

Early Detection of Active Internet Worms*

Vincent H. Berk, George V. Cybenko, and Robert S. Gray

Institute for Security Technology Studies, Thayer School of Engineering
Dartmouth College, Hanover, NH 03755
FirstName.LastName@Dartmouth.edu

ABSTRACT

An active Internet worm is malicious software (or malware) that autonomously searches for and infects vulnerable hosts, copying itself from one host to another and spreading through the vulnerable population. Most recent worms find vulnerable hosts by generating random (pseudo-random) IP addresses and then probing or scanning those addresses to see which are running the desired vulnerable services. Detection of such worms is a manual process in which security analysts must observe and analyze unusual network or host activity, such as scanning activity, exploit traffic, or the impact of the running worm on a production host. A worm might not be positively identified until it already has spread to most of the Internet, eliminating many defensive options. In this chapter, we present an *automated* system that can identify active scanning worms soon after they begin to spread, a necessary precursor to halting the spread of the worm, rather than simply cleaning up afterward. Our implemented system collects ICMP Unreachable messages from instrumented network routers, identifies those patterns of unreachable messages that indicate malicious scanning activity, and then searches for patterns of scanning activity that indicate a propagating worm. We examine an epidemic model for worm propagation, describe our ICMP-based detection system, and present simulation results that illustrate its detection capabilities. In addition, since effective automated detection of scanning worms will encourage worm authors to use other approaches for identifying target hosts, we give a brief overview of proposed techniques for detecting slow-moving or stealthy worms.

1. INTRODUCTION

An active Internet worm is malicious software (or malware) that autonomously spreads from host to host, actively searching for vulnerable, uninfected systems. The first such worm was the 1988 Internet worm, which spread through vulnerable Sun 3 and VAX systems starting November 2nd, 1988.¹ This worm exploited flaws in the `sendmail` and `fingerd` code of that time, and through the `rsh` service and a password-cracking library, also exploited poor password policies (such as a user having the same weak password on multiple machines). The worm collected the names of target hosts by scanning files, such as `.rhosts` and `.forward`, on the local machine, and then attempted to infect those hosts through the `finger`, `sendmail`, and password-guessing exploits. Although the exact number of infected machines is unclear, the worm infected enough machines to disrupt normal Internet activity for several days due to high network traffic and CPU loads.

Recent examples of active worms include Code Red v2, which exploited a flaw in Microsoft's Internet Information Services and infected 360,000 machines,² and Sapphire/Slammer, which exploited a flaw in Microsoft's SQL Server and infected 75,000 machines.³ Code Red, Sapphire/Slammer and most other recent active worms find vulnerable machines by generating random (or pseudo-random) IP addresses and then probing to see if the desired vulnerable service is running at those addresses. Compared to the 1988 Internet, the modern Internet has so many hosts that random probing is an effective way to find vulnerable machines. In 1988, the address space was sparsely populated, and the 1988 worm, if it had used random probing, would have needed years (or even centuries) to find even one *existing* machine, let alone a vulnerable machine. In addition to using random probing, most recent worms probe as quickly as possible, so that the worm can spread to most vulnerable machines before system administrators have time to shut down infected machines and repair the exploited security

*Supported under Award Number 2000-DT-CX-K001 (S-2) from the Office of Justice Programs, National Institute of Justice, Department of Justice, with additional support from DARPA under Contract Number F30602-00-2-0585. Points of view in this document are those of the authors and do not necessarily represent the official position of the United States Department of Justice.

hole. In fact, since current response is entirely manual, a worm only has to spread faster than human response time to succeed. Sapphire/Slammer, the fastest spreading worm to date, far exceeded human response time by infecting most vulnerable machines within five minutes of its launch.³ Clearly, if the Internet community wants to halt the spread of a worm, rather than simply cleaning up afterward, some form of automated detection and response is needed. Here, we will focus on the problem of *detection*, and present an automated system that can identify active scanning worms soon after they begin to spread, a necessary precursor to halting the spread of the worm, rather than simply cleaning up afterward. Worm authors, when faced with such a detection system, might switch from address scanning to stealthier techniques for identifying potential targets, including the older, but effective, techniques of the 1988 worm. For this reason, we also will give an overview of proposed techniques for detecting slow-moving or stealthy worms.

An active scanning worm generates unusual network activity, which can be observed using many possible data sources. One attractive data source is ICMP Destination Unreachable (also known as ICMP Type 3 or ICMP-T3) messages. When a source machine attempts to contact an unreachable or nonexistent target machine, the last Internet router on the path, if configured to do so, will send an ICMP-T3 message to the source machine. Scanning worms, through the process of probing randomly selected IP addresses, will attempt to contact many unreachable machines, and will produce a unique pattern of ICMP-T3 messages. As a worm spreads and infects more machines, more unique source addresses will be attempting to contact the same (or related) ports on unreachable machines. Observing such an increase in scanning activity is a reliable, and early, indicator of worm activity. Using this principle, we have implemented a system called DIB:S in which instrumented routers send copies of their generated ICMP-T3 messages to a central collection station. The collector sorts the ICMP T-3 messages according to the source and destination IP address and ports, and whenever one IP address is seen to be scanning a significant number of target addresses, the collector sends a scan alert to a tracking system called TRAFEN. TRAFEN assembles “tracks” of related scanning activity, based on the time and target port of the scan, and once a track is long enough, TRAFEN takes the track as evidence of an active Internet worm. As we will see, TRAFEN can detect worms before they spread to too many vulnerable machines, opening up the possibility of an effective, early response.

By collecting raw ICMP-T3 messages, the DIB:S system can see scanning activity that spans multiple target networks. Even if a worm instance probes any single network only a few times, DIB:S still will detect the scan as long as there are enough instrumented routers distributed throughout the Internet. The number of instrumented routers can be a modest fraction of the total, making the DIB:S system practical for Internet-wide deployment. In addition, many other unique scanning patterns besides worm propagation can be extracted from the ICMP-T3 data, making DIB:S more extensible and powerful than a system that collects only higher-level scan alerts. Finally, ICMP-T3 messages are relatively compact, and reveal little information about the target network, particularly since the instrumented routers can be configured to send the ICMP-T3 message only to the collection point, rather than to the source machine also. System administrators should have less concern about sharing these messages with a third party than they might with other data sources, although other data sources, as long as they provide insight into scanning activity, could be used within the DIB:S and TRAFEN framework as well.

In the rest of this chapter, we present background on Internet worms and a model for their propagation, describe the architecture of the DIB:S and TRAFEN system, examine simulation results that illustrate the system’s detection performance, and examine future directions for both worm authors and worm defenders.

2. WORMS AND THEIR PROPAGATION

The first step in detecting an active worm is to understand how active worms propagate, and to develop a general propagation model that can be used as the starting point for detection algorithms. First, we compare active worms with other types of malware, and then we present an epidemic model for worm propagation, building upon the modeling efforts of other worm researchers.

2.1. Worms and Viruses

Over the last several years, there has been frequent discussion of the difference between viruses and worms. In the early days of the 1988 worms, Eichin and all⁴ referred to this new event as an “Internet virus”, stating that it bore no resemblance to the biological equivalent of a worm. Today, however, most experts refer to it as the “Morris worm”, indicating that biological equivalence no longer dictates the terminology. Figure 1 is an inheritance graph showing current, commonly accepted relationships in terminology. Viruses and worms are both part of the larger category of malicious code. A related member of the malicious-code group is rootkits and backdoors, pieces of software often installed on compromised systems by hackers to enable them to easily regain control of the machine in the future. Rootkits are associated with the so-called “auto-rooters”, pieces of software that offer a nice GUI to the hacker, making computer intrusions child’s play. A disturbing detail is that many of these tools can perform multiple attacks (exploits) with various target selection strategies, eliminating the need for any understanding from the hacker. The tools often are easier to use than most security products.

Another related member of the malicious-code family is spyware, software that ships and installs with bona fide programs and relays information from the user’s computer back to a data center without the user’s explicit consent. This implies implies that the user often is not aware that spyware programs are present on the system, increasing the risk that private, or even privileged, information might be stolen. Spyware is gaining more attention lately, largely because software packages are increasing in size and complexity, making detection of spyware much more difficult. In addition, spyware programs tend to remain on the system even when the program to which it was originally attached is removed.

Where other malicious code is intended for controlled use, viruses and worms are designed to propagate without control. This makes them very dangerous, since there are no bounds on their spread and their working is fully decentralized. Where rootkits and backdoors provide the hacker with full control of a system, worms and viruses need to be fully autonomous, following the same algorithm over and over again for each newly infected system. There is no reason, however, why the two cannot be combined, creating a massively (self-)propagating piece of malware that leaves backdoors for the hacker to enter all the infected systems at will. Regarding terminology, worms and viruses can be viewed as separate types of autonomous malware (as we prefer and depict in Figure 1, or viruses can be viewed as a broad category of which worms are a special case. Whether worms are their own category or a subcategory has little effect on the discussion of their properties, so we leave it to the reader to form their own opinion.

The difference between worms and viruses lies in their method of propagation. In short, viruses require carriers, where worms facilitate their own propagation. Worms often use an attack strategy that actively selects targets and opens connections to those targets. The worm then launches the exploit, and, if successful, propagates by copying its code to the new system and then running that code. The new system now is infected and will behave the same as the system that infected it, resulting in two copies of the worm, both looking for new systems to infect. This spread continues until most vulnerable systems are infected, or until a built-in timer stops the propagation and switches the worm to another mode, such as a massive Distributed Denial of Service (DDOS) attack using all the infected systems as drones.

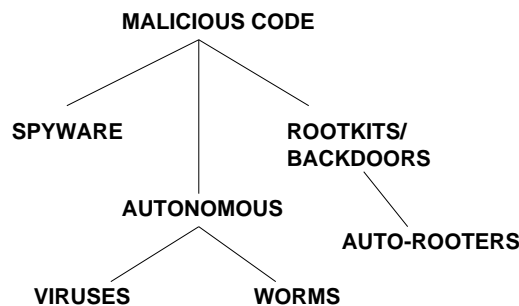


Figure 1. A partial hierarchy of malicious code (or malware).

In contrast to worms, viruses need a carrier to propagate. Traditionally, viruses bind to executable files, the system boot sector, or both. This ensures that the virus is loaded into memory at boot time, or whenever a program is loaded. Once actively in memory, the virus binds to the operating system and tries to infect the boot sector and every program that is run. This will guarantee its spread, since infected executables that are run on clean systems will infect the boot-sector of that system, leading to subsequent infection of that system's other programs as well. This technique requires executable files to be shared between computers, imposing a natural limit on how fast the virus can spread. Recently, however, viruses have been designed to piggy-back on bona fide communication mechanisms such as email. Email viruses often rely on the recipient to open the email and run the attached viral executable, which, in turn, will attempt to send itself to all e-mail address in the user's address book. This is the reason that such viruses often come from your best friend. Virus writers use many techniques to hide the actual virus from the user, such as embedding the viral code inside a screensaver or game. A more sophisticated approach is to include a macro in the e-mail that will run the viral code as soon as the e-mail is opened (without the user having to open the viral attachment itself). This approach, however, requires an email client that understands and automatically interprets and runs such macros. Email viruses, with or without automatic execution of the viral attachment, show propagation patterns very similar to those of active worms.

