The Future of Internet Worms

Jose Nazario, with Jeremy Anderson, Rick Wash and Chris Connelly

July 20, 2001

Crimelabs research

http://www.crimelabs.net/

{jose,jeremy,rwash,devzero}@crimelabs.net

Abstract

Network worms, simple slang terminology for automated intrusion agents, represent a persistent threat to a growing Internet in an increasingly networked world. However, their evolution has been somewhat limited, and they still rely on the same basic paradigms, which contain fundamental flaws. We analyze the basic components of a worm and apply this analysis to three worms found in the wild on the Internet. We then proceed to analyze the limiting factors of existing worm paradigms and outline new ideas which we expect to become prevalent. These new worms will prove to be more difficult to identify and eradicate. It is our intention in sharing this knowledge to stimulate the development of strategies to detect and counteract the threat of smarter network worms.

1 Disclaimer

The material contained in this paper represents a double edged sword. It is the result of ongoing research within Crimelabs to identify weaknesses in current worms and various strategies they may adopt to counteract current eradication techniques.

This information can be utilized, however, to build such a worm.

Crimelabs has chosen to release this information in the spirit of full disclosure, knowing full well that the malicious use of this information is feasible. However, if we are to develop strategies to detect and react to such worms, the defense community must give serious thought to how such a threat may evolve.

We at Crimelabs will not release or build such a worm system, even for research purposes. The risk is too great that it may be adopted for malicious use.

1.1 Copyright

2 Introduction

Autonomous intrusion agents, commonly referred to as 'worms', are fast becoming a popular method of network and system compromise. The most famous start to the history of network worms is the Morris worm, which quickly crippled a substantial portion of the 1988 Internet. Worms have been a persistent security threat on the Internet, though for most of this hostory they focused on Windows hosts. Recent variations on the worm theme have been made using electronic mail viruses, which abuse the MUA API's to propagate quickly. Other than causing a flooded network and mail server queue, they usually don't do damage to the system they're on. They rarely provide for unauthorized external control of the system or the release of sensitive information.

At the beginning of this research and writing in the late part of 2000 and early 2001, the Ramen worm was infecting Linux hosts in an automated fashion, followed quickly by worms such as the Solaris/MicroSoft IIS infecting 'sadmind' worm, the Linux worms 'l1on', 'adore' and 'cheese', and the vigilante worm 'noped'. These represent, we feel, the tip of the iceberg and a glimpse of what a future of widespread intrusion will be. What we envision will be forthcoming is a more generic and dynamically configurable worm system, far more difficult to track and remove.

This paper begins by exploring worm theory, discussing the six components of worms. Several worms found in the wild are then dissected in terms of this six component model. We then examine the limiting factors of current worm paradigms from several standpoints. Several strategies are then outlined of how worms will evolve to counteract their weaknesses and current detection strategies. These include behavioral changes as well as design and infection paradigm shifts. We conclude this paper by examining detection and defense strategies that could be adopted to couteract this evolved worm system.

We define a worm as a software component that is capable of, under its own means, infecting one computer system and using it, in an automated fashion, to infect another system. This cycle is then repeated and the population of worm infected hosts grows exponentially. A virus, by contrast, spreads rapidly to a large number of hosts but cannot do so under it's own power. It

has to spread using the assistance of another program. This includes MUA software and word processing suites.

This definition mirrors an official definition used in the aftermath of the Morris Worm. From U.S. v. Morris[1], the court defined a computer 'worm':

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

One major limitation posed by this legal definition is the degree of modification of the underlying operating system that the infecting worm can perform. The presence of 'root kits' on the system can definitely be considered an attachment to the operating system in that core system tools are modified to hide the compromised state of the machine. As shown by the 'adore' worm, operating system modification can even occur at the kernel level through the use of modules used to hide various facets of the infecting agent. By this court's definition, these worms are more akin to self-propelled viruses, then.

These two definitions will be the basis for most of our discussions of worms and how they differ from computer viruses.

We will frequently make mention of a 'worm network' and the 'node' machines of this network. A worm network is nothing more than a network of systems which have been compromised by a particular worm and may communicate with eachother through the network. A node is, therefore, any of these machines, just as a node on a network is a machine on the network.

2.1 Why Worm Based Intrusions?

Given the relative stealth of a good manual intrusion, and the noise that most worms generate, this is a very good questions to ask. Two main reasons for worms will continue to exist, however:

- *Ease.* In this area, automation cannot be beaten. While there is a somewhat significant overhead to writing the worm software, it continues to work while the developers are away. Due to its nature of propagation, growth is exponential, as well.
- *Penetration.* Due to the speed and aggressiveness of most worms, infection in some of the more difficult to penetrate networks can be achieved. This is usually through serendipity, but could, with some work, be programmed into the worm system.

These are the two main benefits of using a worm based attack model, as opposed to concerted efforts done manually. And, for the foreseeable future, they will continue to be strong reasons to consider worm based events as a high threat.

2.2 Two Possible Futures

This paper is concerned with the future of Internet worms and the structure these attacks may take. We anticipate two likely futures for Internet worms.

[1] Please see the court document 928 F.2d 504, 59 U.S.L.W. 2603 from the 1991 appeal by Morris of his sentence for the release of the worm that has come to bear his name.

The first is a simple increase in the application of existing capabilities to hide itself and evade detection on the target system. This includes the increased use of kernel modifications and rootkits, an increase in the use of encrypted network traffic, and an increase in the number of attack methods, either directly included as a payload or in 'dead drop'[2] sites. This is nothing more than an increased utilization of known techniques, and can be readily identified through intrusion detection techniques and code analysis of captured machines.

We will not be discussing this type of worm scenario much in this paper, as we feel that a new paradigm will emerge which provides greater challenges for defense, detection and along with this an increased capability for a large scale threat. It is this second future that we will describe in detail in this paper.

3 Worm Theory

We first begin by discussing, from a theoretical standpoint, the components of a worm. It is important to remember that the term 'worm' is simply a shorter term for an 'autonomous intrusion agent'. With this in mind, we are able to dissect the properties of a worm and map them to categories.

3.1 The Six Components

At the core of any worm system are six components. A worm may contain any or all of these components, usually in some combination. Most worms at this time, in early 2001, have been monolithic, meaning each copy of the worm was identical to its parent.

These components are:

- reconnaissance capabilities
- specific attack cabilities
- a command interface
- communications capabilities
- intelligence capabilities
- unused attack capabilities

While there may appear to be some overlap within these categories, it is intentional and the apparent overlap is misleading. We define these categories as such because it lends itself to the greatest flexibility in design of new worms and strategies. Furthermore, detection and prevention strategies can be developed more thoroughly when a worm or worm system's structure is broken into these components.

Here we begin to elaborate on the definitions of these components. In later sections we will discuss past and present worms, and then present how they can be combined to form more difficult to detect worm systems.

[2] A 'dead drop' site is nothing more than an unmanaged location where one party leaves something, such as a package or a payload, for another party, who will stop by sometime to pick it up. The two parties do not communicate directly, but have agreed prior to the drop to use a particular site.

