA

Abstract

The self-duplicating, self-propagating malicious codes, known as computer worms, spread themselves without any human interaction and launch the most destructive attacks against computer networks. At the same time, being fully automated makes their behavior repetitious and predictable.

This paper presents a survey an d comparison of Internet worm detection and containment schemes. We first identify worm characteristics through their behavior, and then classify worm detection algorithms based on the parameters used in the algorithms. Further, we analyze and compare different detection algorithms with reference to the worm characteristics by identifying the type of worms that can and cannot be detected by these schemes.

After detecting the existence of worms, the next step is to contain them. This paper explores the current methods used to slow down or stop the spread of worms. The locations to implement the detection and containment, as well as the scope of each of these systems/methods, are also explored in depth. Finally, this paper points out the remaining challenges of worm detection and future research directions.

1. Introduction

The self-propagating malicious codes, known as computer worms, spread themselves without any human interaction and launch the most destructive attacks against computer networks. 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. Since worms involve network activities in the first two phases, their behaviors in these two phases are critical for developing detection algorithms. Therefore, this paper first focuses on worm characteristics that facilitate their detection.

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

After detecting the existence of worms, the next step is to contain the worms. This paper explores the current methods used to slow down or stop the spread of worms. The locations to implement detection and containment, as well as each of these system scopes, are also explored in depth at each level.

1.1 Overview

First, a terminology section is presented. Worm characteristics during target finding and worm transferring phases are identified in Section 2. This is followed by an overview of worm defense mechanisms in Section 3, namely, detection and containment. The classification of detection algorithms is presented in section 4, and the containment systems are presented in section 5. Depending on where the detection and containment systems are implemented, they may construct different views of worm propagation behaviors, so there may be differences in the scope of their defenses. This is discussed in section 6. We conclude the survey by identifying future research challenges in Section 7.

1.2 Terminology

Activation: Activation is when a worm starts performing its malicious activities. Activation might be triggered on a specific date, or under certain conditions

False Alarm: False Alarm is an incorrect alert generated by a worm detection system.

False Positive: False positive is a false alarm where an alert is generated when there's no actual attack or threat.

False Negative: False Negative means the detection system missed an attack. It is a false negative if no alert is generated while the system is under an attack.

Infection: Infection is the result of the worm performing its malicious activities on the host.

Target finding: Target finding is the first step in a worm's life to discover the victims (vulnerable hosts)

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

Transfer: Transfer refers to sending a copy of the worm to the target after the victim (target) is discovered

Virus: A virus is a malicious piece of code that attaches to other programs to propagate. It can't propagate by itself, and normally depends on certain user intervention such as opening up an email attachment or running an executable file to be activated [1].

Worm: A worm is malicious piece of code which self-propagates often via networks connections exploiting security flaws in computers on the network. In general, worms do not need any human intervention to propagate; however, a category of worms called passive worms require certain host behavior or human intervention to propagate. For example, a passive worm only propagates itself until it is contacted by another host.

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 malicious code that duplicates and propagates by itself. Usually, it does not require any human interactions and spreads via network connections.

The life of worm, after released, typically includes the following phases: target finding, worm transferring, worm activation, and infection. Du ring 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 on the 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 based on worms' 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.

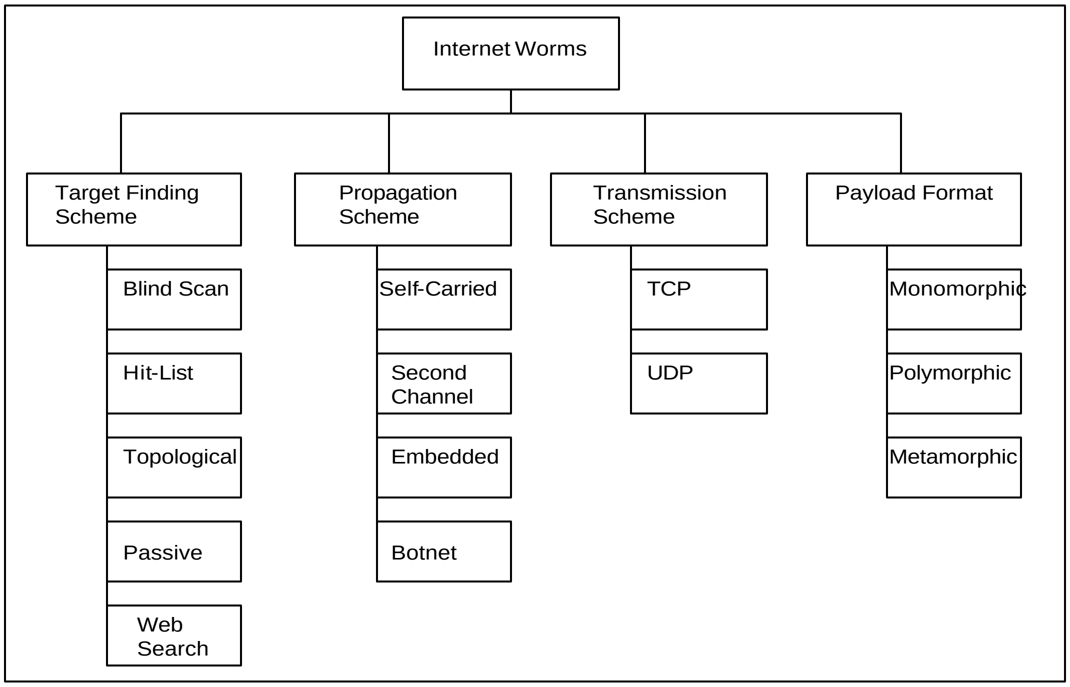

Figure 1 Categorization of Worm Characteristics

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, may spread fast, but are not very accurate. The miss rate can be very high. An improved version of 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.

Although most target scanning worms blindly scan the entire Internet IPv4 address space, an advanced scanning worm --"routing worm" -- targets a smaller scanning space without ignoring potential vulnerable hosts. A routing worm uses the information provided by BGP routing tables to narrow the scanning spectrum and target particular systems within a geographic location (e.g. specific country), an Internet Service Provider (ISP), or an autonomous system. A routing worm can spread between two to more than three times faster than traditional worms [2].

Further, there is a hypothetical category of scanning worms based on importance scanning[1]. Instead of randomly choosing targets, importance scanning worm samples targets in accordance to the underlying group distribution of the vulnerable hosts. This type of worm usually works in two stages, in the first stage random scanning or routing scanning is used to gather information about enough IP addresses to build an initial group distribution of vulnerable hosts, and then worm uses importance sampling technique to reduce the number of scans and to attack a large number of vulnerable hosts very fast [3].

With the advent and adoption of Network Address Translation (NAT) and IPv6, researchers have studied their impact on scanning worms. Upgrading to IPv6 can dramatically increase the scanning space (2^64 IP addresses for a single subnet comparing 2^32 IP addresses in the entire IPv4 space!), and as a result, virtually prevent a worm from spreading through scanning (Zou et al. calculated that for a worm with a scan rate of 100k hosts per second, it would take 40 years to infect 500k vulnerable hosts on a single IPv6 subnet!) [2].

In addition, Rajab et al. showed that NAT affects and limits the worm propagation in three ways by first reducing the number of hosts that are globally reachable. Second, if a host is compromised inside the private address space, NAT affects how efficiently the host can discover other vulnerable hosts outside of the private address space. Depending on the scanning technique used by the worm, NAT sometimes limits the worm scan to only the private IP address space.

[1] Importance scanning is based on importance sampling in statistics, which is used to reduce the sample size for accurately estimating the probability of rare events.

Third, NAT can be a major obstacle for multi-stage worms where the shell code of the worm on the infected machine needs to download its payload from another victim using a file transfer protocol such as TFTP while the IP of the other victim will not be globally accessible [4].