2.2. Worm Spread

The propagation pattern and autonomous behavior that classifies worms leads to a clearly identifiable algorithm:

- Target Selection
- Infection Attempt
- Code propagation (when the infection attempt succeeds)

Intuitively, the faster a worm can identify and infect new vulnerable targets, the faster it can propagate. This is important, since historically it seems that slow and “silent” worms do significantly worse than fast and “loud” worms, in terms of the peak number of infected systems. The major reason for the success of fast worms is the minimal response time that they provide to take appropriate countermeasures. The response time is mainly based on human factors, since it usually involves system administrators learning about new worm events, and then identifying and patching or removing any vulnerable systems in their networks. Given the limits of human response time, the initial propagation of a new worm can proceed unobstructed, giving fast worms the chance to reach a “critical mass”, namely, infect enough systems to create and sustain an epidemic. In the next section, we will back these intuitive explanations with some basic epidemiology.

The target-selection algorithm is crucial to the success of a worm, and worm authors have shown stunning creativity in this part. Below we give a list of the many different target selection options that have been observed or proposed. Many worms have combined these techniques with varying results. The most common, and easily implemented, algorithm is random generation of target IP addresses. This method has gained popularity on the IPv4 Internet, since the IPv4 Internet is densely populated. Selecting a random IP address has a high chance (between 5% to 15%) of hitting an existing machine. On the IPv6 Internet, random IP generation technique is far less effective due to the enormous address space. When all the machines currently on the IPv4 Internet migrate to the IPv6 Internet, the IPv6 Internet would be extremely sparse, and it would take *years* for a random-generation algorithm to find an IP address actually associated with a host.

- Random
 1. Directed
 2. Hitlist
- Sniffing

- Name
 1. Email addresses
 2. System files
- Combination of above

To improve the chance of finding vulnerable machines, many worm authors employ techniques in which they direct the random target selection. By preferring address ranges that are densely populated or address ranges that are suspected to contain a large number of *vulnerable* machines, the worm can propagate significantly faster. As an example of the latter case, the vulnerability that the worm exploits might be typical of home computers. The worm author would attempt to identify up front which target ranges hold the most home computers (dial-up and cable-modem ISPs) and then program the worm to prefer targets in those address ranges. Alternatively, the worm can be programmed to select targets only from a list of *known* targets. This approach usually is called “hitlist propagation”, and is most effectively used as an initial propagation method before defaulting to random propagation.⁵ Such a hitlist would contain IP addresses that are known to be vulnerable systems, and thus would need to be constructed before the worm was released. Construction of hitlists can be done slowly over the course of months by randomly scanning the Internet. To avoid attacking the same system multiple times, the list is split in half every time a worm instance propagates. One half is kept by the infecting system, while the other half is given to the newly infected system. Hitlists are an effective way of establishing a critical mass of infected systems. A further optimization is permutation scanning in which every worm copy scans according to the same reproducible random sequence. Each copy of the worm starts at a position in the sequence determined by the IP address of the local host, and switches to a new random position whenever it encounters a machine that already is infected.⁵ Switching to a new position takes into account the fact that if a host is already infected, some other copy of the worm already is working its way through that portion of the sequence. The permutation approach, which has not been employed in actual worms yet, preserves the simplicity of random scanning, while minimizing duplicative scanning effort.

Scanning activity can be difficult to hide, since intrusion-detection and traffic-monitoring systems can notice the pattern of one machine actively connecting to many other machines. A technique that has been frequently discussed, although not used in implemented worms yet, is passively sniffing the network (or inspecting application-level traffic) to identify reachable IP addresses that likely are running a service that the worm can exploit. As an example, a contagion worm might have two exploits, one for Web clients and one for Web servers..⁵ A copy of the worm on a Web server attempts to infect any Web client that requests a page, while a copy of the worm on a Web client attempts to infect any Web server to which the client connects. Fortunately, this approach is applicable only for some services, since the worm must see enough traffic to build up a reasonably sized set of potential targets. For example, if the worm only had an exploit for Web servers and was passively sniffing the network to identify other Web servers, it might see little or no traffic for any Web server other than the one already infected, particularly given the prevalence of switched Ethernet. On the other hand, a worm exploiting a vulnerability in email servers will have a better chance of succeeding, since email servers contact *each other* to exchange email. As long as users on the local network make moderate to heavy use of email, the worm will be able to identify a significant number of email servers that it can attempt to infect. As an added bonus for the worm author, such an email worm would be equally successful in densely or sparsely populated address spaces.

When the address space is only sparsely populated, random scanning (even to construct a hitlist) can be an impossible task, and thus other methods need to be employed. In addition to the passive network sniffing discussed above, a worm can use DNS names rather than IP addresses to identify systems. When a top-level domain name is acquired, DNS servers often will reveal the names of the associated mail exchange server and Web server. Even if these names are not obtainable directly from the DNS system, the worm author can make an educated guess as to what the names of existing systems would be. Imagine that the worm acquired the domain *exampledomain.com*. A logical naming scheme would suggest that *www.exampledomain.com* would be the Web server and *mail.exampledomain.com* would be the mailserver. A list of other names would include *www1*, *ns*, *ns1*, *dns*, *dns1*, *nameserver*, *ftp*, *smtp*, *pop3* or *skywalker*. Names from Greek mythology also are also very popular.

The worm author's creativity can be endless, and techniques that have been used for many years in password crackers also can be used to construct hostnames. If a site has a hostname *sparc09*, for example, it is worth trying *sparc01*, *sparc02*, ... *sparc99* as well. Additionally, hostnames can be gleaned from many other sources. The 1988 worm⁴ used the *.rhosts* file to obtain hostnames of other systems in the network. Similarly, most operating systems hold small name database files as a backup for when the DNS system fails. Other sources can be email addresses, which have the basic structure *username@domainname*, and provide domain names for the process above. Obtaining the addresses or names of potential targets from information stored on the currently infected machine often is called topological scanning.⁵ Although most worms today use some form of random target selection, the introduction of IPv6 means that it is no longer the guaranteed fastest way to propagate. Future worms most likely will employ combinations of the above techniques to facilitate their propagation. In addition, viruses that use normal network traffic as a carrier will become increasingly popular, since they do not need to select their own targets.

After a target is selected, the worm will attempt to infect it. If successful, the worm will run a copy of itself on the newly compromised system. There are two general approaches to this code propagation:

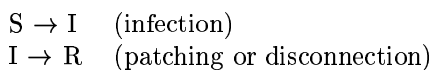
- Central Repository
- Cloning

The central code repository stores the code of the worm, and each time a new host is infected, the worm is downloaded from this location. The benefits of this approach are that the worm author can update the worm code while the worm is propagating. It is even possible to store commands that each instance of the worm will download and execute, leaving all the infected systems open for control by the worm author. The drawback is obvious – the security community easily can take the central repository off-line, effectively disabling the worm and stopping further propagation. The second approach (cloning) does not allow for code updates, but also makes it much harder to stop the worm during propagation. The code of the running worm is copied and started on each newly infected machine, effectively cloning the current copy. Although not allowing any control mechanisms, these types of worms tend to be more successful. Evolutions of the central-repository technique, or programming worm copies to create their own peer-to-peer network for command distribution, will provide significant control capabilities for hard-to-stop worms, however.⁵

2.3. Epidemics

To get a feel for the factors that govern worm (as well as virus) propagation, most researchers take to the classic epidemiological equations. These models describe biological epidemics quite well, and have proven to be very applicable to their cyber equivalents. We will introduce these models here and refer to further reading for a more in-depth coverage of the topic.

In its most basic form, the behavior of a single host is described by the SIR (Susceptible-Infective-Recovered) model as shown in Figure 2. For a given worm, the *S*-state (susceptible) indicates the host is vulnerable to that worm. The *I*-state (infective) indicates that the host is infected and spreading the worm. The *R*-state (recovered/removed) means that the host is not (or no longer) of interest to the epidemic. The reasons for being in the *R*-state may vary, most often the host simply was not vulnerable to the worm in the first place, or the host was patched (whether infected or not). Alternatively, the host might be disconnected from the network, either to prevent infection or further propagation. For any worm, only a marginal portion of all the hosts are vulnerable, i.e., in the group of susceptibles *S*. The majority of Internet-connected hosts will be in the *R*-group, and not be involved in the spread of the epidemic. The transitions between the states are given below, keeping in mind that the transitions apply to the state of a host for one particular infection only:



And furthermore:

- S → R (uninfected system patched)
- I → S (infection removed, but system not patched)
- R → S (susceptible system reconnected to the network)
- R → I (infected system reconnected to the network)

The first two transitions are the most common case, and account for the majority of the total number of state transitions made during an epidemic. They model the infection of vulnerable systems (S→I transition), and the patching or removal of infected systems (S→R transition). Although many systems generally are not vulnerable to a certain worm attack, they do not change state and largely remain in their *R*-group. The classic epidemic equations from Kermack and McKendrick focus on these two transitions (see Daley and Gani⁶) for modeling the spread of an infection in continuous time. The population *N* is constructed from the three groups *S*, *I*, and *R*, which change over time as defined by *s(t)*, *i(t)*, and *r(t)*, where *t*₀ is the time at which the infected begins. Note that *N* = *s(t)* + *i(t)* + *r(t)*, meaning that the population size is assumed constant, which is acceptable considering that we defined *R* to contain disconnected, not just patched, systems. The population changes over time can be defined as

$$\frac{ds}{dt} = -\beta si \tag{1}$$

$$\frac{di}{dt} = \beta si - \gamma i \tag{2}$$

$$\frac{dr}{dt} = \gamma i \tag{3}$$

The parameter β models the transition S→I and γ models the transition I→R. Intuitively, β is the likelihood of one particular infected system contacting (and infecting) one particular susceptible system in *dt* time. Likewise, γ is the likelihood that one particular infected system is patched or disconnected in *dt* time. Putting a number to these factors is not easy since it is different for each worm. The general principals discussed in the previous section, however, lead to some guidelines. First of all, the rate at which a worm can infect new systems is limited by the rate at which it can contact other systems (which determines β). This rate is either limited by parallelism or by bandwidth, whichever reaches its limit first, and which are determined by the capabilities of the infected host and the target selection algorithm of the worm. The most effective propagation would be when the worm uses up all the bandwidth that the host has to offer. Thus, the closer a worm can approach this limit, the better its chances are on fast propagation. There are several factors involved that make this easier or harder. The first factor is the protocol that the worm uses to propagate. When the worm uses a fire and forget protocol (like UDP), it most easily can use all of the bandwidth since it never has to wait for a return packet. When a connection oriented protocol (such as TCP) is used, however, the worm will need to wait for an acknowledgment from the target host before it can send the attack data. The choice is not always up to the worm author since most services (and hence most vulnerabilities) are built using connection-oriented protocols.

The latency between initiation and acknowledgment, however, can be filled with connection requests to other potential targets when the worm interlaces them properly. With appropriate programming, which may include the worm generating its own connection requests and bypassing the operating system's network stack, the worm can hide most of the latency associated with connection-oriented protocols. For example, one thread in the

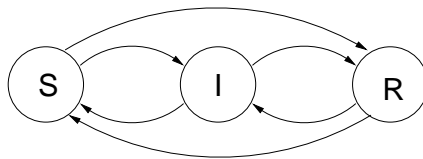


Figure 2. The SIR (Susceptibles-Infectives-Recovered) model is probably the most popular way of identifying the groups in an epidemic, and its transitions form the basis for a broad range of mathematical models.

worm would craft requests packets, transmit those packets, and log the outstanding connection in a table, while a second thread constantly would check (sniff) for return packets and attempt to match them with the entries in the table. Every several seconds the worm traverses the entire table to fault connection requests that have not had a response within a *worm-defined* timeout period. By making this table sufficiently large, the worm should be able to fill the available bandwidth without needing to run thousands of concurrent copies of itself on the infected host. Although such an approach makes the worm more complex and more difficult to implement correctly, the added burden might be well worth the increased propagation speed. In addition, matters become easier when the target-selection algorithm has some predetermined knowledge of reachability or even vulnerability, such as in the case of a pre-constructed hitlist. The targets on this list are mostly reachable and likely also vulnerable. This will significantly decrease the time spent in timeouts, meaning that more network bandwidth will be used since each connection attempt will take less time on average.