3.2 Reconnaissance

We begin our elaboration of worm components by examining the reconnaissance, or information gathering, capabilities of a worm system. Simply put, this is the mechanism by which the system extends its view of the world around itself, determines information about the systems and networks around it, and identifies targets.

When an attacker performs these actions, they have at their disposal a suite of methodologies. By identifying the characteristics which define a system to be of one type, or more importantly of a vulnerability, they can identify systems which will become targets.

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

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

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

3.3 Specific Attack Capabilities

This component is one of the most prominent in a worm's phenotype. This is the methods which a node of a worm system gains entry and, if need be, escalates privilidges on another system. This includes standard remote exploits, such as buffer overflows, cgi-bin errors, or the like. This can also include techniques like Trojan Horse injections and similar methods.

One concern, and the main reason why this is a separate component of a worm system, is the system type being targeted. Attack capabilities that are limited to one platform or method rely on finding vulnerable hosts of the same type with the same vulnerability. Supporting multiple methods of compromise, or multiple platforms on which to run these attack vectors, requires a significantly larger worm.

The attack portion of the code can be further subdivided into two distinct portions: the component which runs on the infecting host, and the portion that runs on the host that is being attacked. This can be binary executable code, or some sort of interpreted script. It usually will utilize network code to attach itself to a remote host, but can, in some situations, simply append the remote attack code into another network delivery mechanism, such as a mail message or a file transfer.

3.4 Command Interface

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

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

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

The objective is quite simple, to allow for the system itself, using a master-slave node relationship, to have an extended reach or capability, or more simply to allow an intruder unfettered access to the system to manually command it. In one form or another, most worm systems have some form of a command interface. This prevents the worm system from lacking any structure, so that it may be used in a controlled fashion. Commands such as file uploads or downloads, status reports, or actions such as 'attack this target' have all been possible through this interface.

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

3.5 Communications Capabilities

Because the nodes of the worm network reside on different systems, they must have some form of communications. This allows for the transfer of information. For reconnaissance information, network vulnerability and mapping information must be distributed to nodes which can use this information in an attack. For commands, they must be able to send requests to the action nodes, to initiate a scan, an attack, or other activities.

Communications channels are usually hidden by the worm using the same techniques hackers use when they have manually compromised a machine, such as rootkits[3].

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

3.6 Intelligence Capabilities

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

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

This information can manifest itself in intangible ways, as well. For example, many Windows worms use their presence on a network chat room, such as IRC, as an intelligence mechanism. They arrive once infected, announce their location and any passphrases needed to gain entry,

[3]A rootkit is a collection of modified system programs that allows the attacker to hide and usually regain entry through a back door program.

and simply sit and wait. In this fashion, the worm network knows about its members, their location and potentially any capabilities they possess.

3.7 Unused Attack Capabilities

This is both one of the more difficult to distinguish components of a worm system as well as one that lends itself to system flexibility the most. By maintaining a set of capabilities, usually for an attack, the worm is able to adapt itself to new targets. Furthermore, this reduces the amount of payload that must be carried, allowing for a leaner worm base.

Currently all known worms carry with them their collection of exploits, including ones that are not used.

Other, non-attack capabilities fit into this category as a 'catch all'. These would include the Distributed.net RC5-64 challenge client in the "Win32.Bymer worm" which affects the Windows operating system[4].

3.8 Assembling the Pieces

Having outlined the six components of a worm as above, it should now be shown how they fit together to form an active worm system. Already it can be seen that this breakdown of a worm's components is useful for dissecting behavior, as well as developing more powerful worms. We will elaborate upon these ideas later in the paper when we discuss considerations for future worm development.

4 Analysis of Three Worms

Now we apply this strategy to three network worms which have been developed in the wild. These exemplify various historical worm strategies and the real world use of the above components.

4.1 Ramen Worm Analysis

Using the above described worm structure, we can map the components of the Ramen worm (late 2000, early 2001) and characterize this instance. Max Vision has written an excellent dissection of the Ramen worm[5], along with a life cycle, which should also be studied.

Ramen is a monolithic worm, which is to say that each instance of an infected host has the same files placed on it and the same capabilities. There exists some flexibility by using 3 different attack possibilities, and compiling the tools on both RedHat Linux versions 6.2 and 7.0, but each set of files (obtained as the tar package 'ramen.tgz') is carried with each instance of the worm.

The reconnaissance portion of the Ramen worm is a simple set of scanners for the vulnerabilities known to the system (below). Ramen combines TCP SYN scanning with banner analysis to determine the potential for infection of the target host. A small random class B (/16) network generator is used to determine what networks to scan.

[4]See http://project.honeynet.org/papers/worm/ for an analysis of this worm.

[5]Please see http://www.whitehats.com/library/worms/ramen/index.html.

The specific attacks known to Ramen are threefold: FTPd string format exploits against wu-ftpd 2.6.0 [6], RPC.statd Linux unformatted strings exploits[7], and LPR string format attacks[8].

The command interface of the Ramen worm is limited. No rootshell is left listening, and no modified login daemon is left, either. The minimal command interface is reduced to the small server 'asp', which listens on port 27374/TCP and dumps the tarball 'ramen.tgz' upon connection.

Communications channels are all TCP based, including the use of the text based web browser 'lynx' issuing a 'GET' command to the Ramen asp server on port 27374/TCP, the mail command to update the database, and the various attacks which all utilize TCP based services for attack. Aside from DNS lookups, no UDP communications channels are used. No other IP protocols, including ICMP, are directly used by the worm system. All communications between the child machine and the parent (the newly infected machine and the attacking machine), along with the mail communications to servers at hotmail.com and yahoo.com, are all fully connected socket based communications.

The intelligence database of the system is updated using electronic mail messages from the system once it is infected to two central email addresses[9]. The email contains the phrase 'Eat Your Ramen!' with the subject as the network address of the infected system. The mail spool of the two accounts is therefore the intelligence database of infected machines.

Unused capabilities can be summarized as the other two exploits not used to gain entry into the system, which allow for some flexibility in targeting either RedHat 6.2 or 7.0 default installations.

A brief comment on the complexity of the Ramen worm: the author has cobbled together several well known exploits and worm components, as well as methods, utilizing only a few novel small binaries. Examination of the shell scripting techniques used show low programming skills and a lack of efficiency in design.

These findings have two ramifications. First, it shows how easy it is to put together an effective worm with minimal coding or networking skills. Simply put, this is certainly within the realm of a garden variety 'script kiddy' and will be a persistent problem for the foreseeable future. Secondly, it leaves, aside from any possible ownership or usage of the yahoo.com and hotmail.com email accounts, very little hard evidence to backtrack to any author of this worm.

4.2 PrettyPark Windows Worm

This worm illustrates two interesting facets of emerging worms on the Windows platform: the abuse of the highly integrated Windows and Office components, and the remote control of the system via a simple interface. In the strictest of senses PrettyPark is not a worm, as it is not able to spread under its own volition. Aside from lacking an independent transport layer, it exhibits all of the other traits of a worm.