The second 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 externally 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. Being pre-generated, a 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 pure blind scan. Because of the high accuracy, the worm will cause very low anomaly on the network and, therefore, be hard to detect with conventional anomaly-based NIDSes.

Staniford et al. [5] had 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 longer than with 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. also simulated this type of worm [6]. 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 flash worm can cause the same damage in 1.3 second.

Many hosts on the Internet store information about other hosts on the network (e.g. /etc/hosts) and possibly reveal their vulnerabilities. Topological worms 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 as each infected host needs to contact few other hosts and does not need to scan the entire network. Further, topological worms can spread very fast especially on networks with highly connected applications [7].

If a worm does not aggressively seek the target but patiently waits, it is said to take the passive approach. Instead of voluntarily scanning the network, it waits for potential victims to approach the machine where the worm resides and then replies with a copy of the worm. It may also wait for certain user actions to find the next victim. For example, Gnuman is a passive worm which acts as a Gnutella node waiting for queries to copy itself, and CRClean is a passive worm waiting for an attack from Code Red II to counter it by installing itself on the attacker's machine [7]. This method is slow, but it is very stealthy and hard to detect.

To avoid being caught by traditional scanning worm detection techniques, a new category of worms has recently emerged which uses popular search engines such as Google and Yahoo to find vulnerable targets. Such worms use carefully crafted queries to find vulnerable hosts on the Internet. As an example, a new worm called "Santy" uses Google to search for web servers which contain the string "viewtopic.php" to exploit a vulnerability in phpBB2 and infect the web server [8].

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 for worm propagation. In [7], three propagation schemes are mentioned: Self Carried, Second Channel, and Embedded scheme.

In Self Carried worms, propagation is straight forward; the worm payload is transferred in a packet by itself. Other worms are delivered through a second channel, that is, after finding the target, the worm first goes into the target, and then downloads the worm's payload from the Internet or a previously infected machine, through a backdoor, which has been installed 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. Contagion strategy is an example of a worm that uses embedded propagation [7].

In addition to the three propagation schemes discussed, botnets have been utilized to propagate worms, spams, spyware, and launching Distributed Denial-of-Service (DDoS) attacks [9]. A botnet is a group of compromised hosts under the control of a botmaster. The communication channel for the botmaster to issue commands can be implemented using different protocols such as http or P2P protocols; however, majority of botnets use the Internet Relay Chat (IRC) protocol for this purpose [10]. Witty is an example worm propagated by botnets. Witty infected 12000 vulnerable hosts in 45 minutes, and when the machine that launched the initial attack was discovered, it was not subject to the vulnerability exploited by the witty worm, instead a different vulnerability was used to take over the control of the machine by a botmaster to launch the attack [9].

Full treatment and defense against botnets themselves are outside the scope of this paper. However, some of the methods that we discuss in section 6.2

such as combining the intelligence of the control and data plane by Zhang et al. [11] can be also used to combat botnets.

2.3 Worm Transmission Schemes

Based on how worms are transmitted, 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 transmission. 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 an additional round-trip time and the two 40 bytes 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's no wait time required like TCP worms, UDP worms normally spread very rapidly, and their speed is only limited by network bandwidth. UDP worms often have to compete over each other for the network resources [12].

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 variable size by padding the payload with garbage data, but the signature won't 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 payload differently and re-assemble the pieces at the target. This type of worm is also classified as monomorphic worm. The term polymorphic worm used here describes those worms that change their payload dynamically by scrambling the program. So every instance of the worm looks different but functions exactly the same way. With changing appearances 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[13].

2.5 Existing Internet Worms

In this section, we look at one of the first Internet worms, Morris worm, which gained extensive media coverage, then discuss five of more recent Internet worms, Code Red, Nimda, Sasser, Slammer, and Witty based on their characteristics.

2.5.1 Morris worm

Morris worm was one of the first Internet worms whose devastating effect gained the wide attention of the media. Morris worm was launched in November 1988 by Robert Tappan Morris who was a student at Cornell University at the time. It is the first known worm to exploit the buffer overflow vulnerability. It targeted sendmail and finger services on DEC VAX and Sun 3 hosts [1].

Based on the creator's claim, Morris worm was not intended to cause any harm, but it was designed to discover the number of the hosts on the Internet. The worm was supposed to run a process on each infected host to respond to a query if the host was infected by the Morris worm or not. If the answer was 'Yes', the infected host should had been skipped, otherwise, the worm would copy itself to the host; however, a flaw in the program caused the code to copy itself multiple times to already infected machines, each time running a new process, slowing down the infected hosts to the point that they became unusable [14].

2.5.2 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! 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.

2.5.3 Nimda

Nimda was first reported in September 2001 targeting Microsoft Windows workstations as well as servers.

Nimda is an advanced multi-vector worm, which uses multiple mechanisms to spread itself including from client to client via email, from client to client via network shares, from web server to client via browsing the compromised web sites, from client to web server by active scanning for various vulnerabilities of Microsoft IIS 4.0 / 5.0, and from client to web server by scanning for the back doors installed by Code Red II and Sadmind/IIS [15],[16].

Nimda propagates itself by sending emails to anything that looks like an email address inside .htm or .html files in user's web cache folder as well as contents of user's email messages retrieved via the MAPI service. The subject line of the message is variable, and the attached binary also has some variations resulting in different MD5 checksums, but all binaries are exactly the same size (57344 bytes). In addition, Nimda scans the network to find and infect vulnerable IIS servers on TCP port 80 as well as using the UDP port 69 to download the worm to IIS via tftp [16].

Once a host infected, Nimda allows the attacker to run commands with the same privileges of the infected user as well as using the infected host as a zombie to participate in DoS attacks on third parties. In addition, high scanning rate of Nimda can result in bandwidth DoS attacks on networks with infected hosts.

Further, Nimda replaces existing binaries on the system with the Trojan horse copies and infects all web contents such as .htm, .html, and .asp on the system, so any user browsing these contents on the system via a web browser will download a copy of the worm, and in some cases, certain browsers will execute the code automatically infecting the user's host [16].

2.5.4 Slammer/Sapphire

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

Slammer worm uses UDP port 1434 to exploit a buffer overflow in MS SQL server. The code size is 376 bytes. Adding the UDP header makes the worm 404 bytes long in total [12]. It uses blind scan scheme where randomly 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 doesn't write to the disk of the infected machines, it only overloads the victim system and slows down the traffic [17].

2.5.5 Sasser

Sasser was released in April 2004 targeting systems running Microsoft Windows XP or Windows 2000 that haven't been patched for the vulnerability of Local Security Authority Subsystem Service (LSASS). 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 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 [18] and Sasser G is 58KB [19].

After 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'.

2.5.6 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 of 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 system memory. This padding doesn't change the monomorphic format of Witty. The payload contains the text "(^.^) insert witty message here (^.^)" and that's why it is named Witty. Witty randomly writes data onto the disk of infected machines [20].

It is harder to detect Witty worms than worms with fixed size packets targeting fixed destination port number because of its random characteristics. 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 [21].

Another significance of the Witty worm, as mentioned in section 2.2, is that Witty was one of the first known worms distributed using botnets [9].

We summarize the characteristics of the above worms in Table 1.

Table 1 Existing Internet Worm Implementation

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

*Code Red II focus 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 as well. This presents a challenge in worm research.

3. Defending Against Internet Worms

Now that the nature of the 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. We categorize worm defense schemes in Figure 2 and will discuss each sub-category in the remaining sections of this paper.