Given this discussion, the average time that each connection takes could be calculated as

$$\tau = r \times t_{latency} + (1 - r) \times t_{timeout} \quad (4)$$

where r is the reachability based on the target-selection algorithm. A perfect hitlist would give $r = 1$, and random target selection would give $r \approx 0.1$.

When a worm does use a hitlist for initial propagation, the worm would have two different values for β , one value for the hitlist part of the propagation, and a second smaller value for the remaining (random) part of the propagation. In addition, I_0 (the initial number of infected systems) for the second part would be the number of infected systems after the hitlist propagation is complete. For completely random target selection, β can be defined as

$$\beta = \frac{1}{N} \times \frac{\alpha}{\tau} \quad (5)$$

where N is the size of the address space (2^{32} in case of IPv4) and α is the number of concurrent scanning threads. In the case of a worm that implemented a fully parallel scan through the construction of its own request packets, *alpha* might be defined quite high (even if the worm itself only used the two threads described above). In the equations, dt is the same as the unit of τ , meaning that if τ is calculated in seconds, dt in Equations 1 also is in seconds. For a perfect hitlist (where every IP address is indeed a susceptible host), we instead could define β as:

$$\beta = \frac{1}{S_0} \times \frac{\alpha}{\tau} \quad (6)$$

where S_0 is the number of systems that are initially susceptible (assuming that the hitlist holds *all* susceptible systems). The second factor $\frac{\alpha}{\tau}$ essentially calculates the average number of successful connections a single infected host can complete in dt time (not all of those are necessarily susceptibles). When network bandwidth is the limiting factor, rather than worm parallelism, the second factor can be replaced with a division of the available network bandwidth by the size of the infection packetstream.

The γ parameter (the I→R transition) can be harder to model since it mostly depends on actions of the system administrator. It will take security personnel some time to discover a newly launched worm, and then they will need to analyze the worm and possibly write a patch. System administrators then must learn about, download, and install the new patch. Another option for system administrators is the disconnection of infected machines from the network. Both processes, patching and disconnection, are hard to model, and likely are not governed by a fixed rate. Note that in the Kermack and McKendrick model, the transition is dependent only on the current size of the group of infected systems, which could be too simple a dependency to model the behavior of security personnel and system administrators.

Additional Transitions. The other transitions in the SIR model are interesting for further study. The S→R transition models uninfected systems that are vulnerable to the worm under consideration, but get patched or disconnected before they are infected. Although this process might be underway before the worm is launched (i.e., a patch for the worm's exploit is available a priori), only its effect on the worm should be modeled. Patching that occurs before the worm is released simply decreases S_0 . Below is an extended set of differential equations taking into account all six transitions from the graph:

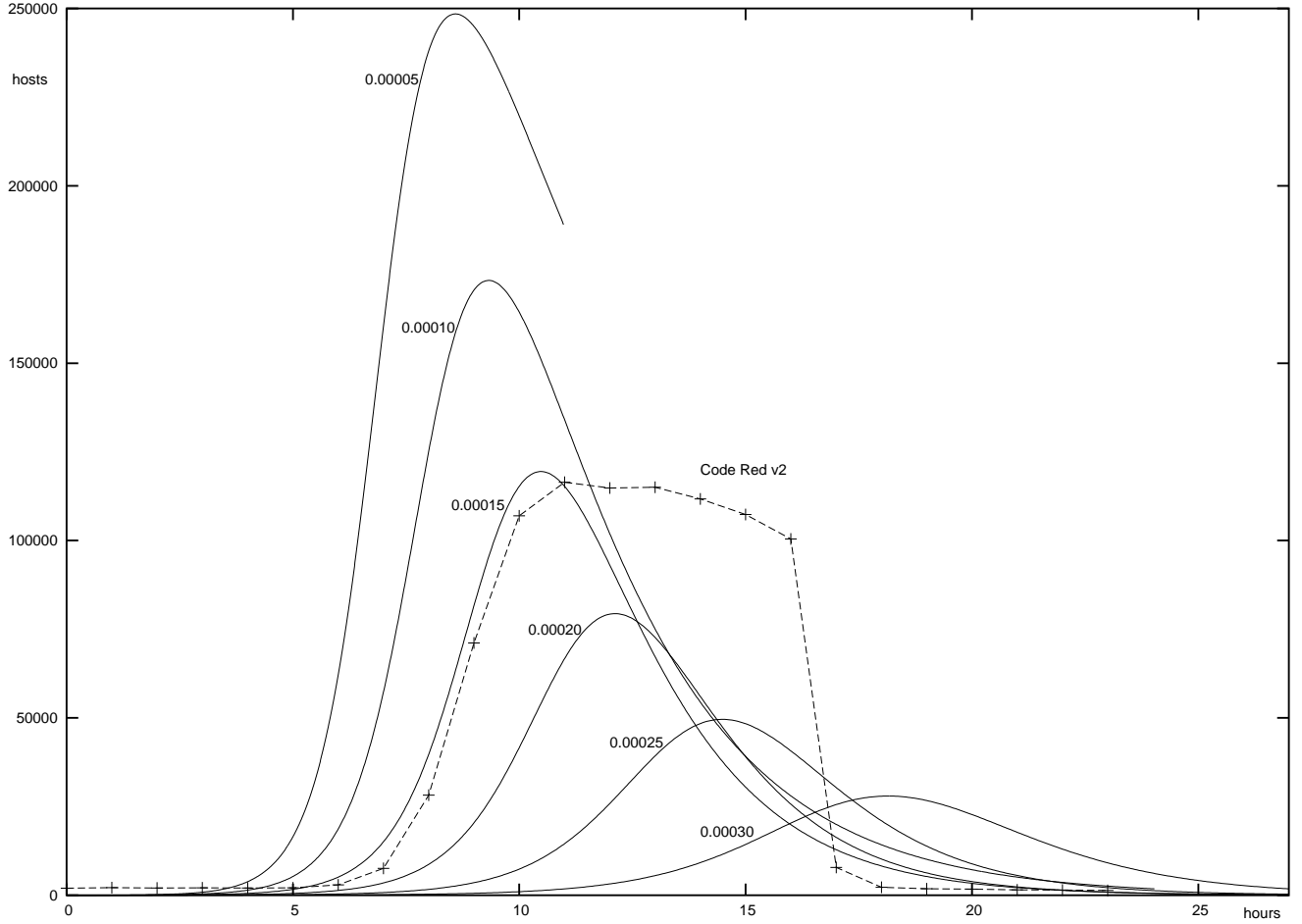


Figure 3. Spread of Code Red v2 versus the epidemic equations for different values of γ . The vertical axis represents the total number of infected systems at any given time, and the horizontal axis is the time in hours. Parameter β was calculated based on Equations 4 and 5 and the characteristics of Code Red v2 on a per-second basis: Code Red v2 used 100 concurrent scanning threads ($\alpha = 100$), with an average reachability of $r = \frac{1}{10}$ and a default timeout on no response of 21 seconds (based on the default Windows NT timeout; exponential back-off with 3 retries after 3, 6, and 12 seconds). This gives (4) $\tau = \frac{1}{10} \times 1 + (1 - \frac{1}{10}) \times 21 = 19$. The address space was IPv4, and thus $N = 2^{32}$ gives (5) $\beta = \frac{1}{2^{32}} \times \frac{100}{19} = 1.23 \times 10^{-9}$. Notice how the total number of systems infected (surface area under the graphs) decreases with higher values for γ . Code Red v2 data was collected at TRIUMF Canada (<http://www.triumf.ca>), which generously made the data available to us for this research.

$$\frac{ds}{dt} = -\beta si - \eta i + \zeta r + \theta i \quad (7)$$

$$\frac{di}{dt} = \beta si - \gamma i + \varepsilon r - \theta i \quad (8)$$

$$\frac{dr}{dt} = \gamma i - \varepsilon r - \zeta r + \eta i \quad (9)$$

The $S \rightarrow R$ transition is governed by the parameter η and is taken to be dependent on the size of the group of infectives over time. This is parallel to the $I \rightarrow R$ transition, and indicates that administrators will patch

uninfected systems, as well as infected ones, with greater urge as the worm propagates. It can be argued, however, that it should be multiplied by the size of the group of susceptibles as well, since the chance that administrators patch or disconnect uninfected systems decreases as there are fewer systems uninfected. The I→S transition (represented by θ) also is taken to increase and decrease as the group of infected systems grows and shrinks. This means that the larger the group of infected systems, the greater the number of systems that will be cleaned, but not patched. A good example of this was the propagation of the Code Red v2 worm. Although Code Red v2 could be removed from a system by rebooting, the system would be susceptible to re-infection after the reboot. The R→S transition (ζ in the equations) is most likely due to uninfected, yet susceptible, systems being taken off-line and then reconnected to the network later. It also could indicate systems that were patched and updated, but became vulnerable again for various reasons. In nearly all cases, this will be a small fraction, and is mostly dependent on how many systems are disconnected or patched. Similarly, the R→I transition (modeled by ε) will be small, representing the infected systems that are taken off-line and later reconnected, allowing them to continue spreading the infection. One final note on these four transitions is their relative insignificance compared to β and γ . Even for very small values of θ , ε and ζ , the equations can be unrealistically imbalanced. The interested reader is encouraged to try different values for all parameters and see how the epidemic curve behaves.

3. RESPONSE

The best way to respond to an epidemic is to prevent it in the first place. History has shown, however, that there have always been unpatched vulnerabilities. Moreover, with software getting more and more complex, it is unlikely that this will change. Software vendors put significant effort into distributing patches to mend security holes in their software, but not nearly enough users install such patches promptly. Often people are not aware of security updates, and many others get tired of the continuous stream of updates, inadvertently leading to disregard. Patching does decrease the size of the pool of susceptibles, however, effectively limiting the damage any worm can do. The most obvious solution would seem to be automated patching services, although the necessary basis of trust is lacking. No security expert would trust *another* piece of software to secure the system, arguing that such a service would itself be a target for attack. The scenario is clear; once the security service is compromised, specifically the central server from where the patches actually come, the attackers can distribute a patch that installs a vulnerability that later can be exploited in a massive worm attack. It would even be possible to distribute the initial copies of the worm through such a service and use it as the initial group of infectives, creating a very broad critical mass. Needless to say, such a situation would be devastating.

Thus, although automated patching does have its place, automated response after the worm is launched must be a critical part of an effective defense. When we consider the epidemic equations, the two parameters that govern the majority of all transitions are β and γ . An epidemic can be reduced by either lowering β or increasing γ . Figure 3 shows how increasing the value of γ reduces the number of hosts effected by the epidemic (surface area under the graph). We now will discuss several ways of influencing these parameters as a form of active response to worms.

Increasing γ . A common way of avoiding communication with infected systems is the ‘blacklist’. This is a technique often used within the security community to filter out IP addresses that have shown aggressive behavior in the recent past. A similar technique could be used to collect IP addresses of systems that are known infectives. This list would grow as the worms propagate. Routers across the world would have to implement filtering rules to disallow traffic from any of these IP addresses. This effectively cuts infected systems off the network by blocking them from communicating, therefore increasing γ . The R-group will increase, and there will be relatively more disconnected, infected systems than normal. Problems with this approach are the implementation requirements. Moore and all⁷ conclude that practically all of the Internets major connections need to employ blacklist filters for this technique to be effective. In addition, the list of blocked IP addresses needs to be continually updated and, as the list grows, it will incur a significant load on all the participating routers. Additionally, a fast and accurate detection system needs to be in place to determine which systems should be added to the blacklist.