PrettyPark is, like most other worms found in the wild, monolithic. Each instance carries with it a compressed copy of the main program, carried to the next instance by electronic mail (and is delivered as the MIME attachment 'PrettyPark.exe', a trojan which masks itself as a diversion based on the 'South Park' television show).

[6] CVE-2000-0573, BID 1387

[7] CVE-2000-0666, BID 1480

[8] no entry in CVE, BID 1712

[9] gb31337@yahoo.com and gb31337@hotmail.com

The PrettyPark worm performs a basic reconnaissance function by scanning the address book of the infected machine for potential victims to contact. This is done through the MAPI functionality built into some popular electronic mail clients.

Attacks carried out by PrettyPark are somewhat basic and of limited destructive power to the machine infected. Aside from infection routines, the program can scour for passwords, phone numbers and perform file manipulations, such as creation, deletion and transport. All of this can be communicated to a central site via the IRC command interface.

The command interface to PrettyPark is a simple IRC client, which is capable of receiving commands. These can include file operations, such as send and receive, as well as destructive commands like file deletions.

The intelligence database for machines infected with PrettyPark is a minimalist setup. Infected machines join an IRC network on a private, passphrase protected channel. Their presence not only gives a command interface and serves as a communication mechanism, but notifies a central point when nodes are added. It does not appear that other nodes know about eachother.

The major modes of communication for Prettypark are twofold: the IRC channel for a command interface and intelligence updates, and the use of electronic mail as an infection vector.

Due to its small size and simple nature, PrettyPark appears to have no unutilized capabilities in its normal attack vector. This limits its range and its adaptability.

Examination of the PrettyPark code shows a reasonable degree of programming savvy. While most electronic mail worms in the past two years have focused on using the Visual Basic Script language, PrettyPark is a compiled binary which was written in the Borland Delphi language. Secondly, it works at a more reasonable pace than its cousins Melissa and ILOVEYOU, spreading only after 30 minutes rather than as quickly as possible. In this way detection can be made more difficult and would require some more dedicated traffic analysis.

Theoretically, PrettyPark could be updated via its IRC interface and an upload of a new executable. This could allow for a continued presence after widespread dissemination of removal routines, such as those from anti-virus companies.

The use of an IRC channel as a command interface is a step forward in the evolution of worms. Furthermore, it is one of the more sophisticated worms to make use of the increasingly interconnected and networked desktop system software, and most definitely a sign of things to come.

4.3 The Morris Worm

We next move on to an analysis of one of the most damaging worms, as well as historically significant automated intrusions systems, the Morris Worm, which was active on the 1988 Internet.

The Morris Worm was also a monolithic worm, and each node carried with it a full payload. Of particular note was the distribution of itself in source code form and calling the system's compiler to build the new binary.

The reconnaissance methods used by the Morris Worm are themselves impressive. Utilizing a combination of scanning and trusted host analysis of the machine upon which it found itself, the worm was able to spread rapidly. By finding vulnerable Sendmail servers and finger daemons, the worm exploited programming errors and delivered its payload to the next system. Furthermore, the worm would look for Berkeley r-command indications of a trusted host relationship, as well as a user's .forward file (used in electronic mail forwarding on UNIX systems) to find new vulnerable hosts.

The attacks used by the Morris Worm were also elegant. The Sendmail attack[10] worked by throwing the server into DEBUG mode and sending a maliciously formatted command which would be processed by the system's shell (/bin/sh). The finger daemon exploit worked by exploiting a buffer overflow on the VAX architecture in the BSD code. Dictionary and username information attacks on passwords were also carried out using custom, high speed routines. This information was then used to compromise additional accounts on the networked systems.

The intelligence database of infected machines was quite simple. After a successful infection, the newly acquired node would send a one byte packet to 128.32.137.13 (which resides in the University of California at Berkeley network). Presumably if this station is listening, the source could be added to the list of known infected hosts.

What is absent from the Morris Worm is any true command interface. The worm only moves along infecting other hosts. Although a root shell is created, no backdoor of access is left. All of the worm's operations are carried out in binary programs, which do, among other things, process hiding techniques.

Communications channels are also minimal in the Morris Worm. Updates to the database of associated machines are done via a one byte IP packet (see above), but after an attack, the child node doesn't communicate with the parent node.

Again, like we saw with Ramen, the unused attack code was what was left over and not used. For example, if the attacked system was a VAX, the Sun3 specific attacks and code remained unused. Additional capabilities that may not have been used included password cracking, had there been some means of sending passwords to crack to the machines.

A review of the code of the Morris Worm reveals a sophisticated approach to worm engineering. Again, three different attack methods are used, which proved devastating in a more homogenous Internet. Furthermore, the trust relationships present in the 1988 Internet were extensive, and their abuse allowed for efficient spread of the worm in less than a week.

It is both frustrating and interesting to note that a good number of the basic problems abused by the Morris Worm persist in one form or another. Trust relationships are still significantly based on implicit hopes that the situation hasn't changed and intentions still manifest themselves as realities. Poorly engineered code still contains a multitude of unchecked input calls, allowing for buffer overflows or other input handling mistakes such as the poor string formatting exploits which Ramen used. And lastly, despite the growth of a cottage industry devoted to strong authentication mechanisms, passwords, typically poorly chosen, remain the dominant scheme used for authentication and authorization.

The code quality, even after a cursory examination, shows proper coding style, including error handling and even an efficient DES password cracking program. At the time the exploits used were known but the code, and wrappers, were developed specifically for this worm.

5 Problem Characteristic of Current Worms

We outline here several of the existing problems that will make the continued detection of network worms possible, and prevention achievable as well. Coupled to the specific nature of individual worms is the structure of typical worm networks, their behavior, and the consequences.

[10] CVE-1999-0095

5.1 Limited Capabilities

This is, by far, one of the greatest limitations of the continued success of the spread of any current worm network. These limited capabilities are used to identify the worms, giving them a particular network or system signature. With an understood set of behaviors, the worm can become quickly revealed.

The limited attack capabilities of the worm network's members will quickly become a hindrance. As the vulnerabilities become well known and patched, the number of potential targets for inclusion into the worm system decreases, and the worm will eventually come under control, having nowhere to run.

This also leads to a signature for host based intrusion detection, using logfile entries or filesystem modifications as a basis of infection. For network based intrusion detection the signatures of the remote attacks can quickly be identified and associated with the spread of the worm.

A finite set of reconnaissance capabilities will also quickly become a problem for the worm network. As nodes are added, each will generate increased traffic. This reconnaissance traffic will also be associated with the worm, identifying the source nodes as compromised.

5.2 Growth Rates and Traffic

The growth of a worm network is typically exponential in nature. This arises from the fact that each instance of a worm network node operates independently of its siblings, identifying targets and infecting new nodes. This operates in parallel, and scales rather quickly.

Consider a worm network which is capable of adding 5 new hosts every round. After five rounds, 3125 systems are infected and active participants. After five more rounds, ten rounds total, 9,765,625 hosts are members of the worm network.