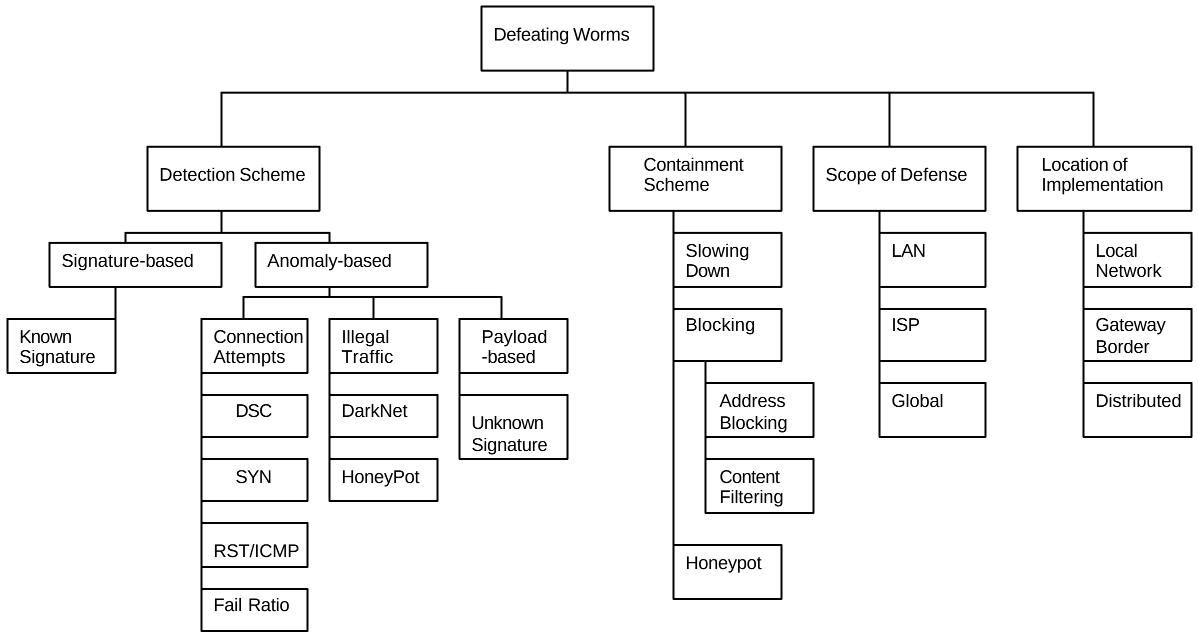

Figure 2 Categorization of Internet Worm Defense

4. Worm Detection

Based on the parameters used for detection, detection algorithms can be roughly divided as 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 discusses 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 systems. This type of system doesn't care how a worm finds the target, how it propagates itself, or what transmission scheme it uses. The signature-based systems take a look at the payload and identify 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 the 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 and making the IDS vulnerable to DoS attacks. Some of the IDS evasion tools use this vulnerability and flood the signature-based IDS systems with too many packets to the point that the IDS can not keep up with the traffic, thus making the IDS time out and miss packets, and as a result, possibly miss attacks [22]. Further, this type of IDS is still vulnerable against unknown attacks.

We believe the deficiencies of the signature-based detection method can be addressed by incorporating an anomaly-based unknown signature detection scheme with the signature-based detection in a two-tier architecture, supported by an aging and removal process to keep the size of the signature database small. The signature-based detection engine can run on a small and efficient database to look for any known threats, and at the same time the anomaly-based unknown signature detection scheme can work slower on the traffic and provide signatures for any new threat to the IDS' database. The aging process ensures removal of the old signature not seen for a long time from the database to keep the database small and the process as efficient as possible; if an old worm resurfaces again, it will be detected by the unknown signature detection engine and will be added back to the database.

4.2 Anomaly-Based

The signature of a new worm is unknown before it's 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 scan. 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 detections check the payload of the worms to generate and match signatures. Most anomaly-based detections do not care about the payload format or content; instead, they check the header of the packets to define the type of connection the packet belongs to. They observe 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.

While packet's header information is useful to detect attacks exploiting vulnerabilities of the network stack or the attacks probing hosts for vulnerable services, packet's payload information can be used to detect attacks directed at vulnerable applications. There is a category of anomaly-based detections that examines the packet payload to detect attacks directed at applications as the connection in these types of attacks is normally established (so checking the headers would not reveal the attack). An example of this anomaly-based NIDS is POSEIDON [23].

In general, the anomaly-based detection systems detect abnormal behaviors and generate alarms. This technique often requires the definition of the normal network behavior, which depends upon 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 an anomaly as normal or trigger false alarms. While this method is generally the best approach to detect unknown worms, 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 performance.

As seen in Figure 2, anomaly-based algorithms are categorized based on connection attempts, illegal traffic, or packet payload. Using connection attempts, the schemes may rely on connection count/traffic rate, failure connection rate, success/failure ratio, or destination-source correlation in their detection. Using illegal traffic, the schemes may monitor darknet and Honeypots. Using packet's payload, the schemes try to measure anomaly by comparing the packet payloads to a reference model made during the normal traffic (training period). The following sub-sections will explained these categories in details.

4.2.1 Traffic Rate/Connection Count – TCP SYN

Worms send out large number 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 the 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 sending from 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[24]. 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's not very accurate, nor is it efficient. Also, if the system logs TCP SYN only, then it is useless against UDP worms. Due to inefficiencies and high number of false alarms, this method is for the most part obsolete, and we do not recommend using it.

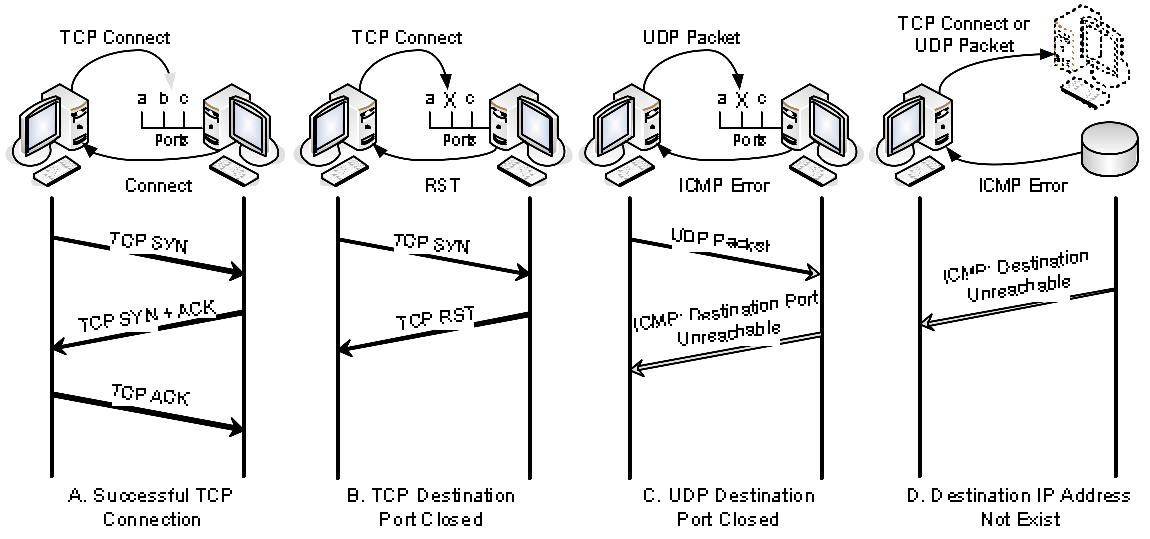
TCP Connect
Ports
Connect
TCP SYN
TCP SYN + ACK
TCP ACK
A. Successful TCP Connection
TCP Connect
RST
TCP SYN
TCP RST
B. TCP Destination Port Closed
UDP Packet
ICMP Error
UDP Packet
ICMP: Destination Port Unreachable
C. UDP Destination Port Closed
TCP Connect or UDP Packet
ICMP Error
ICMP: Destination Unreachable
D. Destination IP Address Not Exist

Figure 3 Connection Attempts

4.2.2 Connection Failure 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 the 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, if the destination host does not exist, an ICMP host unreachable packet is returned (shown in Figure 3, part D). A TCP RST packet is returned when a TCP connection targets an existing host with the destination port closed (shown in Figure 3, part B) and an ICMP destination unreachable packet error message is returned if this is a UDP connection (shown in Figure 3, part C). 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, failure rate won't be much useful because the scan and spread of worms have valid targets and won't 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. However, this scheme becomes less effective if the ICMP error messages are blocked or dropped by some border routers or gateway systems.