A second, similar technique is automated firewall configuration. This is based on the same idea, but it requires an even larger participation. When an IP address is determined to be infected, the firewall (or firewalls) closest

to that system are notified and will block out traffic from that specific host. Additionally, all non-essential traffic on the port targeted by the worm should be filtered at every firewall. This will effectively disconnect the infected system from the network, again increasing γ . A common problem that both these techniques share is the ability for attackers to perform a DOS attack on arbitrary hosts or networks. Attackers can spoof malicious traffic, making it seem like it came from a particular network, and get the worm response system to blacklist or filter out all traffic from that network, effectively disconnecting it from the Internet.

Reducing β . Since β governs the growth of the worm, worm authors will try to maximize β , and the security community, in turn, must try to minimize it. A technique that has been discussed by Williamson⁸ is to reduce the number of new connections that a host may initiate per timeslice. A connection is counted as new when it connects to an IP address that it was not communicating with in the recent past. Known IP addresses (i.e., those with which a machine communicates often, such as mail or DNS servers) are stored in a list of a given size and will never incur a delay. For the unknown IP addresses, however, the connection limit is imposed. This will limit the number of probes a worm can do per second. As more and more new IP addresses are contacted, those requests end up in a delay queue that will grow larger and larger. Under normal working load, this queue will occasionally grow and incur some delay, but never at the rate it will grow when a fast-scanning worm is present. A reasonable limit of 10 new connections per second, however, would not have delayed Code Red v2, since it did not effectively initiate more than 5.2 new connections per second. (This number is based on the calculation of the average time for randomly probing one IP address $\tau = 19$ (4), combined with $\alpha = 100$ concurrent scanning threads in Code Red v2, gives an average of $\frac{100}{19} = 5.2$ completed scans per second.) This means that this technique would only disrupt the fastest of worms, likely providing ample response time for other detection and active response mechanisms (none of which are in place at this time). An additional argument for implementing this method is the minimal overhead it puts on the system. Rate limiting does put a limit to β because the speed which a worm can scan for targets is the prime factor determining β . Some server systems, however, would suffer badly from this method, since they usually have more active outbound connections. Consider, for example, DNS or email servers, both of which will connect to many other systems based on the name queries or email messages sent by the users. A similar difficulty is encountered on multi-user systems, where multiple users are logged on at the same time. This technique works better on “static” servers like web servers that mainly listen for incoming connections. Additionally, it may be possible for a worm to circumvent the rate-limiting mechanisms by crafting packets instead of traversing the TCP/IP stack.

A second technique is “traffic content filtering”. It is based on the idea that routers and/or firewalls will test all traffic flowing through against a set of known, viral signatures. When a malicious signature is detected, the packet is dropped, effectively limiting the propagation of malicious code and decreasing β . The technique, however, requires very elaborate signatures and matching on port/protocol combinations, since the sheer volume of traffic traveling through large routers creates a fair possibility that smaller signatures would be matched in regular, bona fide traffic. As Moore and all⁷ discuss, for application during a new worm event, this approach requires the signature to be generated as early as possible. Signature-capable routers would need to be in widespread use, as well as a mechanism to quickly and securely distribute new signatures. Once again, this defense system allows for a DOS attack when an attacker is able to insert a falsified signature that would block all traffic for a particular service. In addition, this system would put a tremendous overhead on critical network routers on the Internet, since signature matching (especially when the pool of signatures is large) is very processor intensive. Combined with the need to re-assemble each fragmented packet, to avoid overlooking fragmented attacks, this cure might be difficult to deploy widely.

Conclusion. The general mantra for this section is the need for very early detection of new worm events. Whatever the response will be, it will never be useful if the alert and classification come too late. Considering that the Sapphire/Slammer worm^{3,9} propagated in just several minutes, it is clearly not a humanly possible job to generate the alerts. Although automated alert and response systems would be up to the task, they are at the risk of becoming the target themselves, potentially being more dangerous than any regular worm could ever be. It seems, therefore, that there will always remain a delicate balance between human interaction and machine automation. We can envision a system in which the monitoring and detection is done automatically, such that alerts and signatures are generated for a human first responder to assess. Next the human responder

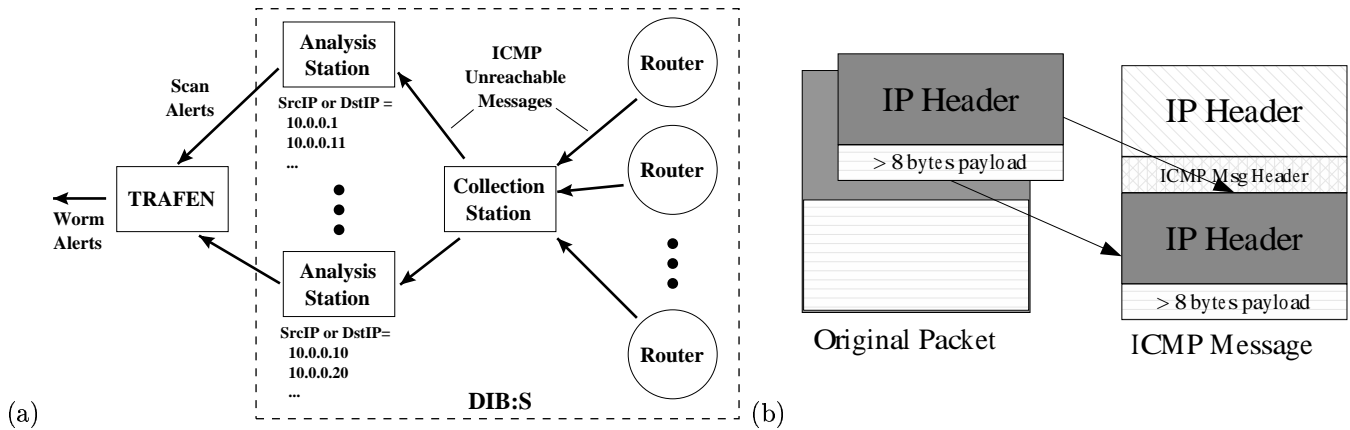


Figure 4. (a) The combined TRAFEN and DIB:S system. ICMP Unreachable messages are sent to the collection point, which divides the messages among the analysis stations (so that messages with the same source or destination IP address go to the same analysis station). The analysis stations identify scanning activity, and send scan alerts to TRAFEN, which identifies increases in scanning activity characteristic of propagating worms. (b) The IP header, plus at least eight bytes of the protocol data, of the packet that provoked the generation of an ICMP-T3 message becomes the payload of that ICMP-T3 message.

can decide which (if any at all) of the active response mechanisms to activate, allowing an appropriate response to the event.

4. EARLY DETECTION OF SCANNING WORMS

Our prototype system for detecting scanning worms collects ICMP Destination Unreachable (or ICMP T-3) messages from instrumented routers, aggregates these messages to identify scanning activity, and then looks for patterns of scanning activity that indicate a propagating worm. The system, whose architecture is shown in Figure 4a, has two major components, the Dartmouth ICMP BCC: System or DIB:S, which aggregates the ICMP T-3 messages into scans alerts, and our Tracking and Fusion Engine or TRAFEN, which identifies propagating worms based on their scanning activity. Within a Multiple Hypothesis Tracking¹⁰ (MHT) framework, TRAFEN assigns likelihoods to sets of alerts or observations that appear to be correlated, thus forming tracks of related observations. By defining the likelihood functions so that observations are highly correlated only if they appear to represent worm activity, TRAFEN can quickly and accurately detect an active worm. The prototype DIB:S and TRAFEN system focuses on the detection of fast-moving worms that attempt to infect the vulnerable population within minutes or hours, since automatic detection of such worms is the only way to provide enough early warning to take appropriate countermeasures. For slower-moving worms, manual detection of the worm, in time to warn system administrators and other personnel, is much more likely. As we will see, however, the system will allow the detection of slow-moving worms through straightforward extensions. The system will not detect worms that do not probe randomly generated IP addresses, however, such as worms that use a pre-constructed hit-list or that monitor network traffic to identify existent, reachable machines. Detection of these “stealthy” worms will be discussed in a later section. In this section, we present background on ICMP T-3 messages, describe the DIB:S and TRAFEN components, and examine the detection capabilities of the overall prototype.

4.1. ICMP T-3 Messages and Instrumented Routers

When a source machine attempts to contact a nonexistent or unreachable machine, an Internet router, somewhere between the source machine and the target network, will determine that the packets can go no farther. This router, if configured to do so, will send an ICMP T-3 message to the source machine. Scanning worms, through the process of probing randomly selected IP addresses, will attempt to contact many unreachable or nonexistent machines, such as machines protected by a firewall or addresses from an unassigned part of the Internet address space. If this scanning activity produces enough ICMP T-3 messages, we can infer the presence of a propagating

	PING		PING		TCP/80		Total	
	24.[0-128]/16		[209-211].[32-64]/16		[209-211].[32-64]/16			
Requests sent	1628977	100%	6487973	100%	1171298	100%	9288348	100%
no response	1258388	77.3%	4911425	75.7%	800636	68.4%	6964182	75.0%
Echo replies	244445	15.0%	636135	9.8%	37707	3.2%	918287	9.9%
ICMP Unreachable	77361	4.7%	398841	6.0%	104555	8.9%	571857	6.2%
Other	48783	3.0%	550472	8.5%	228400	19.5%	827655	8.9%

Table 1. Responses to random probing on the Internet - ICMP echo request on the 24.0/16 - 24.128/16 networks. ICMP echo and TCP port 80 request on the 209.32-64/16 - 211.32-64/16 networks

worm through its unique scanning pattern, specifically, the growth in scanning activity as the worm infects more and more machines.

Table 1 shows the responses we received when we probed selected address ranges on the Internet. The data, which was obtained for a separate project, is skewed slightly, since only populated address ranges were scanned. Many address ranges simply are unassigned, and contain no reachable machines at all. The two most significant numbers are the high response rate (25%) and the number of ICMP-T3 messages returned (6.2%). The latter number, although seemingly low, means that a significant fraction of scan attempts will produce an ICMP T-3 message at some router. Thus, if we can collect and analyze ICMP T-3 messages from multiple, distributed routers, we will have enough messages to detect a worm’s unique scanning activity.

Due to privacy concerns, we have chosen *not* to sniff for ICMP T-3 messages, but instead to ask network providers and other organizations to forward the ICMP T-3 messages from their routers to our analysis systems. These forwarded messages are essentially a Blind Carbon Copy (BCC) of the original ICMP-T3 message, which is a legitimate action since the generating router was a participant in the original conversation. Although site policy may require that no response be sent to the source machine, the router can remain silent to the outside world while still sending the ICMP-T3 messages the analysis systems. A worm would not get a response to a specific probe, but the analysis systems still would see that the probe has occurred.[†] Although only 6.2% (Table 1) of random probes returned an ICMP-T3 message, 75% returned no response. This silent fraction may include many routers that have been instructed to silently ignore unsolicited traffic. These routers could easily forward ICMP-T3s to the analysis systems, while still dropping the original packet without a response to the sender. This would allow broader coverage, while still respecting the security policies of individual organizations.