The growth of the worm network can be described by the following equation:

$$d < T_{node} >= k < T_{node} > dt \tag{1}$$

where $< T_{node} >$ is the average node traffic, equal to:

$$< T_{node} >= \sum < T_{comm} n_{comm} + T_{attack} n_{attack} + T_{scan} n_{scan} > \tag{2}$$

This is nothing more than the sum of the total traffic generating portions of the worm node, communications, attacks and scans, and the number of events for each traffic component. In Equation 1, k is the infection rate constant, or the mean number of hosts infected per round of attack. This rate constant can be tuned, and indeed some worms do tune this variable. This can be done by having a fixed number of infections per node, or, more commonly, delays between infection rounds. When this equation is integrated, it becomes:

$$< T_{node} >=< T_{node} > e^{-kt} + C \tag{3}$$

where C is the baseline traffic that mimics the worm's traffic. The total traffic for the worm network, T_{worm}, is therefore the summation of the traffic all of these nodes.

With time, the worm's traffic grows. At any time the worm's visibility is proportional to the fraction of the traffic on the network:

$$fT = \frac{T_{worm}}{T_{total}} \tag{4}$$

Hence, as time passes and the worm attacks new hosts, its fraction of the total network traffic will increase, and its visibility profile will increase as well. This has been observed in many typical worm networks, for example in email worms which often shut down their mail server infrastructure due to an overloaded network and server.

The growth of two different types of worm networks differs wildly. In a common scenario, where the infecting agent removes the vulnerabilities used to gain entry, the worm network will grow in a manner that mirrors biological systems. In theory, the growth rate of such a worm network should reach a steady state value before trickling off, and finally ceasing growth. This would occur even in the absence of removal of infected machines from the network. As the number of targets available to the known attack methods decreases, the 'food supply' for the worm system is depleted and growth will slow. Eventually the growth rate will stop as no new nodes available for addition are found, as all of the vulnerabilities usable by the worm network are removed.

If, instead, the worm leaves itself still vulnerable to attack by another instance of the worm system, the growth of infected machines will reach a plateau, and the rate of infection will mirror that for the above situation, but the number of infections will continue to rise as multiply infected machines are attacked. Eventually network capacities will be reached and saturated with nodes scanning for and infecting new nodes. Only then will the worm network growth stop.

The second case may seem to be a pointless repeat of the cycle of target acquisition and assimilation, but this is usually not of great concern. Members of the worm network will always remain a minority of the total number of systems on the Internet, so the probability of finding an already infected node is slim.

This exponential growth has an impact on the worm's profile as time passes. As more systems are introduced to the worm network and begin their attempts to identify new targets, the number of systems actively scanning increases. This leads to an exponential increase in the amount of traffic produced. All of this traffic is nearly identical in nature in worm networks with a limited capability set.

The problems of this design are immediately obvious: first, the food supply will quickly run out; secondly, the profile of the worm network will increase rapidly with time as increasing amounts of signature traffic is generated.

5.3 Network Structure

Additionally, the structure of the worm network has two important consequences. First, it rapidly fans out, communicating with as many machines as it can, often as rapidly as possible. Basic network monitoring of a host or a network will reveal that the infected system is suddenly communicating with a great deal of new hosts, usually in a predictable fashion. If this system is high in the chain of heirarchy in the worm network, as more hosts are added its traffic load will increase as they send updates to this host or receive direct communications from this node. This also leaves an audit trail which can be followed to determine the paths of infection, identifying hosts which may have remained otherwise undiscovered.

Secondly, it affords no control over the direction which the worm will take and the capabilities it can utilize. Worms so far have worked to spread as rapidly as possible and as disperse as possible. Any penetration of a particularly valuable network (such as a corporate network) is purely the result of serendipity. While deep penetrations into traditional strongholds of security can occur, it cannot be an explicit goal with this technique.

5.4 Intelligence Database

To date, all of the major worms that have utilized a reporting mechanism as new nodes are added have relied on a centralized facility for the reciept of such updates. This includes email drop boxes, the presence on a specific IRC chat room, or a simple packet to a particular IP address.

This has two disadvantages. First, it becomes possible to identify the complete, or nearly complete, membership list of the worm network by obtaining that database. When the system that receives the updates is under the control of cooperative administrators, this can be readily accomplished. Alternatively, an operation could be performed and the main intelligence machine compromised by those attempting to erradicate the worm network, with the attackers obtaining the database.

Secondly, it becomes possible to block the updates if they have a predictable form. Using central email servers, it can be possible to block outbound mail to the known accounts. With a firewall a block on the network traffic that indicates a new node has been added. This is even without the cooperation of the network that is hosting the database, which may be difficult to obtain.

In either case, a centralized database which contains a full, or nearly complete, list of the nodes in the worm network is a risky proposition. It leaves the entire worm network open to discovery with one mistake.

6 Considerations of Future Worms

Having outlined several of the dominant limitations of current worm paradigms, we can now develop some of the considerations of improved worms.

We will not discuss here strategies such as process hiding, the use of kernel modules or rootkits, all of which are based on older, existing techniques. Zalewski has covered many of these in his earlier paper on WormNet[11].

6.1 Infection Mechanisms

True inspiration for novel mechanisms of the spread of a worm network come from living systems. Many of the ideas can be adopted into the world of network computer systems.

The life cycle of an infection begins first with the attachment of the infecting particle, a virus or phage, to the cell which will be infected. This usually comes as a result of a complementarity between the infecting agent and the target host. This is then followed by the entry of the infecting material, such as DNA and accessory proteins, into the cell. This material is then used to hijack the cell into becoming a virus production factory, increasing the number of viruses and, ultimately, the number of infected cells.

Worms act in much the same way, as do computer viruses. However, one facet that has long been overlooked is the difference between passive and active methods of finding new hosts to infect.

The passive nature of biological infection is due to the ability for the infecting agent to exist in the transport medium, be it blood, water or air. This is the substantial problem, however, in applying this paradigm to computer worm infections: infectivity agents cannot exist in the

[11] "I Don't Think I Really Love You, Or Writing Internet Worms for Fun and Profit", Michal Zalewski, 2000. Available at http://lcamtuf.coredump.cx/worm.txt.

transport medium alone (the network), they have to exist on a host (or even a network device, such as a router). This problem exists for two reasons: first, the capacity of the network is transient, meaning that the network itself is not a storage medium. While we speak of Gbps networks, the bits reside 'on the wire' for only a brief period of time. Secondly, for any section of code to operate it must be executed by some processor, at the very least. Aside from network infrastructure equipment (ie the processors on switches, routers), there is nothing to interpret and execute the worm programs.

To utilize this idea, the infected machine can become opportunistic about infections and observe the traffic as it goes by. In this manner targets are identified with a minimal amount of traffic generated by the reconnaissance node. Traffic analysis can be utilized to determine the type of machines on the network, server vs workstation vs network component, and the services, with resulting vectors of entry.

Passive infection methods include insertion into a file transfer stream, such as over Windows file sharing, MacOS file sharing, or NFSv3. In this manner one of the simplest attacks would be a hijacking condition, where the server data is 'bumped' off the network and malicious data is substituted.