Berk, Bakos, and Morris [25] 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 Distributed Anti-Worm (DAW) architecture [26] 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 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. The Worm Early Warning System (WEW) proposed by Chan and Ranka [27] considered this problem. The proposed architecture utilizes gateways and hash algorithms to not only detect the error messages from failure connection attempts, but also verify whether the source address is legitimate or forged. The system only takes actions on failed connection attempts that are sent from existing source addresses.

Another issue of detection schemes based on the connection failure rate is that the worm might initiate thousands of connections before enough failures are observed. Schechter et al. proposed an algorithm based on a combination of Reverse Sequential Hypothesis Testing algorithm to monitor the connection failures and Credit-Based Connection Rate Limiting (CBCRL) algorithm to limit the rate, in which the first-contacts can be initiated by each host on the local network [28].

Further, any method relying on connection status requires resources to keep track of hosts and connection information which means such a method will not be suitable for large networks.

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 the correlation of successful and failed connections that matters. Counting the number of connections, whether they are successful or not, it depends 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 detection parameter may be less accurate. Thus, both success and failure connections should be taken into account. When the percentage of failed connection is large enough, this is said to be anomalous 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. [29] proposed Threshold Random Walk (TRW) algorithm, derived from Sequential Hypothesis Testing. The algorithm says that 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 (mark 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 conclusion and does not require training of the sy stem in advance. It focuses on detecting TCP traffic only.

Weaver et al. [30] introduced the concept and importance of the hard-LANs. They explained the three limitations of 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 to be completed or failed, it 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 successful, failure or both, often requires keeping track of each host’s and/or each connection’s information. If the network being monitored is large, this can be very resource consuming.

4.2.4 Destination-Source Correlation (DSC)

Destination-Source Correlation (DSC) algorithm is a two-phase local worm detection algorithm that aims to detect fast spreading scanning worms [31] [32]. Instead of watching for connections and the ratio of successful and failed attempts, this algorithm is based on the correlation between incoming and outgoing traffic. DSC 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 certain port (e.g., port 25 in the illustration) and then starts sending packets designated to the same port that it previous received packets, a counter is incremented. When the counter reaches a certain threshold, an alert will be issued.

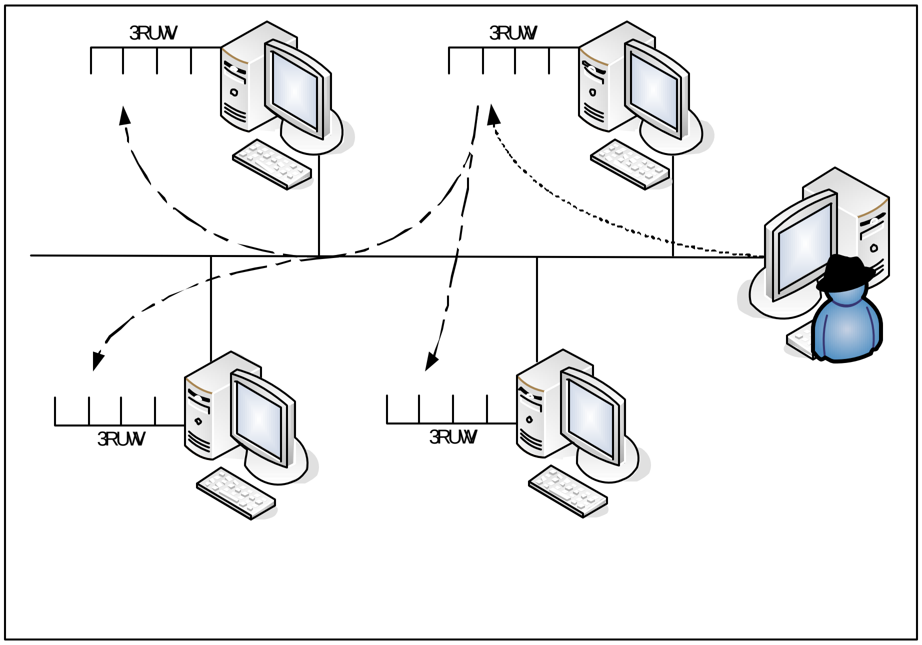

Figure 4 Illustration of Destination-Source Correlation Scheme

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 on the same port. It can detect aggressive scans including blind and topological scans. The effectiveness for 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.

The major issue of DSC is that it can only capture scans from worms targeting the same port. To address this issue, Qin et al. [32] 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 capture scans with the same port and HoneyStat can capture scans with different ports, HoneyStat can cover what DSC can't see.

In addition, the EarlyBird system combines unknown signature-based algorithm with DSC-like algorithm for worm detection [33]. An alarm is generated when packets with similar contents are sent to a number of destination

IP addresses, are received from a large number of source IP addresses, or are sent from a number of hosts to a large number of hosts (source and destination IP address pairs). Staniford et al. [6] also indicated that this type of system may be able to capture Flash worms.

4.2.5 DarkNet/Unused Address Space

Worms using blind scan generate random numbers for target addresses. There's 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 the anomaly-based detection since scanning or connection attempts toward non-exiting addresses are abnormal behaviors of regular network.

The monitored address space has to be big enough to have this method be useful. Chen, Gao, and Kwiat [34] 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. [35] 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" [35]. This means if an IP address send packet to an unused IP address, then this source IP address is considered as a victim. To prevent false alarms, they combined this definition with the Two Scan Decision Rule (TSDR), which means if the system captured two packets sending to unused IP addresses from the same host, the host is a victim. When victim count reaches a certain threshold, the system will generate a worm alert.

One of the major advantages of this method is that it requires significantly less resources comparing to the methods looking at the normal traffic in the used address space (as there should be normally no traffic to the unused address space!), and it records much less information in the IDS database.

Monitoring the unused IP address can find worms using blind target finding scheme. But again, it's not very useful against hit-list, topological or passive scans. This method works for both TCP and UDP worms since both transmission 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 [36], Spitzner defines the Honeypot as "security resource whose value lies in being probed, attacked, or compromised". Figure 5 illustrates the setup of a Honeypot where it appears as a normal vulnerable machine on the same network, just like other servers and hosts to lure attackers.

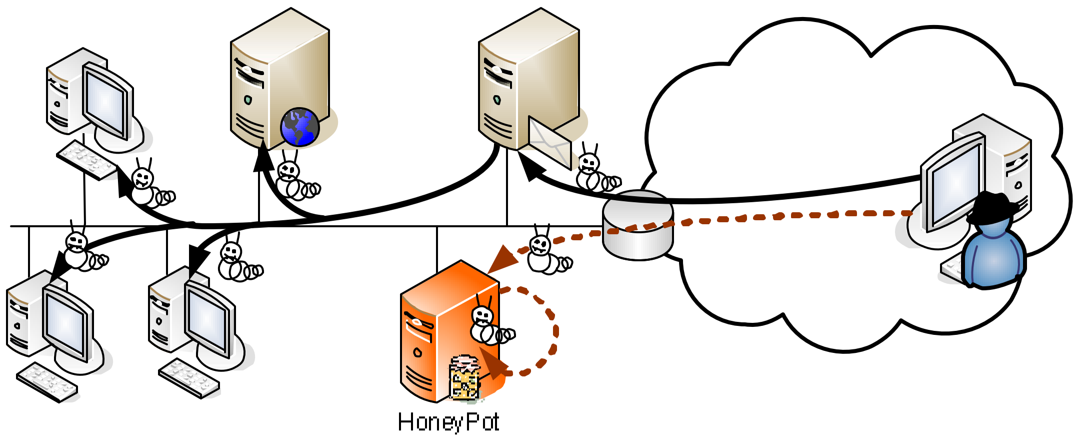

Figure 5 Honeypot Used in Worm Detection and Containment

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

Honeypot can detect blind scan worms for the same reason as the approach of monitoring unused address spaces. Honeypot is also possible to defend hit-list scanning worms. If the hit-list is generated automatically, the pre-scan may very likely include Honeypot systems because it appears as a normal vulnerable host on the Internet. Honeypot can be useful against topological worms if other working hosts on the network contain proper information of the Honeypot and let the worm finds it. But Honeypots are not useful against passive worms because they only sit and wait but don't initiate connections. As long as they are properly configured, Honeypots can detect both TCP and UDP worms.