Standard routers, of course, do not produce Blind Carbon Copies of their ICMP T-3 messages, so the router operating system must be extended to provide this functionality. To facilitate participation in our worm-detection system, we currently provide a patch for the popular Linux kernel that adds the forwarding capability to Linux-based routers, and we are working with CISCO on a CISCO IOS patch. The patched or instrumented routers, which can be located anywhere in the world, act as sensors, forwarding their ICMP T-3 messages to the analysis systems. As we will see, we do not need all, or even a sizable minority of, Internet routers to forward their ICMP T-3 messages. We can achieve effective worm detection with only modest participation from Internet service providers and other organizations, making the problem of participation tractable, albeit still administratively complex.

ICMP-T3 messages come in several different flavors,¹² two of which are of particular interest for detecting scanning activity: Network Unreachable (Code 0) and Host Unreachable (Code 1). A router generates a Network Unreachable message when a desired network cannot be reached. This might happen when a packet is sent to an IP address that resides in an unassigned portion of the Internet address space. Far more commonly, a router generates a Host Unreachable message when a router cannot find the addressed host in its network. This might happen when the packet could be routed to the correct network, but the router responsible for that network could not locate a machine in its network that bears the requested IP address.

[†]RFC 1812 section 5.2.7.1 states that routers *should be able* to generate ICMP-T3s, not that they *should* generate them.¹¹

The feature that makes analyzing ICMP-T3 messages useful is their message body. When a router builds a Destination Unreachable message, it includes the IP header, and at least the first eight bytes of the body of the *original message* (i.e. the message that provoked the ICMP-T3 response) as the payload of the ICMP-T3 message (Figure 4b). To clarify this, imagine a system that is randomly scanning IP addresses to find Web servers listening on Port 80. From Table 1, we see that this will elicit a significant number of ICMP-T3 responses that would include, as their payload, the IP address of the scanning system, plus at least eight extra bytes from the original payload, which, in this example, would be the beginning of the TCP header. If the analysis systems received a BCC of such an ICMP-T3 packet, the IP address of the scanning system would be known, as well as the port for which it was scanning.[‡] Now, if multiple routers across the Internet forward the ICMP-T3s that they generate to the data analysis systems, it would soon become clear that a host (i.e., an IP address) was scanning the Internet to find Web servers. We refer to this pattern of a scanning host as a “bloom”.

4.2. DIB:S

The primary task of DIB:S is to collect ICMP-T3 data and identify blooms of scanning activity. The instrumented routers, described in the previous section, send carbon copies of their ICMP-T3 messages to one or more collectors, which, in turn, will forward the messages to one or more analyzers. Each analyzer is assigned an IP address range within which it will look for scanning activity, and more analyzers can be spawned dynamically as needed (with appropriate updates to the assigned address ranges). When an ICMP-T3 message arrives at a collector, the collector extracts the embedded content, namely, the IP header and at least eight additional bytes of payload of the packet that provoked the generation of the ICMP-T3 messages. For UDP and TCP packets, the eight additional bytes are the first eight bytes of the TCP or UDP header, which includes the destination port. The collectors and analyzers will use the source and destination addresses and the destination port to aggregate the messages and generate alerts. First, based on the embedded source and destination IP addresses, the collector forwards two copies of the payload to the analyzers. The collector sends one copy to the analyzer for the address range of the embedded source IP address, and sends one copy to the analyzer for the address range of the embedded destination IP address.[§] In this way, an analyzer will see all information about a specific range of IP addresses, regardless of the routers from which the information came. Organizing the analysis by source and destination address, rather than the generating router, is critical, since randomly scanning worm will hit many different networks, and the resulting ICMP-T3 messages will come from many different routers. Thus, the scanning activity is much more visible when viewed across routers, rather than at a single router.

The analyzers keep a history of the ICMP-T3 messages received for a particular IP address over the last Δt seconds. If an analyzer sees more than N ICMP-T3 messages for a single address within the Δt -second window, the analyzer will generate a scan alert, and send that alert to the TRAFEN system. The analyzer will not generate a second alert for an address if it already has generated an alert within the last Δt seconds. As time progresses and the history expires, however, the analyzer will generate a second alert if scanning activity from or against that address continues. We generally keep Δt in the range of five to thirty minutes depending on the number of instrumented routers and the volume of incoming ICMP-T3 messages, and, similarly, keep N in the range of three to ten ICMP-T3 messages per generated scan alert. We will consider these two parameters in more detail in a later section.

DIB:S will generate an alert in four primary cases. The first two cases are symmetrical – in the last Δt seconds, on the same port p and using the same protocol P , one host has contacted N different IP addresses (Case 1), or one host has been contacted by N different IP addresses (Case 2). Similarly, the other two cases also are symmetrical – in the last Δt seconds, on the same port p and using the same protocol P , one IP address has contacted another IP address at least N times (Case 3), or one IP address has been contacted by another IP address at least N times (Case 4). In addition to the four primary cases, DIB:S also can generate alerts in two symmetrical secondary cases – in the last Δt seconds, one IP address has contacted another IP address on at least N different ports (Case 5), or one IP address was contacted by another IP address on at least N different

[‡]The destination port appears within the first 8 bytes of the TCP or UDP header.

[§]Depending on the number of analyzers and the particular source and destination IP addresses, the two copies might go to the same analyzer, in which case only one copy is actually sent.

ports (Case 6). The alerts or observations sent to TRAFEN contain the case number, the embedded source and destination IP address, the protocol, and, if available, the source and destination port numbers.

Case 1 is the most direct and obvious indication of (random) scanning behavior, and also is the clearest case of a “bloom”. It could be a direct result from an active worm. Case 2 would be seen when a public server fails and requests keep arriving. If Case 2 increases over time instead of decreasing, it is most likely the result of a successful denial-of-service (DOS) attack. Cases 3 and 4 are most likely the result of one of two communicating hosts going off-line, and packets from the other system are attempting, but failing, to reach the first host. Cases 5 and 6 could be a sign of one system doing a vertical port scan on another system. It is very unlikely that DIB:S will see these cases, however, since ICMP is rate limited, and routers generally will not generate more than three or four ICMP messages per second. Thus, it takes significant time to see these latter types of observations. In fact, a worm simulation with a steady stream of injected noise showed that 99% of all observations are actually of Type 1.

4.3. TRAFEN

TRAFEN was not implemented specifically for the detection of active worms, but instead is a prototype process query system.¹³ A process query system (PQS) is a software system that allows users to interact with multiple data sources, such as traditional database (DBS) and real-time sensor feeds, in new and powerful ways. In a traditional DBS, users express queries as constraints on the field values of records stored in a database or arriving from a sensor network, as allowed by SQL and its variants for streaming data. In contrast, a PQS allows users to define *processes*, and to make queries against databases and real-time sensor feeds by submitting those process definitions. The PQS parses the process description and performs sequences of queries against the available data sources, searching for evidence that instances of the specified process or processes exist. Depending on the capabilities of the PQS and the problem domain, the process description might be specified as a rulebase, a Kalman filter,¹⁴ a Hidden Markov Model,¹⁵ or any of a number of other representations. A major innovation of the PQS concept is the virtual process-description machine that it presents to the programmer. Such a system abstracts away the details of observation collection, management, and aggregation, and allows the developer to focus on the task of defining and implementing an appropriate process description.

The TRAFEN prototype is implemented in Java and uses Multiple Hypothesis Tracking (MHT)¹⁰ as the core operation of its PQS virtual machine. Process descriptions are expressed as Java classes, which, externally, support a general interface that measures the *likelihood* that two observations are related, and, internally, use any desired model to represent the underlying process that is generating the observations. Although many such classes are specific to a particular problem domain, other such classes implement a general model, and can be applied to many different problem domains through appropriate parameterization. More specifically, TRAFEN provides facilities for (1) subscribing to one or more available eXtensible Markup Language (XML) observation streams, (2) publishing correlated observations to higher-level consumers, (3) dynamically loading the desired Multiple Hypothesis Tracking algorithm (such as Reid’s original algorithm¹⁰), and, most importantly, (4) using that algorithm to maintain a set of hypotheses that explain the observations. Each hypothesis is a set of mutual consistent tracks, where each track is set of correlated observations. The MHT algorithms assign a probability or likelihood to each hypothesis and to each track within a hypothesis. At any given time, the current most likely hypothesis typically is taken as the correct view of the events in the real world.

To apply TRAFEN to a particular problem domain, the developer must define an XML message format for the observations, and must provide concrete implementations of two abstract classes, (1) a class that holds the *state* information for an observation, and (2) a class that measures the *likelihood* that particular observations are correlated in some way (i.e., the likelihood that an observation is related to a previously established track). Optionally, the developer also can implement a more specialized MHT algorithm than those provided in the TRAFEN framework, although this was not needed for our worm detection. We were able to use the existing MHT algorithm for our worm detection. The MHT algorithm uses the likelihood class and its own internal methods to assign probabilities to hypotheses and tracks, and the likelihood class uses the state class when it makes its likelihood comparisons. In this way, only the likelihood class directly examines the domain-specific state, allowing the same tracking algorithm to be used for multiple problem domains, and keeping the TRAFEN

framework independent of any particular problem domain. Finally, the developer also can provide additional filtering logic that will be applied to the incoming observation stream. We apply simple filtering logic, as described in the next paragraph, to keep only the most relevant DIB:S scan alerts.

For our active-worm detection, the observations are the scan alerts from the DIB:S analyzers, and for simplicity, the probability assigned to a track is the probability that the track represents a worm. Under direction from a domain-specific configuration file, TRAFEN subscribes to the observations from the DIB:S system, which provides the observations in XML format.[¶] TRAFEN filters out the two secondary alert types (Case 5 and 6), and filters out any alert in which one source address is scanning multiple *ports* on a target machine or machines, since such scanning is not typical of any active worm. TRAFEN passes the filtered observations to a dynamically loaded, simplified version of Reid's Multiple Hypothesis Tracking algorithm.¹⁰ The tracking algorithm, if it is receiving the first observation ever, will create a one-observation track with a very low probability (obtained from the domain-specific configuration). The low probability reflects the fact that a single observation of scanning activity does not by itself indicate a worm. For subsequent observations, the MHT algorithm iterates through each active hypothesis and each track inside the hypotheses. For each track, it calculates the likelihood that the observation is related to a track, or, in other words, that a scan represents a continuation of the worm scanning activity represented in the track. The algorithm then calculates the new track likelihood as the weighted average of the old track likelihood and the relational likelihood, and creates a *new* hypothesis in which the observation is added to the track. The probability of the new hypothesis is the weighted average of the individual track probabilities, which is sufficient for our application. The algorithm also creates a new hypothesis in which the observation is not associated with any existing track, but instead is in a track by itself (with the same low probability as the first observation). The MHT algorithm then prunes the hypothesis set, and keeps only the n most likely hypotheses, where n is a configurable value. In this way, the algorithm keeps those hypotheses that are most likely to be tracking an active worm.