6.2 Network Topology

As described above, the structure of a typical worm in existing instances has been a rapidly branching tree. This has the effect of assimilating a large number of machines quickly into the worm network and generating an ever increasing amount of associated traffic. Two alternative structures to a worm network can be utilized to provide stealth, survivability to the worm network, and maintain an effective penetration strategy.

6.2.1 Guerilla Structure

The guerilla method takes many aspects from the organization of guerilla armies and coup d'etats as mentioned in the books "1984" by George Orwell, "Guerilla Warfare" by E. Che Guevara, and "The Moon is a Harsh Mistress" by Robert Heinlein. Basically, the guerilla method takes the features of the broadcast method (e.g. exponential growth speed and redundancy) and augments this by using intelligence gathering techniques along with compartmentalization to protect the safety of the whole network.

This method extends the metaphor of a rogue assault (the worm) on an established entity (host machines) in a semi-transparent, complicated terrain (the network). Upon introduction, the worm needs to survey the terrain. This can be done via many means, some as simple as basic traffic analysis, via sniffing the network, to identify NIDS machines or central servers, or as complex as reading DHCP packets to learn network layouts, or actively probing machines in a subnet. Once some semblance of the terrain is gathered, the worm can then make decisions as to how to behave in the network environment. Based on this information, it may want to simply fan out and broadcast or run a few parallel chains. If it is in a hostile environment, it may want to lay low for a while or simply cease to spread if the risk of exposing itself is too dangerous[12].

Another key feature to guerilla tactics is compartmentalization. In a guerilla war, discovery of the rogues is a severe risk. To ensure the safety of all of it's members, nodes isolate knowledge

[12]Though such a turn-off mechanism could be a severe weakness in the spread of the worm. Imagine a scenario where false traffic is generated which triggers the shutdown reaction for the worm node.

of each other to a functionally minimal level. Assuming that an intelligent worm wants to communicate with other instances, it would need to store contact information about other nodes. This is a problem, as if one node is compromised and its data analyzed, it can supply information about other nodes. We discuss below communications topologies which can help mitigate these risks.

One way to control the damage of captured nodes is to compartmentalize subsets of the nodes so that you have fully connected isolated sub-graphs of nodes that only know about themselves. This can work via out of band communication to an anonymous source (e.g. Freenet or Usenet) or a predetermined marking method (e.g. leave this randomly assigned UDP port open on infected machines for this cluster). Alternatively the worm could use objectives (say, infect n machines) and you generate a fixed number of node members such that the compartment is created and jettison all knowledge of the other nodes. Then all child nodes from that parent are part of your compartment. That way, there will be little chance to destroy all instances of the worm, especially if it starts out in several distant places on the network.

6.2.2 Directed Tree

This topology differs from the traditional fan network or the guerilla topology in that instead of trying to spread out to as many hosts as possible, the worm limits its search to one host at a time. This has the effect of creating a chain of infection as mere jumps from machine to machine. External detection will be much more difficult from the perspective of monitoring for an activity spike, however, this has several important disadvantages.

First, the spread time for the network is taken from exponential to linear time. This is a severe slowdown and detriment to the effectiveness of a worm. Secondly, it will leave a direct audit trail, if there is any deterministic method the chain uses to go from machine to machine, it can easily be backtracked (even pseudo-random number generators). Thirdly, it would create a running single point of failure for the lifespan of the worm. Simply stated, if the worm jumps into a system where it can be contained, such as a honeypot or a machine that is about to shutdown, the worm will effectively die.

This type of worm structure is difficult to successfully execute. Without prior knowledge of the network topology which is to be penetrated, there is a high probability that the direction of the tree will be unproductive in relation to the stated goal of the worm. The highest utility of this worm structure would be in a large network, where a subnet is to be penetrated. Furthermore, some feedback mechanism would be required to initiate a new chain from some higher point after failure. This makes, at the present time, this type of worm network an unlikely structure.

6.3 Communications Topology

Two major points must be addressed when considering worm communication topologies: latency of communications, and the resistance to discovery of the network by traffic analysis. Each of these will have a significant impact and must be balanced. This mix, and the resulting topology and communications methods, can be altered for the purpose of the worm network.

Latency is introduced when the communications model is designed. Two methods of communicating, for updates to the nodes or to the member database, are intuitively obvious. The first is a broadcast mechanism. In this case, a central node sends a message to all nodes simultaneously. Utilizing the list of member nodes, a master site sends messages to each node with a payload. This can include a capability injection point announcing a new component, the

database requesting an updated status from the member nodes, or a master node directing an attack, for example in a DDoS ring.

This model introduces the lowest latency, as all nodes are contacted at the same time, and the time to send the message from the first node to the last is a fixed value. However, it has various features which makes it unattractive when the traffic is analyzed.

Basic traffic analysis would immediately show one system sending a flurry of communications, followed by a period of high return traffic directed at or in the vicinity of the source host. Due to the widespread deployment of traffic monitoring tools such as NIDS systems and generic network accounting tools (ie network flows on routers), this becomes unattractive due to the high profile that would be incurred as the network scales.

An alternative model is to step down the tree of member nodes, from a central position to the edge machines. In this case, each node acts in a store and forward routine, receiving the traffic from its parent and passing it on to its children.

The latency here would scale as the depth of the worm tree of infection. Furthermore, due to protections incurred, such as sending traffic at optimal times, the delay in moving from the source to the furthest edge of the tree could be several days[13]. Under optimal conditions this is only a few minutes, depending on the number of nodes and the time it takes the parent to determine whom to communicate with.

This would be advantageous, however, from a traffic standpoint. At any one time a minimal amount of traffic is used to communicate between parents and children. Traffic analysis would not reveal anything other than a normal flow of traffic, assuming stealth channels were used as described below.

The high latency, store and forward mechanism would be most suited for a highly connected worm network, such as the traditional fan network with many branches or the directed tree with minimal branches. This also adds a cellular type of structure to the intelligence database, where each node knows only about its immediate parent and children. This is very similar to the routing tables of networks, where each router doesn't have to know the entire network topology, only how to reach its neighbors. A guerilla network would require some form of broadcast, however, to communicate across worm cells. Techniques to communicate using covert or stealth channels are discussed below.

6.4 Communications Methods

Due to the great intelligence needed by the worm to achieve a specific objective, receive updates from various sources, such as scouts and new capabilities, communications channels have to exist. Furthermore, to protect the integrity of the worm network, the existence of the sender and the receiver, and their participation in a larger network, must be guarded. To do this, we will utilize various covert communications methods.

It is tempting to employ the use of encryption for communications methods. There is a problem with this use, however. When the entropy of the bits is evaluated for normal traffic, it shows a non-random distribution for standard, unencrypted sessions. While encrypted traffic such as SSH or SSL traffic will indeed be present on the wire, the bulk of the data will be

[13] Assume a situation where the node is smart enough to monitor its usage and knows to send traffic via HTTP channels when use of the web is at its highest. This point may be the lunch hour for a workplace. If the update is received shortly after this period, and HTTP traffic is low and the node's traffic would stand out more, it would have to wait until the next peak period to transmit its update, nearly one day later. The worst case scenario is where every node is in a similar situation.

unencrypted. Following the same methods as used by Shamir[14], the variance in the entropy of the bits on the wire can be evaluated. In this manner, simple traffic analysis will highlight interesting traffic, traffic worth observing. When correlated with other analysis, infected machines will be readily identifiable.