Virtual Honeypot was used for worm detection [32] [36, 37]. In an emulator, they created a minimal Honeypot that uses virtual machines and 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, so if there are 64 virtual machines running windows and every window have 32 IP addresses, then a single node can have 2^{11} IP addresses. The hardware requirement for such a system can be as low as 32MB RAM and 770MB virtual drives to capture worms. HoneyStat generates alerts base on the correlation of three types of events: memory, disk write, and network events. An example of memory event is buffer overflow. An example of network event is a downloading of some malicious code. And an example of disk event is writing to registry keys or critical files.

HoneyD [38], a low interaction open source Honeypot daemon that supports both UNIX and Windows platforms, can detect and log connection 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 [39], 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 the one with limited configuration settings. But no matter what, Honeypot has a narrower view, since it can only see the traffic coming towards the addresses that it simulates. Honeypots can also be used for containment. It will be discussed it in session 5.3.

Honeypots are most useful when combined with other IDS methods. For example, Honeycomb project complements honeypot-based IDS system with generation of signatures to detect unknown worms. Honeycomb uses Longest Common Substring (LCS) algorithm to detect similarities and patterns in the packet payloads of the traffic seen on the honeypot [44].

4.2.7 Anomaly Detection Systems Based on Packet's Payload

The anomaly-based detection systems that we have discussed so far do not use packet's payload information. While packet's header information is useful to detect attacks exploiting vulnerabilities of the network stack or the attacks probing hosts for vulnerable services, packet's payload information can be used to detect attacks directed at vulnerable applications since the connection in these types of attacks is normally established and checking the headers would not reveal the attack [23].

Packet Header Anomaly Detection (PHAD) is a partly payload-based system, which learns the normal ranges of values for each packet header field at the data link (Ethernet), network (IP), and transport/control layers (TCP, UDP, ICMP); however, PHAD does not check the application layer protocols. PHAD examines 33 packet header fields, and its design is to be as much protocol independent as possible. PHAD, like all anomaly-based systems, checks for unusual events and uses ranking systems to decide how unusual they are. The rarer they are, the more likely they are to be hostile. PHAD uses the rate of anomalies during the training period to estimate the probability of anomalies in the network traffic. Based on this information, PHAD calculates a score for each packet header inversely proportional to the probability of being anomalous. At the end, the scores for all 33 packet header fields are added up to calculate the final score of the packet to help decide if the packet should be considered as being anomalous [40].

PAYL is a fully payload-based system based on modeling normal payload that are expected to be delivered to the network specific to the site that PAYL is installed. During the training period, PAYL creates a profile based on the traffic to each service during normal operation, and it produces a byte frequency distribution as a model for normal payloads. Based on this information, a centroid model is created prior to the anomaly detection phase for each service. In the anomaly detection phase, the distance of each packet payload from the centroid model is calculated, and if the payload is too distant from the normal payload, it will be considered as anomalous. The main difference of PAYL in comparison to PHAD is that PAYL looks at the whole payload rather than looking at the 33 packet header fields, so it can also detect application level anomalies. Further, PAYL clusters the centroids to increase accuracy and dramatically reducing the resource consumption [41].

POSEIDON is one of the most recent anomaly detection systems based on packet's payload. POSEIDON has a two-tier architecture, including a Self-Organizing Map (SOM) as a pre-processor to classify the payload data, and a slightly modified PAYL system. SOM is a topology-preserving single-layer map, which preserves the neighborhood relation between nodes during the classification. It requires some parameters on the start-up including total number of nodes on the network. The modification in PAYL system used in POSEIDON is the pre-processing of packets by SOM. Damiano Bolzoni et al. [23] reports that POSEIDON has higher detection rate, lower number of false positives, and higher runtime efficiency, when compared with PAYL and PHAD [23].

4.2.7.1 Detecting Polymorphic Worms Based on Unknown Signature Detection Systems

In section 4.1, we mentioned that one of the limitations of the signature-based detection systems is the vulnerability against unknown attacks. To remedy this issue, some algorithms have been proposed to detect unknown attacks by generating signatures in real-time. These algorithms are considered as anomaly-based as they generate the signatures based on what they detect to be a worm when analyzing the network traffic rather than using existing signatures to detect worms. As discussed in the previous section, PAYL is a good example of such a system capable of automatically generating signatures for unknown worms [42].

Madhusudan and Lockwood [43] introduced an algorithm to detect frequently-occurring strings in packets and use them as signatures to use 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 [33]. As the algorithms detect sub-strings in a worm signature, they can not only detect unknown worms sent in one packet, but also be effective against worms sent in fragments across several packets, even when worms break the payload differently each time. In addition, it can catch worms that are embedded in legitimate packets for propagation.

In section 4.2.6, we discussed honeypots and how they can be used as non-payload anomaly-based detection systems. Further, honeypots can be also used to generate signatures for unknown worms. Honeycomb is a honeypot-based IDS system, which is capable of generating signatures for unknown worms. Honeycomb deploys Longest Common Substring (LCS) algorithm to spot similarities and patterns in packet payloads of the traffic seen on the honeypot [44].

No matter whether the signature is known or unknown, most detection algorithms target monomorphic worm payloads only, and have no defense against polymorphic worms, which change the payload dynamically. Kim and Karp [45] proposed “autograph”, a distributed worm signature detection system capable of dealing with polymorphic and potentially metamorphic worms. Autograph relies on unsuccessful scans to identify suspicious source IP addresses and segregates flows by destination port. It automatically generates signatures for TCP worms by analyzing the contents of the payload based the most frequently occurring byte sequence in the suspicious flow. Autograph consists of three modules: a flow classifier, a payload-based signature generator, and tattler. Tattler is a protocol based on RTCP (RTP Control Protocol), which facilitates sharing suspicious source addresses among all monitors distributed across the network [45].

Autograph still relies on a single, contiguous substring of worm's payload of sufficient length to match the worm, and the assumption is that this single payload substring will remain invariant on every worm connection; however, a worm in theory can substantially change its payload by encoding and re-encoding itself on each connection to evade being detected by a single substring [46]. To address this problem, Newsome, Karp, and Song [46] proposed "polygraph", an algorithm to automatically generate signatures for polymorphic worms without a single payload substring. 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's 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 systems, it may be less efficient when facing known worms, but it can detect newly released worms and possibly catch other kinds of internet attack such as DDoS attack. 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 doesn't just grow bigger and bigger. This conserves the system resource from the processing time of comparing signatures and the storage space of the database.

4.2.8 Detecting Search Worms

The techniques that are used to detect scanning worms (e.g. TCP/SYN, connection failure/success rate) do not work for search worms. Also, signature-based systems are not suited to detect search worms as different queries can produce the same result. Provos et al. has proposed a solution based on the "Polygraph" framework, which is not dependent on the search queries, but instead looks for the search results. If it finds a particular query returns too many vulnerable hosts (which are tagged during indexing), it removes the vulnerable results from the return list, hence stopping the spread of the worm [8].