The likelihood calculation, then, is the heart of the MHT algorithm. This calculation, which is encapsulated behind a general API from the standpoint of the MHT algorithm, is essentially rule-based (although the rules are implicitly represented in the code, rather than explicitly in a separate rulebase). After initial experiments, we arrived at three straightforward rules. *Rule 1:* If a machine scans the same port, using the same protocol, as the machines already in a particular track, the type match is *high* (0.9); otherwise the type match is *low* (0.1). This rule captures the fact that an active worm typically scans for and exploits one particular vulnerable service, although the rule could be extended easily to take into account those worms that scan two or more *related* service ports. *Rule 2:* If a machine performs a scan only a short period of time after a previous series of scans, the time match should be higher than if the scans occur farther apart. This rule captures the fact that an active worm must scan continuously if it is to propagate quickly. We assign a time match of 1.0 if the new scan occurs 10 seconds or less after a previous scan, a time match of 0.0 if a new scan occurs 300 seconds or more after a previous scan, and a time match scaled linearly between 0 and 1 if the scan is between 10 and 300 seconds after the previous scan. Although the exact thresholds have little effect on tracking performance, these thresholds are best for fast-moving worms. More will be said about slow-moving worms in the Future Extensions section. *Rule 3:* Finally, if the type match is low, the overall likelihood that the new scan is related to the tracked scans is set low, again 0.1. Two scans on different destination ports likely do not represent the same active worm, no matter how closely together those two scans occur in time. If the type match is high, the overall likelihood is set between 0.675 and 0.925, scaled linearly according to the time match. Again, the exact values of 0.675 and 0.925 do not have a significant effect on tracking performance, as long as the high end of the range is greater than our worm detection threshold in the next section. Since the probability of an initial single-observation track is set to a low value, the rules ensure that it takes several observations for the track probability to increase significantly, reflecting the fact that only a series of scans can indicate a worm.

Since the three rules are simple, it could be argued that TRAFEN was not needed, and in fact, the rules could be represented in a simpler system that counts the number of related scans that occur in a moving time

[¶]DIB:S collects a tremendous volume of ICMP messages, but abstracts and summarizes those messages to create the scan alerts. A much lower volume of messages is sent to TRAFEN, and the overhead associated with the XML representation is not significant.

window. In addition, the rules can produce false positives, whenever a significant number of coincidental scans occur on the same port within a short period of time (for example, a set of hackers independently searching for the same vulnerability with manual scanning utilities). Even with the current ruleset, however, the number of coincidental scans must be relatively large. Moreover, a more complex worm detector could provide two process descriptions to TRAFEN, one for worm activity and one for coincidental scanning activity, allowing TRAFEN to create hypotheses in which each track has a *type*, namely, *worm* or *coincidental scan*, and greatly reducing the possibility of a false positive. Overall, the TRAFEN framework allowed us to produce a working worm detector (given the DIB:S input) in only a few hours, and provides the flexibility to extend the tracking system later, through more complex models, likelihood calculations, or even MHT algorithms. In the next section, we will examine the detection performance of the current ruleset, and in the following section, we will discuss extensions to the current TRAFEN-based worm detector.

4.4. Detection Capabilities

DIB:S and TRAFEN currently are deployed at Dartmouth College, with instrumented Dartmouth routers sending their ICMP-T3 messages to our DIB:S installation. This initial local deployment is not enough to analyze the detection performance of the system, however, and we turn to simulated worms for that purpose. Our simulator allows a worm to propagate through a virtual address space according to the model from the previous section, and generates real ICMP-T3 messages as the worm probes unreachable addresses. These ICMP-T3 messages are fed into the DIB:S analyzers, just as would be done with the ICMP T-3 messages from an actual deployment.

Figure 5 shows the detection performance of DIB:S and TRAFEN for simulated Sapphire-like worms. Both graphs in Figure 5 show the percentage of infected machines at the time that the worm was detected. In Figure 5a, this percentage is shown as a function of the router coverage, namely, the percentage of the address space that is behind an instrumented router sending carbon copies of its ICMP-T3 messages. Each line in the graph corresponds to a different network size, or, more specifically, to a different address-space size. As a companion to Figure 5a, Figure 5b shows the same results as a function of the network size, with each line corresponding to a particular router coverage. For each network size, 75% of the addresses were unreachable, 25% of the addresses were reachable, and 0.1% of the addresses were reachable *and* vulnerable. For example, for a network size of 500,000 unique addresses, 375,000 addresses are unreachable, 125,000 are reachable, and 500 are vulnerable. The reachable 25% corresponds to our observed data from the scans of selected *populated* address ranges, while the vulnerable 0.1%, although large, corresponds to a vulnerability in Web, mail, database, or other widely installed software. Each data point in the graphs is an average across ten simulated worms; each simulated worm probed 100 target address per infected machine per second, slightly lower than, but consistent with, the average Sapphire/Slammer scan rate; DIB:S had to receive $N = 5$ ICMP-T3 messages for the same IP address before issuing an alert; and DIB:S maintained a history window of $t = 300$ seconds. Each of the addresses in the address space was pre-determined to be one of the vulnerable machines, one of the remaining reachable, but immune, machines, or one of the unreachable machines. Each simulated worm was assumed to have already started on one of the vulnerable machines, perhaps a machine that an attacker would have identified through a manual scan. When a simulated worm attempted to contact a nonexistent machine through an instrumented router, an ICMP-T3 message was generated. Once the worm found a vulnerable machine, it infected that machine, and both the first and second worm instances continued to scan the address space. The worms selected random target addresses uniformly distributed through the address space, with the random seed for each worm instance derived from the current (simulated) time and the address of the infected machine. Each simulation run continued until the worm infected all vulnerable machines, and TRAFEN was assumed to have detected the worm as soon as the probability of a track containing the relevant scanning activity went above a likelihood threshold of 0.9.

The simple rules in our first version of the TRAFEN worm detector ensure that a worm will be detected as soon as TRAFEN receives seven to ten scan alerts for different source IP addresses (corresponding to seven to ten infected machines) within a sufficiently short period of time. As seen in Figure 5a, the detection performance improves significantly as the router coverage increases from 1% to 2%, but then levels off at different, roughly constant, values for the different network sizes. For a network size of 500,000, for example, the infection percentage starts at a peak of 5% when the router coverage is 0.5, but drops quickly to around 2% as the coverage

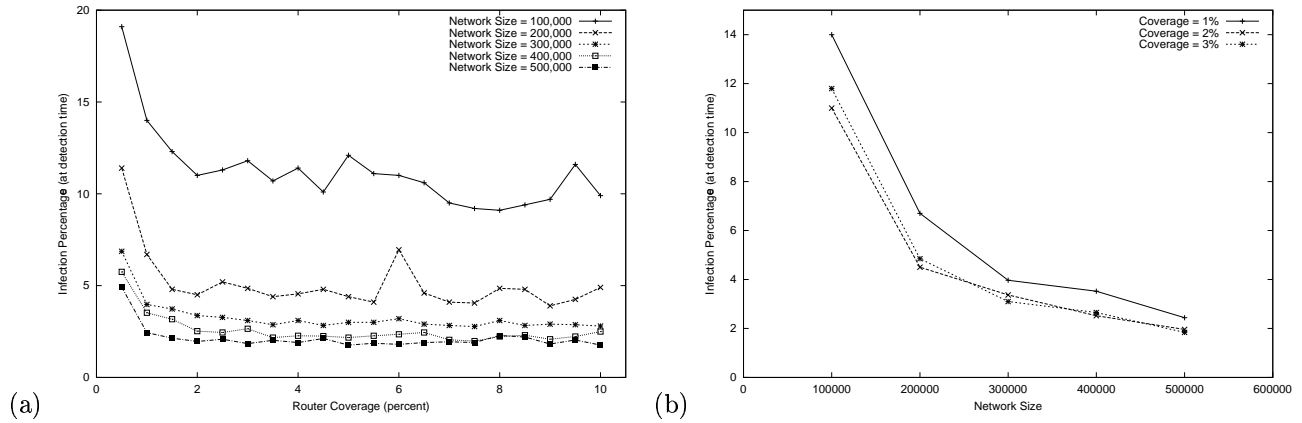


Figure 5. (a) The percentage of vulnerable machines already infected at the time that an active worm is detected. The horizontal axis is the router coverage, i.e., the percentage of the address space that is behind an instrumented router, while the y-axis is the infection percentage. Each line corresponds to a different network size, where the network size is the number of distinct addresses in the address space. (b) Same as graph (a), except that the x-axis is the network size, and each line corresponds to a different router coverage. For both graphs, each data point is an average across ten unique worms; each worm propagated at Sapphire/Slammer speeds; 25% of the address space contained reachable machines; and 0.1% of the address space contained vulnerable machines.

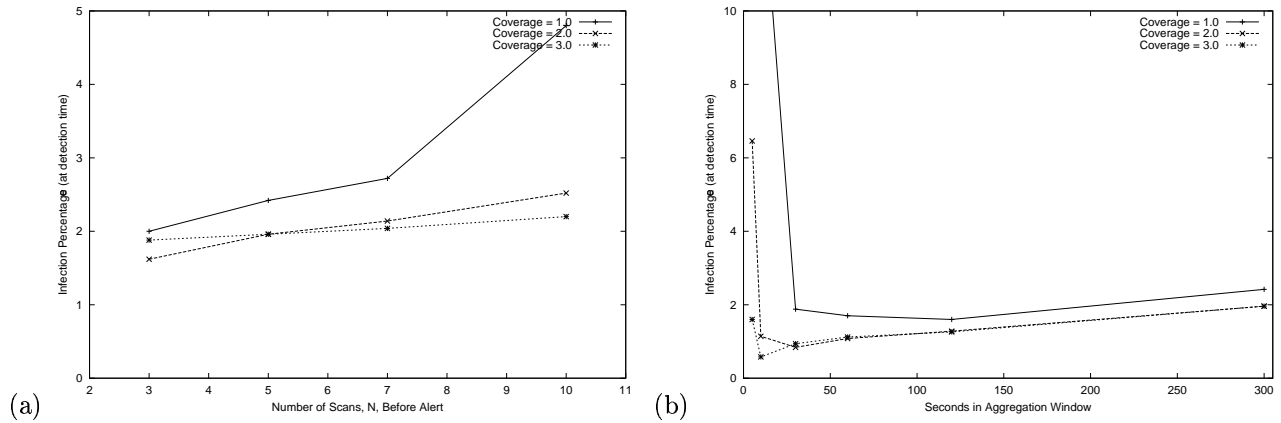


Figure 6. (a) Detection performance for different values of N , the number of ICMP T-3 messages required for the generation of a scan alert. (b) Detection performance for different values of Δt , the length, in seconds, of the history window over which ICMP T-3 messages are aggregated. For both graphs, the network or address-space size was fixed at 500,000, 125,000 addresses were reachable machines, and 500 addresses were vulnerable machines.

increases. The straightforward reason is that, for router coverages of 2% and higher, DIB:S receives enough ICMP T-3 messages to reliably detect the scanning activity of the *first* seven to ten infected machines. Thus, at these higher coverages, the detection always will take place within a fixed number of infected machines, no matter whether the coverage is 2% or 10%. For router coverages below 2%, however, DIB:S will not receive enough ICMP T-3 messages to reliably detect all scanning activity, and correspondingly more machines will be infected before DIB:S can conclude that a worm is present. The critical message of this graph is that router coverage of 2% provides just as good detection performance as higher coverages, meaning that we need only a modest number of instrumented routers, and that we need only transmit and process a manageable volume of ICMP T-3 messages.

As seen in both Figures 5a and 5b, the detection performance improves as the network size increases. The explanation is simply that DIB:S detection performance is dependent not so much on the percentage of machines

infected so far, but on the absolute number of infected machines and the amount of scanning activity that the worm generated while infecting those machines. In fact, if the infection percentages are converted into the absolute number of infected machines, we can see that detection occurs at approximately 12 infected machines in a 100,000-address network, and at a comparable 10 machines in a 500,000-address network. Overall, in terms of our ability to detect the worm early and eventually protect the largest *percentage* of vulnerable, but not yet infected, machines, we can keep the router coverage fixed, and still do better and better as the network size increases. Alternatively, for a larger network, we can achieve the same detection performance with a smaller router coverage.