Hence, we feel that the use of steganography is more preferable. By hiding the presence of the data in traffic that appears normal, the presence of the worm can be hidden as well. Furthermore, by utilizing simple analysis of the traffic, the infected machine can begin transmitting during a burst of traffic in its local network segment. This will further hide its presence.

Data can be hidden in a significant number of places on network traffic. One such strategy draws upon the method discussed by Simple Nomad[15]. The methods described use promiscuous mode listening stations which have the data sent *near* them from spoofed addresses. Data is then hidden in a variety of fields, such as HTTP requests, DNS information, and SMTP message ID values. The key is that by sending the data *near* the intended recipient, investigators will have difficulty in finding the compromised machines, as they would have to search all machines in a network segment.

Our overriding paradigm for stealth communications for a small worm will be to use an existing infrastructure as much as possible. This will lessen the communications overhead of the worm system and allow for a small payload for either endpoint of the communication. Furthermore, it helps to ensure that the traffic is likely to look legitimate for the network the worm finds itself on.

Infrastructure components that can be employed to communicate across wide areas using broadcasts yet remain undiscovered include the Usenet network, web cached mailing lists, DNS servers and various web boards.

Spam makes an effective covert channel for minimal information. Usenet and mailing list spam[16] can be utilized by a central system to notify the nodes of an updated condition. This could include directives concerning new capabilities or an update concerning directives. Image data and embedded communications would be useful on Usenet groups as well. Spam messages in those cases usually go unchecked, providing that cross posting thresholds have not been crossed and spam detection has been tripped.

Nodes then need only peek at the posted communication and, if needed, decipher it to obtain the updated information. Usenet spam makes an attractive node notification scheme due to the sheer bulk of data in a typical Usenet stream.

This kind of mechanism makes for an attractive, and low profile, broadcast mechanism for updates and communications from one node to many. Furthermore, the use of spam messages or data embedded in binary content (ie image data) on a medium like Usenet, which contains thousands of similar images, can be utilized by the nodes to communicate with a central site.

Additional communications mediums include the use of peer-to-peer networking models such as Napster, the DC Freenet, the Gnutella network and their clones. The DC Freenet is an especially attractive network to use as a communications medium for content updates (see below). The persistence of data on such a network is dependent upon the perceived value of it, which is simply a measure of its popularity[17]. The use of a popular game or pornographic

[14]"Playing hide and seek with stored keys", Adi Shamir and Nicko van Someren, September 1998.

[15]In a talk entitled "Stealth communications across networks", at the SANS Technical Workshop, May, 2001, as well as at the July, 2000, Blackhat Briefings.

[16]This is best done using lists which have no moderator and are cached on the web.

[17]The paper 'What's on Freenet?', available at http://www.openp2p.com/pub/a/p2p/2000/11/21/freenetcontent.html, examined the content and persistence of various files on Freenet.

images to deliver the update, using embedding techniques, would facilitate the persistence of such data. Nodes could then obtain the data from this pool upon receiving notification. We elaborate on this topic below.

6.5 Target Selection

There are two aspects to target focus that are worth noting. The first is the nature of the specific hosts targeted by the worm, and the second is the type of targets.

In the past several years, the rise of consumer broadband, coupled to the increased number of devices utilizing full OS installations on embedded devices, has created an attractive number of targets. This leaves a network target that is both always online and difficult to secure. While a consumer can readily patch and remedy an exposure on their desktop or server system, the nature of embedded devices makes it difficult to upgrade for a typical consumer.

One popular situation is the adoption of a nearly complete Linux distribution for a specific task. Devices that may be targeted include cable modem routers and DSL devices, printers, small appliance firewalls, and even some television devices which lie on the network (such as a cable modem). These are often based on outdated distributions of an operating system which contain well known security flaws.

By specifically targeting these devices, a number of worm nodes can be assimilated, which may prove valuable for target identification or other high profile tasks. These systems can be disposable, and new ones acquired readily. Furthermore, they rarely have logs that are examined, so they would make excellent points to create anonymity.

The second problem is made increasingly larger through the use of networks such as Akamai and other distributed content systems[18]. For various reasons, maintaining a comprehensive security plan, including keeping a large number of systems up to date, and accepting the strategy used to inject content into the Akamai system, the security level of the system can never be too high.

This also makes an attractive political or financial target. With heavily used sites like Yahoo! and CNN utilzing networks such as Akamai, the ability to rapidly spread a message of political content is large. Furthermore, through various mechanisms, including technical and social, financial havoc can be achieved in a short timespan.

The serendipity used by a worm can help it to gain access to networks that are normally not prone to security attacks. If the worm system is capable of identifying a goal, such as an internal network like Yahoo!, and taking a specific action, this chance entry can be used for a specific purpose. While a worm could have a significant radar profile for intrusion detection systems, an opportunity such as this could warrant the possible discovery if motivation was sufficient enough.

Other politically minded worms include the Noped worm[19] and the Cheese worm[20] . Noped infects Windows systems and attempts to weed out child pornography and alert legal authorities. Cheese attempts to remedy the damage caused by the L1on worm, though it has been found to leave a backdoor entrance mechanism. These vigilante style worms follow a call for action after various Outlook worms decimated many corporate email infrastructures.

[18] As this paper was being finalized for BlackHat, reports began circulating that a hacker had gained access to the SourceForge, themes.org, and Apache systems, and had also gained a list of Akamai account passwords. A mirror was placed at http://66.92.75.28/ vladimir/themes-org.html but deleted the passwords

[19] http://www.symantec.com/avcenter/venc/data/vbs.noped.a@mm.html

[20] http://www.symantec.com/avcenter/venc/data/linux.cheese.worm.html

6.6 Dynamic Behavior

So far our definition of worms has focused on a complete replication of the worm from one host to another. Thus, every instance of the worm is identical. However, there is no need for each member of the worm network to be identical, as they can have specific roles or capabilities. This variation in the worm network can occur on two different levels: the microscale, where the worm programs behave in a constantly changing fashion, and on a macroscale, where the members of the worm network do not appear to be identical on different hosts.

6.6.1 Microscale Possibilities

A great deal of work has gone into studying computer viruses which utilize polymorphic behavior, whereby they become more difficult to track and identify. Computer worms so far have not yet adopted these strategies, but by including them, they become more difficult to identify as a larger instance of malicious code.

Some viruses have utilized encryption, in a self decrypting and extracting fashion, to evade detection. By using varying keys, the virus body changes with each infection. Worm components could also do this, obscuring their true nature. However, just as for network data, encryption can be used in a scan to highlight interesting data[21]. As such, simple encryption schemes that preserve the entropy of the underlying bits should be used.