4.3 Limitations, Benefits, and Combination Usage of Detection Schemes

So far in this section, we have discussed various algorithms in worm detection. These systems can be classified as signature-based or anomaly-based, and are further organized into several sub-categories based on the algorithms. Different detection schemes are useful against different worm characteristics. This is summarized in Table 2.

An unknown signature-based detection system may defend 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 and efficient, and 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 single 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 anomaly, and have Honeypot to aid the detection and the gathering of worm information.

As shown in Table 2, no single algorithm provides complete protection against worms with different characteristics. Most anomaly-based systems focus on detecting blind scan worms, which is by far the most commonly seen technique for active 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 harder 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 inspect them individually with the consideration of the other pieces. Passive scan and embedded payload are often used together in worm implementations. Doing so will make the worm very stealthy, but its spread is slow and results in less damage, making it less of a threat.

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

(NASR) a solution based on the concept of frequently changing the IP addresses of the nodes on the network to neutralize hit-list worms; however, this method has many limitations and faces issues including dealing with hosts with static IP addresses or entries in DNS [47]. This is an area for future research.

Only token-based signature detection is able to detect truly polymorphic worms. If the worm fragments itself differently every time it attacks, only systems that are able to handle partial signatures can catch them.

As for the metamorphic worms, there are limited solutions available. M. Chouchane and Lakhotia proposed the “engine signature” approach to detect metamorphic worms based on a scoring system that would measure how likely the code (worm) might have been generated by a known instruction-substituting metamorphic engine [48].

There have been a lot of work done in worm detection, but there are still more challenges to face in this field.

Table 2 Anomaly Detection Methods vs. Worms Characteristic

	Characteristic of Worms / Method of Detection	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	—	—	—	—	**v**	**v**	maybe	**v**	**v**	**v**	x	x
	Token based Signature	—	—	—	—	**v**	**v**	**v**	**v**	**v**	**v**	**v**	x
Anomaly Based	Destination Source Correlation	**v**	**v**	**v**	x	**v**	—	—	**v**	**v**	—	—	—
	TCP SYN - Connection Count	**v**	**v**	**v**	x	**v**	—	—	**v**	maybe	—	—	—
	Failed Connection Attempts	**v**	x	x	x	**v**	—	—	**v**	maybe	—	—	—
	Ratio of Success/Failure Attempts	**v**	x	x	x	**v**	—	—	**v**	maybe	—	—	—
	Monitoring DarkNet	**v**	x	x	x	**v**	—	—	**v**	**v**	—	—	—
	Honeypot	**v**	**v**	maybe	x	**v**	—	—	**v**	**v**	—	—	—
	Payload-based (Unknown Signature)	—	—	—	—	**v**	**v**	maybe	**v**	**v**	**v**	**v**	maybe
Hybrid	Modified DSC + HoneyStat [21]	**v**	**v**	**v**	x	**v**	—	—	**v**	**v**	—	—	—
	Early Bird (unknown sig+DSC)	**v**	**v**	**v**	x	**v**	**v**	**v**	**v**	**v**	**v**	x	x

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 worm 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's 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. We need containment systems to eliminate the worms. Many containment methods were proposed in the past few years. These methods are summarized in Figure 2 and are classified into three categories: 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 [49], and using rate limiting techniques at different network level to slow down the infection [50].

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 content, and the other is to block traffic to and/or from certain addresses. Moore el al. [51] had simulated containment with both methods and the result shows that 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 had been proposed for address block-based containment. One system mentioned in Section 4.2.3, which uses the success/failure connection ratio for detection, is also designed to block the address when the miss and hit ratio is greater than 10 or other pre-defined value [30]. The DAW architecture [26] mentioned before also implements address blocking. If certain hosts persistently keep a 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, launch denial-of-service attacks, and bring damage to the organization. In addition, Brumley et al. analysis showed that the effectiveness of address blocking is dependent on short reaction time in putting infected hosts on the blacklist. This can especially pose severe challenges to defend against fast propagating worms [52].

5.2.2 Content Blocking

Content blocking is used in most signature-based detection systems. If packet 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 message might not be let through to avoid giving information to a scan source.

In the containment algorithm proposed by Weaver et al. [53], 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’s 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

Honeypot was originally designed to lure the attackers as a non-existing host that appears to be valid. Following this spirit, Honeypot can be used as a decoy 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 Honeypot can be used to contain Internet worms. The top part of the illustration shows that worms infect one machine then propagate and infect more hosts on the network (shown in solid lines). The bottom part of the figure illustrates the case when worms infect a Honeypot machine. The worm stays there without infecting more hosts (shown in dotted lines). This is because the targets that the worms have found and try to infect are all simulated by the Honeypot, which the worms are trapped inside.

In [39], Provos noted that HoneyD not only can be used in worm detection, but also it can be configured to appear as a host having vulnerable applications to decoy and control the spread of worms. HoneyD can emulate large size of 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 Honeypot, thus slowing down the infection of real hosts on the network. These Honeypots should be installed and activated as soon as a worm is detected. The earlier this activation is, the less damage the worm will cause. In the research reported in [39], 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 deployment in different locations of the Internet, which may have different views of network traffic. A system installed at a LAN gateway has the scope of the 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. Further, 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 the 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 as well as several based on the payload information. For example, border routers check the headers of every packet for routing purpose. 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 regular hosts or servers.

A detection algorithm implemented at the border router is good for defending pure random scanning worms and is less likely to catch worms that employ 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 of slowing down Internet worms using rate limiting, Wong et al. [50] tested rate limiting at individual host, 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. But it's more complex to install rate limiting filter at every single host than only at the backbone routers. If it's 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.

While detection at the host level does not provide many of the benefits of detection at the network border level such as visibility on all packets going in and out of the network, it provides its own unique benefits and advantages. These advantages include: 1) verification of an attack by checking if the attack or exploit was successful (less prone to false positives when compared to network border level detection) 2) ability to monitor system specific activities such as adding or removing users, root/admin privileges, and the system logs 3) ability to monitor changes, such as file size or disk space usage, to specific key components of the system such as libraries, DLLs, and executable files. 4) ability to monitor and inspect application-specific logs and information not normally available at the network level detection systems [54]. However, Brumley et al. analysis showed if the local detection and containment is used alone, to be effective, there is a need for a very high deployment ratio (to slow down the worm propagation by factor of two, half of the hosts on the network need to deploy the defense) [52].

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 greater portion of the network and have a broader view and possible to stop the worms faster and more effectively. Malan and Smith [55] proposed a collaborative detection system to reduce the false positive in host-based anomaly detection systems by defining a host as behaving anomalous if it's behavior correlates too well with other networked, but independent hosts. Using this system, they were able to distinguish non-worms processes on a system from worm processes 99% of the time [55].

Stolfo [56] proposed 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 Antura sensors and the Columbia Packet Payload Anomaly Detector (PAYL) sensors [57] 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 [51], Moore et al. simulated Internet worm containment in different percentages of customer Autonomous 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 containment system and requires a wide cooperation among ISPs, which is very difficult to achieve.

6.2 Scope of Defense

The different location of implementation gives different scope of worm detection and containment. Figure 6 illustrates the correlation of these different network 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, w hich has a central management when it comes to making decision on the type of IDS to implement.