There are many parameters within the DIB:S and TRAFEN systems that affect detection performance. Two of the most important are N , the number of ICMP T-3 messages per generated DIB:S alert, and Δt , the size in seconds of the DIB:S history window. Figure 6a shows the detection performance as a function of N , while Figure 6b shows the detection performance as a function of Δt . For both graphs, the network size is 500,000, and the number of vulnerable machines is 500. When N is varied, Δt is held fixed at 300 seconds, and when Δt is varied, N is held fixed at five ICMP T-3 messages per alert. In Figure 6a, we see that detection performance decreases as N increases, particularly when the router coverage is only 1%. At lower coverages and higher values of N , DIB:S might not see enough ICMP T-3 messages to actually generate an alert, and scanning activity will go unreported. In Figure 6b, we see that detection performance is very poor for the lowest values of Δt , and then after an initial improvement decreases steadily as Δt increases. The very poor performance is due to the fact that when the history window is too small, ICMP T-3 messages will expire from the history window before enough messages are received to produce an alert. The steady decrease in performance after the initial improvement is arguably illusory, since when Δt is small, DIB:S will generate multiple scan alerts for the same source address, whereas when Δt is large, DIB:S will generate only one scan alert per source address (during the worm's initial propagation). Although the multiple alerts per source address drive the track probability in TRAFEN above 0.9 quite quickly, multiple scans from the *same* source address are not, in fact, a reliable indicator of worm activity. They could merely indicate an intense, but manual, scanning effort. In the current system, therefore, Δt must be kept high enough to avoid "duplicate" alerts within too short a time period.

The results here are representative of the results that we have seen for a wide range of simulated worms. For example, in a recent paper,¹⁶ we saw the same results for simulated worms that spread at Code Red speeds, 5-6 probes per second, rather than Sapphire/Slammer speeds, in excess of 100 probes per second. Different worms produce significant differences in absolute time values, but demonstrate similar behavior in terms of the percentage of infected machines at detection time. Of course, the current TRAFEN ruleset will only detect worms that move relatively quickly, attempting to infect most or all of the vulnerable population within minutes or hours. With slower-moving worms, TRAFEN will not see enough scan alerts until a large percentage of hosts are infected. We discuss this and other challenges next.

4.5. Future Extensions

With the current TRAFEN ruleset and thresholds, the probability that a track represents a worm will rise above the detection threshold of 0.9 only if the worm eventually generates a series of (observed) scans that occur within 39 seconds of each other.^{||} Worms that spread at Code Red speeds or better will quickly generate such a series of scans, and TRAFEN easily detects them. Slower scanning worms, however, might not generate such a sequence until the later stages of an infection. Clearly, the thresholds in the time calculations (or the overall probability threshold of 0.9) could be lowered, but this leads to a problem with false positives.

For false positives, our experiments have shown that *random* scanning noise does not affect detection performance, but not all scanning noise is random. For example, attackers constantly scan TCP port 80 looking for vulnerable Web servers. If many of these scans coincidentally occur within seconds of each other, TRAFEN incorrectly will detect a worm that exploits Web servers. Similarly, although lowering N , the number of ICMP-T3 messages needed to detect a scan, gives us faster tracking and detection times, there is a limit to how low N can

^{||}There must be a series of scans, each of whose probability match with an existing track is *greater than* 0.9. An overall probability of 0.9 means that the time probability must be at least 0.9, which corresponds to 39 seconds between scans with the current time rule.

be. Setting $N = 1$ would result in DIB:S not analyzing anything, but instead passing every unreachable message to TRAFEN. In this case, TRAFEN sees all ICMP-T3 “noise” and has significant problems distinguishing between scanning activity and innocuous ICMP-T3 messages after a normal failed connection attempt. Conversely, when N is chosen too high, DIB:S will generate an accurate view of the world (and will be immune to almost any noise in the ICMP-T3 data), but will generate that accurate view far too late. Choices for Δt are less critical, as long as Δt is high enough that older ICMP T-3 messages do not expire from the history window too soon.

TRAFEN presents graphs of current ICMP T-3 activity to system administrators, and the graphs for worms and coincidental scanning activity are immediately and visually distinct, meaning that false positives are less of a problem than they might be in other intrusion-detection applications. We still need to minimize false positives, however, since any false positive involves administrator time. Fortunately, the TRAFEN and DIB:S framework provides significant flexibility for improving the ruleset. In our case, we need to extend the ruleset so that it takes into account all of the scanning characteristics of a propagating worm. Scanning activity uniformly distributed in time, no matter how intense, likely is not a worm, but instead simultaneous, but unrelated, attacker scanning efforts. On the other hand, scanning activity that increases linearly or exponentially over time almost certainly is a worm, no matter how much time that increase takes.

The most important question, then, is how quickly we can detect an exponential increase in scanning activity (i.e., detect the worm) without incorrectly classifying non-exponential behavior as exponential (i.e., avoid false positives). We must detect the worm even if it is spreading slowly, and we must separate simultaneous exponential and non-exponential processes in case a worm and a human attacker coincidentally are targeting the same port at the same port. There are several modeling techniques that can be used to detect a wide range of worms while still minimizing false positives, but we are particularly interested in Hidden Markov Models,¹⁵ which (loosely speaking) allow a system to infer the state of an unobservable generation process through statistical properties of the observed effects of that process. Hidden Markov or other models could be defined for the scanning activity associated with worms, machine failures, and the simultaneous, but unrelated, activity of individual attackers. The MHT algorithm then could hypothesize about the type (worm, host failure, coincidental) of the observed scanning activity, rather than just the likelihood that the scanning activity represents a worm. In addition, since the models could be applied at different time scales, the MHT algorithm could hypothesize about both fast- and slow-moving worms. If Hidden Markov Models turn out to be unwieldy, mathematical models that simply match observed activity to linear and exponential curves and uniform random distributions, again at different time scales, still will provide significant improvement over the current ruleset.

To evaluate the performance of improved detection models, we will need both finer-grained simulation and more intensive real-world tests. Our current simulation does not provide sufficient resolution to explore signal-to-noise and false-positive issues. It can generate *random* scanning noise only, which does not take into account the fact that attackers routinely scan for particular services, not entirely random target ports. Just as importantly, the current simulation assumes that susceptible machines are uniformly distributed throughout the address space. In the real Internet, however, vulnerable systems often are concentrated within certain border networks, and there is generally more traffic within these networks than between them. For more accurate simulation, we are currently working with simulation experts specializing in massive Internet traffic simulation. The aim is to have accurate packet-level simulation of worms *within* autonomous systems (microscopic level) and use that data to simulate the traffic *between* autonomous systems (macroscopic level).¹⁷ This cooperation looks very promising.

Simultaneously, we are working to deploy additional instrumented routers within the networks of selected partners. Initially, this deployment will provide information about the amount of scanning “noise”, the independent scanning activities that serve to obscure a propagating worm. Later, the deployment will be used to test enhanced worm-detection models. Finally, production use requires a wide deployment on the Internet. Given that detection performance improves as the network size increases, a participating router coverage between 1% and 2% would be enough to provide reasonably early warning of Internet epidemics.** Achieving a coverage of 1.5%, which corresponds to approximately 3.5 Class A networks, will be administratively difficult, but can be achieved with a mix of Internet Service Providers or other large organizations. Alternatively, large portions of the Internet address space are unassigned. If these unassigned address ranges were routed to a system that

**Remember that this is coverage in terms of address space, not the total number of routers.

provided no response to the sender, but merely forwarded appropriate alerts to TRAFEN, we would gain significant data with minimal risk of “noise”. Unassigned address ranges never should be contacted in normal Internet communication. Of course, there still would be administrative overhead, and there also would be the added risk that worm authors quickly would adapt their worms to avoid any unassigned space.

In terms of scalability, if DIB:S is installed at a single, central location, the network bandwidth will limit the number of incoming ICMP T-3 messages. This limit is not as serious as it might appear, however. With 64K instrumented routers covering 4 Class-A networks, for example, the routers would generate approximately 200 Mbps of ICMP T-3 messages (assuming that each router was configured to generate only four ICMP T-3 messages per second as commonly is done now). In addition, if 200 Mbps is too much network traffic for a single collector site, DIB:S can be distributed almost to an arbitrary degree. Instrumented routers can send their ICMP T-3 messages to “nearby” collectors, and the analyzers, each of which is in charge of a particular address range, can be distributed throughout the Internet. Even TRAFEN could be distributed by having different copies of TRAFEN handle different sets of destination ports, although this likely is not necessary since the stream of scan alerts coming from DIB:S is significantly smaller than the stream of ICMP T-3 messages flowing into DIB:S.

ICMP-T3 messages are not the only data source that can provide indications of worm activity. Although ICMP messages are particularly attractive since they indicate scanning activity that spans multiple independent networks, scan reports and other information from firewalls, intrusion-detection systems and even host-based sensors also can be fed into the DIB:S/TRAFEN system, serving as a useful complement to the ICMP-T3 data. The ICMP-T3 messages can provide useful additional information themselves, since passive OS fingerprinting^{††} would allow DIB:S to infer the type of the operating system that is performing the scan, adding to the hypothesis-generation ability of TRAFEN. Two scans originating from a Linux and Windows machine respectively almost certainly do not belong to the same worm.

Finally, regardless of how effective an early warning system is, there is no use in detecting a worm unless something can be done. As discussed earlier, this can be as little as informing system administrators or as much as having a framework in place that will automatically reconfigure firewalls and IDS systems as the epidemic is occurring. Automated response will be a critical topic of future work, both for our group and many others. Even so, early warning is always worthwhile.

5. FUTURE

When we think of the history of computers and the Internet, we have to conclude that history is not a very solid indicator of the future. The way computers and micro-controllers have integrated our lives, through appliances, motor vehicles, communication systems, and personal computers, was completely unforeseen at the time the first computers were constructed with vacuum tubes. It often has been said that truth is stranger than fiction, and this is certainly true in the computing and communication fields. Therefore, it would be pretentious of us to try to paint a picture of the future of computers, the Internet, and their malware. We can make a few inferences, however, based on the worm code of today and proposed techniques for improving worm capabilities.

Over the last several years, computers have increasingly taken on the task of “home appliances”. They integrate the home-office, game-console, and DVD-player all in one machine. With the emergence of voice-over-IP and television recording and playback, the capabilities keeps growing. As computers start to integrate more and more “home services” into one machine, the code-base on these computers grows fast. Broadband Internet is commonplace in many homes, steadily increasing the number of connected systems. With this increase in the number of connected systems, as well as an increase in the tasks they perform, it would be ridiculous to say that (remotely exploitable) vulnerabilities will be a thing of the past. Additionally, when application software is produced by only a few large and monolithic companies, the chances of a homogeneous vulnerability, (i.e., a vulnerability present in a majority majority of connected systems) becomes more likely. It goes without saying that this is an ideal environment for viruses and worms, so they will be around in the future as well. Moreover, with the increased bandwidth available, they will be spreading lightning fast. In this vein, there will

^{††}Michael Zalewski wrote some of the first passive-fingerprinting code, which is available at <http://www.stearns.org/pOf/>.

remain a desperate need for diversity in software and operating systems, decreasing the likelihood of massively homogeneous vulnerabilities.