In a basic sense, this would include dynamic capabilities which can be modified as the situation warrants. This has been observed in a limited sense with two well known malicious code examples, BO2K and TFN2K. Each of these had the possibility of utilizing any of a few methods to communicate, including TCP, ICMP and UDP, and hooks to provide optional encryption. This is one simple example of the type of polymorphism that would make a worm component more difficult to track. These adjustments could be made by the worm, either using a statistical analysis of the traffic it observes, choosing to blend in to the surrounding network traffic or to utilize available mechanisms (ie GRE), or randomly chosen protocols and ports, negotiated with a communicating partner.

One more interesting trick would be to utilize a multithreaded engine, possibly even using nonsense threads, that would assist in masking the tracing of the application's behavior.

An additional method to obfuscate the application's execution, and thus mask the behavior from detection schemes, would be to have a variety of methods to chose from, either at build-time or run-time, such as searching and sorting algorithms, encryption algorithms, and data processing algorithms.

6.6.2 Network Scale Behavior

So far we have witnessed worms that work in a monolithic fashion, which is to say that each node of the worm network contains the same code as every other node. This is, in fact, one of the parts of the definition of a worm. However, this does not have to be the case.

The six components of the worm as we outlined in Section 2 can be fit together in a variety of ways to provide for a dynamic worm network. Within that, there can be a variety of modules to accomplish each of those tasks.

For example, there can be several reconnaissance modules, some of which perform various types of network scans, fingerprint systems in a variety of ways, and some which are passive and

[21] See the Shamir paper listed above on stored keys.

analyze traffic. This can be combined with a variety of methods for communication, along with varying attack capabilities, to create a dynamic network of worms.

This can be expanded upon to adopt a strategy of worms where they adopt various roles, rather than being complete systems unto themselves. For instance, a worm node could lack attack routines but be rich in methods to identify targets, becoming a scout node. Others could be attack nodes, with several methods to gain entry to a remote target included with them. Lastly, an intelligence node would lack both the reconnaissance and attack components, and focus instead on communications techniques.

6.7 Dynamic Updates to the Worm Nodes

As we stated above, the ability to adopt new capabilities into the worm infrastructure has been an as yet underutilized function of most worms observed in the wild.

There are, however, some considerations that must be taken into account. New capabilities must be propagated, and their existence first communicated to the nodes, through some mechanism. The communications concerns are addressed above (Communications Methods), but the capability of distribution mechanisms represent a larger problem with two concerns that must be addressed: the size of the message is larger than a simple communications update, and the trustworthiness of these updates.

The size concern can be addressed with a reasonable design of the worm instances. By using a mechanism that is friendly to modules from the beginning, only small modules will have to be distributed. For example, if it were a new buffer overflow exploit, the socket code wouldn't have to be rewritten, only a few parameters, such as the connecting port and shell code.

By utilizing this modular approach, small updates can be made, which can then be hidden in several places, such as embedded in web or Usenet images which are then extracted by the clients.

This system easily leaves the worm network open to injection of malicious code, designed to subvert the goals of the worm network. To overcome this, the use of encryption and digital signatures can be used. A minimal PKI can be established with the worm network to provide a mechanism for authentication of the modules using a minimal RSA scheme. The key pairs of the issuing machines could be used to authenticate the source of the update.

The encryption functions could be used also to protect the contents from analysis. As noted previously, the bit entropy should not be maximized so that it can blend in to the neighboring data on the network.

7 Defenses

We turn our attentions to defense strategies for combatting worms, present and future. Because of their increased popularity and use, detection and defense strategies will have to grow to meet the rising threat. Also, worms require a different scale and type of analysis than manual intrusions.

There are, of course, some obvious defenses:

- Staying up to date with security patches
- Defense in depth, not simply perimeter defenses
- Installing intrusion detection and response mechanisms

These represent nothing more than a repeat of the same mantras that are at the core of any security system. While these will certainly help defend a site or a system against intrusions by worm agents, they cannot be achieved totally and cannot be totally effective at defeating all attacks.

Having stated the obvious, we now turn to different forms of analysis that may serve as useful indicators of a larger system at play.

7.1 Worm Network Identification

Before we can begin to identify new and dynamic worm systems, we must first describe how we can identify existing worm networks based on the current paradigms outlined above. While this may seem readily available, differentiating initial worm activity from manual attracker activity can prove difficult. Worms are usually identified only after a node has been captured and the infecting applications analyzed.

Worm activity can be identified, even in the absence of captured nodes and the resulting analysis, by a distinct traffic shift. Typical worm behavior, such as those described above as current worms, will have a small set of traffic patterns grow exponentially, usually rapidly. This traffic will include scanning activities as well as attacks. The exponential increase in similar traffic across a wide area within a short timespan is a telltale sign of a worm system growing. Furthermore, these scans and attacks happen often within very short timespans of eachother even though they are quite distant from one another in the space of a network.

To abstract this analysis, we can begin to use correlation analysis to develop signatures of worm networks. Correlation analysis is a mathematical method of evaluating the connectedness of two variables. Auto-correlation is the connectedness of the same sample with itself, and cross-correlation is the analysis to determine the connectedness of two different samples.

In this scheme, scans and attacks would be the two elements under corelation analysis. Auto-correlation analysis would show the correlations between scans or attacks, meaning that a high number of similar scans or attacks happening in a short timespace would be indiative of a worm network being active. Cross-corelation analysis would evaluate the connectedness of scans with attacks, such as a quick followup of an attack after a scan or service sweep of a network. This connectedness is usually time dependent. For most worm systems, these correlations would be high when evaluated on a per-source basis, especially with most worms moving as fast as they can to scan and attack additional networks.

This correlation analysis can be extended to the dynamic worms discussed above, though it would require additional investigations into the attacks and scans the worms are performing. Because of the variances in the scans and attacks used, the disparity between the behavior of one node of the worm system when compared with another may lead to a loss of some points of analysis.

7.2 Strategies for Worm Node Detection

Because of its dynamic content, when nodes of the same worm are compared, a signature based approach to intrusion detection will be difficult to develop. This is due to the possibility of of updates to the worm nodes and their parents. On a fundamental level, it would appear as though possibly several groups of attackers were increasing their activity. Against the typical baseline of probes and attacks on the Internet, diverse activity may be difficult to place in terms of a larger set of activity.

Though the worm is dynamic, with various capabilities ensuring that no two instances of the worm appear identical, it does have a core set of code that acts as a 'glue' layer to bind these modules together. This element will be the common thread that binds parts of the worm together. It is on this that detection strategies should hone.

For a more dedicated defender, traffic analysis would be an effective strategy to detect the presence of a larger worm system. Using the approaches of anomoly detection and statistical analysis of the traffic patterns, deviations should appear which suggest malicious activity. The analysis required, though, is quite large, and would most certainly only be justifiable after other indicators suggest a larger problem is afoot.

An additional strategy that would be helpful is an increased use of agent based intrusion detection systems. With a large number of host based sensors on a network, and an appropriate central analysis system in place, trends could be spotted as they occur. An increase in probes, attacks, and possibly anomolous traffic, with timings that are similar, would suggest a larger system is at work. Analysis would also have to cross networks to show an upsurge of similar attacks and traffic across networks within the same time frame. Furthermore, the time dependency of this information would have to be analyzed to show an exponential increase in the number of such scans or attacks, along with an increase in the number of source machines.