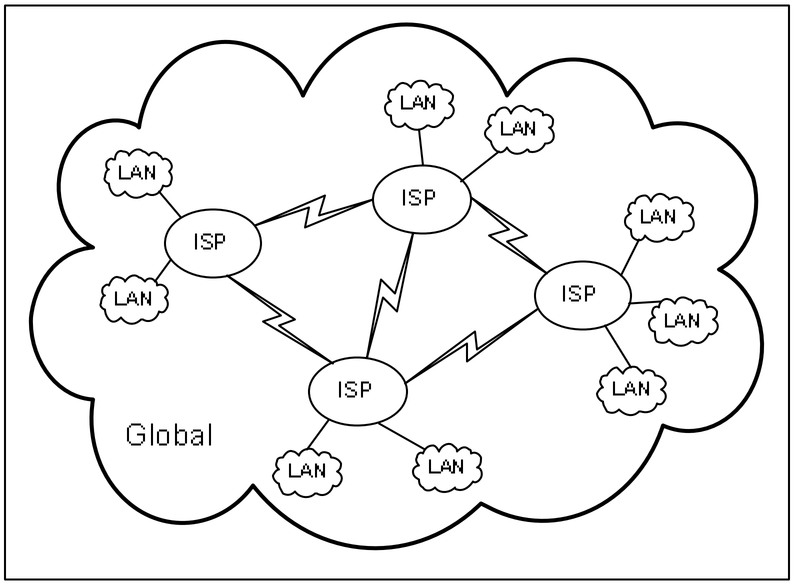

Figure 6 Different Scopes of Detection and Containment

A higher extent, that's well defined, is the scope of an ISP or Autonomous System (AS), which has multiple customer networks connecting to it. After detecting worms from certain customer network, 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 amount of traffic, so anomaly-based algorithms may be more feasible. Wagner and Plattner proposed an Entropy[2] based anomaly detection system to detect worms in fast IP networks (networks with large amount of traffic) such as the Internet backbones [58]. Essentially, the larger the coverage is, the more accurate the normal model definition would be. Distributed Anti-Worm architecture [26] is another example of a system done for

[2] Entropy is a measure on how random the traffic is and comes into perspective as worm traffic is more structured in some respects and more random from other respects when compared with the normal network traffic [58].

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. In previous sections, we see that many of the worm detection algorithms are implemented based on monitoring larger size networks (2^{20} nodes or more). Moore et al. [51] also showed that worm containment is only practical if a large fraction of the Internet unite and work together. This leads us to conclude that global scope is necessary in defending worms. Zhang et al. [11] proposed a system combining both control plane data (routing data) as well as data plane (packet headers and payloads) to detect and contain Internet worms more effectively. In this system, anomalies detected on data plane are used to identify ASes that are associated with the attacks and apply control plane filters to contain them. Furthermore, anomalies detected on the control plane (such as IP hijacking) can be used to deploy strict data plane controls on a particular ASes [11].

The idea of setting up a Center for Disease Control (CDC) for global scope detection was brought up by Staniford, Paxson, and Weaver [5]. They believe that the CDC should have the following roles: identifying outbreaks, rapidly analyzing pathogens, fighting infections, anticipating new vectors, proactively devising detectors for new vectors, and resisting future threats. This center should be deployed across the globe. There are real benefits for this approach when "one's allies are awake and working while one sleeps".

Qin et al. [32] 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, a 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 [25] discussed in Section 4.2.2 is another system that tries to obtain global scope. This global-scale worm detection and analysis system is based on ICMP destination unreachable error messages forward 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 resource for processing. The global scope is essential, yet very hard to achieve mainly because it requires wide cooperation among ISPs, organizations or countries and there is also the privacy issues. In addition, these entities might have conflicts in their interests which can make the cooperation very difficult.

Detection and containment have to be implemented with a hierarchical approach. Most of the enterprises already have their own security systems to protect their networks. Other local area networks should do the same to defend attacks at the lowest level even going as far as having detection mechanisms on some individual hosts as discussed in section 6.1. Detections at the local area network level are more detailed when compared to the ISP level, and if these local networks can flag the ISPs when worms are found, ISPs can then confirm with this situation and start the containment procedures. 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, everybody should work together. Briesemeister and Porras [59] presented an approach to evaluate collaborative worm defense mechanisms against future, unseen, and possibly defense aware worms. In their model, Briesemeister and Porras don't assume any specific worm propagation strategy and consider all possible infection sequences and propose that studying these propagation sequences will result in understanding how the current worm defense algorithms can be improved to prevent worms with similar patterns from succeeding in the future [59].

7. Conclusion

We have identified the characteristics of existing and hypothetical worms during target finding and propagation phases of a worm's life cycle. They are classified based on target finding, propagation, transmission schemes, and the payload format. Current detection algorithms are organized based on the categories of signature-based, anomaly-based or hybrid approach. We have evaluated these categories against worm characteristics. We have classified current containment schemes based on the methods they use to control the spread of worms. We have also explored the implementations of detection and containment at different network locations and system scopes.

An ideal system should use a combination of the schemes to have a more comprehensive coverage. Different detection schemes are useful at different levels of implementation. So far, there's no ultimate solution to deal with all the existing and hypothetical worms. New attacking technologies are being developed everyday and the threat constantly exists. We pointed out the remaining challenges and future work to be done based on the analysis of current algorithms. So far, there are limited solutions on detecting passive and topological scanning worms, flash worms and metamorphic worms; nevertheless, as pointed out in a research by Kienzle and Elder [60], majority of new worms coming out everyday are not novel and are derivative in nature. As a result, by defending against yesterday's worms, we can effectively protect ourselves against most new worms, while at the same time, we also need to prepare for the threats of new novel worms that can hit us in the future[60].

References

[1] E. Spafford, "The Internet Worm Program: An Analysis," *Computer Communication Review*, 1989.

[2] D. T. C. Zou, W. Gong, S. Cai, "Routing Worm: A Fast, Selective Attack Worm Based on IP Address Information," Proceedings of 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS'05), 2005.

[3] C. J. Z. Chen, "A Self-Learning Worm Using Importance Scanning," Proceedings of ACM CCS Workshop on Rapid Malcode (WORM'05), 2005.

[4] F. M. Moheeb Abu Rajab, Andreas Terzis, "On the Impact of Dynamic Addressing on Malware Propagation," Proceedings of ACM Workshop on Rapid Malcode (WORM'06), 2006.

[5] V. P. S. Staniford, & N. Weaver, "How to Own the Internet in Your Spare Time," Proceedings of the 11th Usenix Security Symposium, 2002.

[6] D. M. S. Staniford, V. Paxson, & N. Weaver, "The Top Speed of Flash Worms," Proceedings of the 2004 ACM Workshop on Rapid Malcode, 2004.

[7] V. P. N. Weaver, S. Staniford, & R. Cunningham, "A Taxonomy of Computer Worms," Proceedings of the 2003 ACM workshop on Rapid Malcode WORM '03, 2003.

[8] J. M. Niels Provos, Ke Wang, "Search Worms," Proceedings of ACM Workshop on Rapid Malcode (WORM'06), 2006.

[9] G. P. Schaffer, "Worms and Viruses and Botnets, Oh My! Rational Responses to Emerging Internet Threats," *IEEE Security & Privacy Magazine*, vol. 4, pp. 52-58, 2006.

[10] J. Z. Moheeb Abu Rajab, Fabian Monrose, Andreas Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon," Proceedings of 6th ACM SIGCOMM on Internet measurement IMC '06, 2006.

[11] E. C. Ying Zhang, and Z. Morley Mao, "Internet-Scale Malware Mitigation: Combining Intelligence of the Control and Data Plane," Proceedings of ACM Workshop on Rapid Malcode (WORM'06), 2006.

[12] V. P. D. Moore, S. Savage, C. Shannon, S. Staniford, & N. Weaver, "Inside the Slammer Worm," *IEEE Security & Privacy Magazine*, vol. 1, pp. 33-39, 2003.

[13] D. G. Glazer, "Computer worms," vol. May, 2005: http://www.research.umbc.edu/~dgorin1/is432/worms.htm.