Simply put, though, without any idea to look for something larger, detection efforts would be futile. Without something on which to focus, this analysis would be unable to find the anomolous behavior.

Unfortunately, anomaly detection is in its infancy and still largely a pipe dream, and host based intrusion detection is an unwieldy situation for overall management. This will certainly slow its adoption, however the gains should be encouraging enough to spur on feasibility research and product improvements.

7.3 Attacks on the Worm Network

Like any network, a worm network will have vulnerabilities inherent in its design and implementation. These can be exploited to gain control of the worm network and alter its behavior.

Assuming minor errors in the development of the worm network, such as weak authentication models or default passwords, the network itself can be vulnerable to attacks. These attacks mirror the known attacks on DDoS rings[22], or attacks on routing infrastructures[23].

To perform such an attack, a significant amount of traffic analysis will have to be performed on a compromised system. This is required to identify the types of communications it makes, with whom it communicates and what sorts of mechanisms are in place. If cryptography is used, cryptanalysis must be performed on the commuications, but this can be greatly facilitated by analysis of the worm components found on a system.

For dynamically updatable worms, however, with commuications mechanisms updated to

[22]Tools like 'Zombie Zapper', from the Razor team at Bindview, exemplify this technique. ZZ works by sending shutdown messages to the DDoS member nodes, though it assumes default ports and authentication tokens. More information on ZZ can be found on http://razor.bindview.com/tools/ZombieZapper_form.shtml

[23]A significant number of problems exist in Internet routing protocols, including weak or absent authetication mechanisms, allowing for forged routing table updates. In the worm networks described above, most of the communications mechanisms rely on knowing about the other members. In this situation their membership lists are much like routing tables. Without strict checking mechanisms in place, the worm is vulnerable to these attacks as well, leading to lost nodes, capability unloading, or even shutdown of the worm subsystem. Routing vulnerabilities are described in a paper by Curt Wilson at http://www.netw3.com/documents/Protecting_Network_Infrastructure.htm

support new ports, protocols and authentication mechanisms, this attack would require similar analysis of every node captured. This may make this approach difficult, if not impossible, to mount on a well designed and highly heterogeneous network of worm nodes.

7.4 Future Considerations for Defenses

The above strategies are offered as stimuli for the discussion of how to detect and defense against a dynamic worm network. They are not complete nor exhaustive, only representations of the adoption of existing strategies, as well as some evolved streategies, to detect evolving worm networks.

There are inherent problems, though, in these strategies as outlined. They are labor intensive, requiring analysis of a significant number of worm node stations to identify the presence of a larger worm network. They also would require, from a host based or network based standpoint, analysis of the worm binaries to indentify the common components and detect them. This fact will certainly slow adoption of these strategies, but we hope they will encourage the development of new strategies.

8 Conclusions

This paper has provided a new framework for the evaluation of existing network worms and analyzed three well known Internet worms within this context. Limiting factors of the current worm paradigms were then analyzed, and their consequences outlined. These then lead to considerations that will apply to new worms, which will call for new detection strategies.

It is this new paradigm that will be difficult to defend against. It is built on several new ideas, namely that not all nodes are required, not all nodes are required to look the same, and the traditional rapid spread model is too high profile for continued success. By moving to distributed points of action, and incorporating update strategies, along with new network and communications topologies, worms can become difficult to identify and defend against.

We expect these ideas to become prevalent as worms evolve and their use grows.

9 References and Bibliography

Several references were key in the development of these ideas. The interested reader may want to examine them for further information.

The following papers are available from the IBM Antivirus Research Center library, found at http://www.research.ibm.com/antivirus/:

- *An Undetectable Computer Virus*, David Chess, Steve White
- *The Future of Viruses on the Internet*, David Chess
- *How Topology Affects Population Dynamics*, Jeffrey Kephart
- *Directed-Graph Epidemiological Models of Computer Viruses*, Jeffrey Kephart, Steve White

The following paper is available from the IBM Systems Journal, volume 35, numbers 3 and 4, 1996:

- *Techniques for Hiding Data (and references therein)*, Watler Bender, Daniel Gruhl, Norishige ¡Morimoto, Anthony Lu

The following is available from the 1992 proceedings of the Winter USENIX Conference:

- *NFS Tracing by Passive Monitoring*, Matt Blaze

The following is available from the NCSC Rainbow Series library, available online from the National Security Agency at http://www.radium.ncsc.mil/tpep/library/rainbow/:

- *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, National Computer Security Center

The following reference is available from the Proceedings of the Financial Cryptography 1999 conference:

- *Playing Hide and Seek With Stored Keys*, Adi Shamir, Nicko van Someren

The following paper is presented on the O'Reilly Peer-to-Peer website:

- *What's on Freenet?*, Jon Orwant

The following references are available from the website of Fred Cohen and Associated at http://all.net/:

- *Attack and Defense Strategies*, Fred Cohen
- *Computer Viruses - Theory and Experiments*, Fred Cohen

The following is available on the Sun Microsystems website at http://www.sun.com/:

- *Java Remote Method Invocation - Distributed Computing in Java*, Sun Microsystems

The following paper is available from http://www.symmantec.com/:

- *Understanding and Managing Polymorphic Viruses*, Carey Nachenberg, Symantec, Inc.

The following presentation is available from http://www.research.att.com/ smb/:

- *Computer Insecurity*, Steven M. Bellovin

The following reference is especially noted as inspiring:

- *I Don't Think I Really Love You, Or Writing Internet Worms for Fun and Profit*, Michal Zalewski

Interested readers may want to consult these volumes for background material:

- *TCP/IP Illustrated, Volume 1*, W. Richard Stevens
- *Intrusion Detection*, Rebecca Brace
- *Codes and Cryptography*, Dominic Welsh
- *Handbook of Applied Cryptography*, Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone

Vulnerabilities discussed above have been cross referenced to these two vulnerability databases:

- *CVE*, Mitre Corporation, http://cve.mitre.org/
- *Bugtraq Vulnerability Database*, SecurityFocus, http://www.securityfocus.com/

The following paper is especially worthwhile reading for a futher underastanding of computer worms and their communications:

- *A Mathematical Theory of Communication*, Claude E. Shannon

The following two papers discuss the 1988 Morris Internet Worm and its impact in high detail. Please also see the website http://www.worm.net/:

- *With Microscope an Tweazers: An Analysis of the Internet Virus of November 1988*, Don Becker
- *RFC 1135: The Helminthiases of the Internet*, J. Reynolds

10 Acknowledgements

The main author of this paper, Jose Nazario, wishes to extend his thanks to the other contributors to this work, J. Anderson, R. Wash and C. Connelly, for their helpful discussions, input, and editing of this manuscript. It's always a pleasure to work with them, and the rest of the group at Crimelabs. Additionally, Simple Nomad and Dug Song have provided helpful discussions and insights, and the prior work by Zalewski is to be noted as inspiring. Also, numerous people have sent me copies of worm variants through private email, and I thank them for their help in this research.

The author also wishes to extend his most sincere appreciation to the organizers of the Blackhat briefings for this opportunity.