[14] "Morris (computer worm)," vol. Retrieved July, 2007: http://en.wikipedia.org/wiki/Morris_worm.

[15] "F-Secure Virus Descriptions : Nimda," vol. Retrieved July, 2007: http://www.f-secure.com/v-descs/nimda.shtml, 2001.

[16] "CERT® Advisory CA-2001-26 Nimda Worm," vol. Retrieved July, 2007: http://www.cert.org/advisories/CA-2001-26.html, 2001.

[17] "F-Secure Computer Virus Information Pages: Slammer," vol. May, 2005: http://www.f-secure.com/v-descs/mssqlm.shtml.

[18] "Sasser Worm Analysis - LURHQ," vol. May, 2005: http://www.lurhq.com/sasser.html.

[19] "Secunia - Virus Information - Sasser.G," vol. May, 2005: http://secunia.com/virus_information/11515/sasser.g.

[20] "F-Secure Computer Virus Information Pages: Witty," vol. May, 2005: http://www.f-secure.com/v-descs/witty.shtml.

[21] C. S. D. Moore, "The Spread of the Witty Worm," *IEEE Security & Privacy Magazine*, vol. 2, pp. 46-50, 2004.

[22] R. A. e. al., "Snort 2.1 Intrusion Detection," 2nd ed. Rockland, MA: Syngress (Distributed by O'Reilly and Associates), 2004, pp. 490-491.

[23] S. E. D. Bolzoni, P. Hartel, "POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System," Proceedings of 4th IEEE International Workshop on Information Assurance (IWIA), 2006.

[24] "Snort," http://www.snort.org, retrieved May, 2005.

[25] G. B. V. Berk, & R. Morris, "Designing a framework for active worm detection on global networks," Proceedings of 1st IEEE International Workshop on Information Assurance (IWIA), 2003.

[26] S. C. Y. Tang, "Slowing Down Internet Worms," Proceedings of The 24th IEEE International Conference on Distributed Computing Systems, 2004.

[27] S. C. S. Ranka, "An Internet-Worm Early Warning System," Proceedings of IEEE Global Telecommunications Conference, 2004.

[28] S. S. J. Jung, A. Berger, "Fast Detection of Scanning Worm Infections," Proceedings of The Seventh International Symposium on Recent Advances In Intrusion Detection (RAID), 2004.

[29] V. P. J. Jung, A. W. Berger, & H. Balakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," Proceedings of IEEE Symposium on Security and Privacy, 2004.

[30] D. E. N. Weaver, S. Staniford, & V. Paxson, "Worms vs. perimeters - the case for hard-LANs," Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects, 2004.

[31] M. S. G. Gu, X. Qin, D. Dagon, L. Wenke, & G. Riley, "Worm Detection, Early Warning and Response Based on Local Victim Information,"

Proceedings of 20th Annual Computer Security Applications Conference, 2004.

[32] D. D. X. Qin, G. Gu, W. Lee, M. Warfield, & P. Allor, "Worm detection using local networks," *Technical report, College of Computing, Georgia Tech*, 2004.

[33] C. E. S. Singh, G. Varghese, & S. Savage, "The EarlyBird System for Real-time Detection of Unknown Worms," *UCSD Tech Report CS2003-0761*, 2003.

[34] L. G. Z. Chen, & K. Kwiat, "Modeling the Spread of Active Worms," Proceedings of IEEE Computer and Communications Societies Annual Joint Conference, 2003.

[35] S. V. J. Wu, L. Gao, & K. Kwiat, "An efficient architecture and algorithm for detecting worms with various scan techniques," Proceedings of Network and Distributed System Security Symposium, 2004.

[36] L. Spitzner, *Honeypot: Tracking Hackers*. Boston: Addison-Wesley, 2002.

[37] X. Q. D. Dagon, G. Gu, W. Lee, J. Grizzard, J. Levine, & H. Owen, "Honeystat: Local Worm Detection Using Honeypots," Proceedings of the 7th Symposium on Recent Advances in Intrusion Detection (RAID), 2004.

[38] "Honeyd Virtual Honeypot," http://honeyd.org, retrieved, August 2005.

[39] N. Provos, "A virtual honeypot framework," Proceedings of the 13th USENIX Security Symposium, 2004.

[40] P. K. C. Matthew V. Mahoney, "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic," *Florida Institute of Technology Technical Report CS-2001-04*, 2001.

[41] S. J. S. Ke Wang, "Anomalous Payload-based Network Intrusion Detection," Proceedings of Recent Advances in Intrusion Detection (RAID 2004), 2004.

[42] G. C. Ke Wang, Salvatore J. Stolfo, "Anomalous Payload-based Worm Detection and Signature Generation," Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection (RAID 2005), 2005.

[43] B. M. J. Lockwood, "Design of a system for real-time worm detection," Proceedings of the 12th IEEE Annual Symposium on High Performance Interconnects, 2004.

[44] J. C. C. Kreibich, "Honeycomb - Creating Intrusion Detection Signatures Using Honeypots," Proceedings of the 2nd Workshop on Hot Topics in Networks (HotNets-II), 2003.

[45] B. K. H. Kim, "Autograph: Toward Automated, Distributed Worm Signature Detection," Proceedings of the 13th USENIX Security Symposium, 2004.

[46] B. K. J. Newsome, & D. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms," Proceedings of IEEE Symposium on Security and Privacy, 2005.

[47] P. A. S. Antonatos, E. P. Markatos, K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," Proceedings of the ACM workshop on Rapid malcode WORM '05, 2005.

[48] A. L. Mohamed R. Chouchane, "Using Engine Signature to Detect Metamorphic Malware," Proceedings of the ACM Workshop on Rapid Malcode (WORM'06), 2006.

[49] J. C. R. Dantu, & A. Yelimeli, "Dynamic control of worm propagation," Proceedings of International Conference on Information Technology: Coding and Computing, 2004.

[50] C. W. C. Wong, D. Song, S. M. Bielski, & G. R. Ganger, "Dynamic Quarantine of Internet Worms," Proceedings of International Conference on Dependable Systems and Networks, 2004.

[51] C. S. D. Moore, G. M. Voelker, & S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," Proceedings of IEEE INFOCOM, 2003.

[52] L.-H. L. David Brumley, Pongsin Poosankam, Dawn Song, "Design space and analysis of worm defense strategies," Proceedings of the ACM Symposium on Information, computer and communications security, 2006.

[53] S. S. N. Weaver, & V. Paxson, "Very fast containment of scanning worms," Proceedings of the 13th USENIX Security Symposium, 2004.

[54] B. Laing, "How To Guide-Implementing a Network Based Intrusion Detection System," Retrieved July, 2007: http://www.snort.org/docs/#deploy, 2000.

[55] D. J. M. a. M. D. Smith, "Exploiting Temporal Consistency to Reduce False Positives in Host-Based, Collaborative Detection of Worms," Proceedings of the ACM Workshop on Rapid Malcode (WORM'06), 2006.

[56] S. J. Stolfo, "Worm and Attack Early Warning," *IEEE Security & Privacy Magazine*, vol. 2, pp. 73-75, 2004.

[57] "Worminator Projector," http://worminator.cs.columbia.edu/public/index.jsp, retrieved April, 2005.

[58] B. P. A. Wagner, "Entropy Based Worm and Anomaly Detection in Fast IP Networks," Proceedings of the 14th IEEE WET ICE / STCA security workshop, 2005.

[59] P. A. P. L. Briesemeister, "Automatically deducing propagation sequences that circumvent a collaborative worm defense," Proceedings of the 25th

IEEE Conference on Performance, Computing, and Communications (IPCCC), 2006.
[60] M. E. D. Kienzle, "Recent Worms: A Survey and Trends," Proceedings of the ACM Workshop on Rapid Malcode (WORM'03), 2003.