The increase in connectivity also has prompted a shortage in available IP address space. Although this shortage has been mostly mended Network Address Translation (NAT), eventually a more structural solution will be needed. Increasing the address space will bring with it the nice property that random scanning for vulnerable IP addresses will become nearly impossible. IPv6 offers 128 bits of address space versus the 32 bits available in IPv4, requiring a significant change in the way authors write their worms. As an example, we take Code Red v2, which we analyzed for IPv4 in Figure 3. We limit ourselves to propagation within a single IPv6 site, which has 2^{64} possible IP addresses. We assume 2^{16} responding machines, of which $1/100^{th}$ are vulnerable. We pick $\gamma = 0$, so that there is no recovery or removal and the worm is free propagate. This makes $r(t)$ a constant, leading to $s(t) + i(t) = M$ being a constant as well, and effectively rewrites the epidemic-model equations (see also Daley and Gani⁶):

$$\frac{di}{dt} = \beta si = \beta(M - i)i \quad (10)$$

This also is known as the logistic growth equation, and it represents a worst-case epidemic in which there are no recoveries or disconnects, and each infective stays infective forever. Propagation speed will be higher than in a realistic scenario, but this allows us to define the cap, the fastest propagation time possible. Citing Daley and Gani⁶ once more for the integral over $(0, t)$, we have

$$i(t) = \frac{i_0 M}{i_0 + (M - i_0)e^{-\beta M t}} \quad (11)$$

We can use this formula to find out how fast a worm would spread in the fastest scenario, given ideal connectivity and no countermeasures. To do so, we set $i(T_\epsilon) = \epsilon M$ with ϵ being the fraction of susceptibles infected (for example we could define T_{END} by taking $\epsilon = 0.95$):

$$i(T_\epsilon) = \epsilon M$$

Moving terms across the equality gives

$$i_0 M = \epsilon M i_0 + \epsilon M (M - i_0) e^{-\beta M T_\epsilon}$$

Isolating e and inverting both sides of the equation

$$\frac{\epsilon M (M - i_0)}{i_0 M - \epsilon M i_0} = e^{\beta M T_\epsilon}$$

Simplifying the fraction and taking the \log_e of both sides yields

$$T_\epsilon = \frac{1}{\beta M} \times \ln \left(\frac{\epsilon (M - i_0)}{i_0 (1 - \epsilon)} \right) \quad (12)$$

Looking at Equations 10 and 12, we note two important properties. First, realizing that $\mathcal{O}(\beta) \gg \mathcal{O}(M)$ and that $\mathcal{O}(M) \approx \mathcal{O}(i)$, it is clear that the propagation time will be mostly dependent on β . Second, the relationship between propagation time and β is a linear one. If β is doubled, $\frac{dt}{dt}$, which is the propagation speed, also doubles. If the speed is doubled, the time it will take for all hosts to be infected will be halved. The linear relationship with β , as well as M , also can be clearly seen from Equation 12. Now we can fill in the numbers for Code Red v2 in IPv4 space, assuming the initial number of infected hosts in this site is 10, and we are looking for how long

it takes to infect 95% of all susceptible hosts. Remembering that $\beta = 1.23 \times 10^{-9}$ (see the caption of Figure 3), we have a time in seconds of

$$T_{0.95} = \frac{1}{1.23 \times 10^{-9} \times 360000} \times \ln \left(\frac{0.95 \times (360000 - 10)}{10 \times (1 - 0.95)} \right)$$

which is $30220/3600 = 8.4$ hours, a good approximation of what we can read from Figure 3 and thus verifying our equations. Now we fill in the numbers for the Code Red v2 worm propagating within one IPv6 site. First we calculate r , τ , and β : $r = 2^{16}/2^{64} = 2^{-48} \approx 10^{-15}$. We obtain τ by filling in equation 4: $\tau = 10^{-15} \times 1 + (1 - 10^{-15}r) \times 21 \approx 21$. Giving β by filling in equation 5: $\beta = \frac{1}{2^{64}} \times \frac{100}{21} \approx 2.5814 \times 10^{-19}$. With $M = 2^{16}/100 \approx 655$, the time to reach 95% propagation is:

$$T_{0.95} = \frac{1}{2.5814 \times 10^{-19} \times 655} \times \ln \left(\frac{0.95 \times (655 - 10)}{10 \times (1 - 0.95)} \right)$$

which gives 4.2057×10^{16} , over 1.3 billion years, and confirms the intuition that an enlarged address space will pose a significant challenge to randomly propagating worms.

This will undoubtedly lead to new and improved target selection techniques, most of which were already discussed in the *Worms and Viruses* section. We will mention two of them again, however, and suggest probable detection strategies. To acquire IP addresses of hosts running a vulnerable service the worm could sniff the network wire for traffic from that service. Mail and DNS servers will be most vulnerable to this, since they constantly communicate between peers. Imagine a worm that moved from Web browser to Web server, and from Web server to Web browser. Similar worms could be designed for any peer-to-peer or client/server service (as long as the clients regularly communicate with different services). One possible way of detecting such a worm is by inserting bogus communication into the network. By spoofing non-existent IP addresses and so making fake queries to all the services in the network, sniffing worms can be provoked to connect to these non-existent machines. The challenge would be to make the fake communication look as real as possible, to ensure that the worm cannot distinguish between real and false events. The worm will attempt to connect to these nonexistent IP addresses, and these attempts then would provoke ICMP Destination Unreachable messages, once again providing a fast and early indicator of a propagating worm.

The DIB:S/TRAFEN system also can be used in the case of DNS exploration. As noted before, worms can gain hostnames from probing DNS servers and potentially trying whole ranges of possibly related hostnames (recall the example with *sparc01*, *sparc02*, ... *sparc99*). DIB:S could be configured to receive notification of all failed DNS queries, as a blind carbon copy from name servers. The analogy is simple: one IP address attempted to contact many *hostnames* on many different networks (and failed). This would be a clear bloom, and TRAFEN soon would detect the worm when multiple hosts show the same behavior. The analogy ends, however, when we try to distinguish between multiple worms that are using DNS queries to obtain IP addresses. There will be no significant differences. To tackle this problem, a more radical approach is needed. DNS servers could be configured to respond with unassigned IP addresses when presented with repeated DNS requests for unknown hostnames. This way worms would attempt to connect to the targeted service port on unassigned IP addresses, leading to generation of ICMP-T3 messages suited for DIB:S. The drawback is a long timeout when users mistype hostnames, since the DNS server might give an unassigned IP address instead of an error message.

Finally, companies that today provide automated virus-signature update services will continue to do so, will add worm signatures, and possibly will provide services for content-based filtering in transit. To elude detection during propagation, however, worms can be created that use some of the same polymorphism techniques as the most advanced viruses. The easiest way of creating polymorphic code is by encryption. A small section of code can decrypt the entire program using a key that is propagated along with the code. When such a worm found a vulnerable target, it would create a key, then encrypt itself, and propagate both the key and the encrypted version of its code. On the newly infected machine, the key would be used to decrypt the worm and start the procedure of looking for new vulnerable hosts. When the initial decryption code is relatively small, it might be

very difficult to create a proper signature for such a worm. An alternative for a polymorphic worm would be to permute the basic code blocks before each propagation. These blocks are linear sets of instructions between branches or jumps, and can be moved to arbitrary locations when all the corresponding branch instructions are updated properly. Signatures can no longer be singular in such a case, and would need to include several sufficiently large basic blocks and a query that checks if all those blocks are present, regardless of their order. In short, inferring the existence of worms through their secondary network traffic (such as ICMP T-3 messages), rather than using signatures, always will be an important detection strategy.

6. RELATED WORK

In 1991, Jeffrey Kephart and Steve White already were working on analytical models of computer viruses and the epidemics they cause.¹⁸ The SIS model that they described still makes sense in the active-worm arena, and can be expanded easily to include I-R and R-S transitions. Other researchers, such as Moore, Shannon, Voelker and Savage⁷ and Zou, Gong and Towsley,¹⁹ start from the same equations of Kermack and McKendrick⁶ as we do, and arrive at related, but distinct, worm-propagation models. These models differ in how and if they include certain transitions and worm characteristics, but are able to make similar predictions about how long it will take a worm to spread through the Internet. In all of these models, the parameters governing the transitions are still basic. The formula we use for β , for example, does not take into account the dynamics of a saturated network, which was the primary limiting factor on the Sapphire/Slammer worm. It remains to be seen if there is a proper way to model the effect of Internet topology.

There has been some recent work on the detection of Internet worms. The NetBait system²⁰ does not provide automated detection of previously unknown worms, since it relies on the availability of signatures for extracting probe data from available logfiles. NetBait, however, does allow security personnel to pose complex queries against distributed, multi-source probe data through a highly efficient overlay network. After the detection of a worm through some other means and the development of appropriate signatures, system and network administrators can use NetBait to identify infected machines, and analyze the nature and extent of the epidemic. In addition, the NetBait architecture could support queries against general scanning data (such as DIB:S-like scan alerts). Such general queries might provide some measure of automated detection. Even if not, however, a system like NetBait could provide TRAFEN with a means for efficiently fetching additional information with which to confirm or refute the presence of a worm. Other systems that allow complex queries against distributed attack data, such as Kerf,²¹ could play a similar role.

In contrast to the more manual nature of NetBait, Zou, Gao, Gong, and Towsley have developed an approach based on Kalman filters for automatically detecting worms based on their scanning activity.²² This work place ingress and egress scan monitors at key network points, and collects the resulting scan alerts. A Kalman filter, which has the advantage of being robust to missing scan data, is applied to the scan alerts (for a particular port) to see if the pattern of scanning activity matches their SIR-based model of worm propagation. For an address space with 2^{32} addresses (i.e., the Internet), monitoring coverage of $2^{17}/2^{32}$, and 500,000 vulnerable machines, their system can detect a simulated Code Red worm, and predict its overall infection rate, as soon as the worm infects approximately 5% of the vulnerable machines. From the standpoint of our work, the Kalman filter has several attractive features compared to our current ruleset, and could be a plug-in replacement for that ruleset within the TRAFEN framework. Comparing the detection performance of other rulesets, Kalman filters, and our planned Hidden Markov models will be an interesting component of future work.

Both signature-based and anomaly-based²³ intrusion-detection systems can detect worm scans and probes. These systems, however, see only the network traffic that reaches a particular network boundary, and thus might not recognize a scan or probe as evidence of a propagating worm. Some researchers, such as Burroughs, Wilson and Cybenko²⁴ and Neumann and Porras,²⁵ are collecting and analyzing data from *distributed* intrusion-detection systems. Such systems can provide more insight into worm activity than stand-alone systems, but scans still might be overlooked if they hit any individual network only a few times. By collecting ICMP-T3 messages from a broadly deployed set of instrumented routers, DIB:S can detect a scan even if that scan never hits an individual network more than once. On the other hand, distributed intrusion-detection systems could provide additional data that would improve TRAFEN's detection capabilities. Finally, some researchers are using similar

architectures to detect different classes of distributed attacks. Cabrera, Lewis, Qin, Lee and Mehra, for example, collect and analyze SNMP MIB data to detect denial-of-service attacks.²⁶

Institutions like SANS (www.sans.org) and CERT (www.cert.org) provide first responder information regarding new vulnerabilities in operating systems and services. It is a known fact that most worms use older, well known vulnerabilities and epidemics can be avoided if patches and updates are widely applied. The value of these services is enormous, when system administrators are using them regularly. Researchers also are beginning to consider automated response mechanisms. Moore, Shannon, Voelker, and Savage, for example, investigated address blacklisting and content filtering as two automated ways to slow or stop the spread of a worm.⁷ DIB:S/TRAFEN could be used as the detection system that triggers their automated responses. CAIDA, the Cooperative Association for Internet Data Analysis at UC San Diego (<http://www.caida.org/>), does a significant amount of worm analysis and research, including the automated response work of Moore, Shannon, Voelker and Savage. The DShield service (<http://www.dshield.org/>) provides alerts of hosts that are engaged in scanning activity, but relies on system administrators to report such scanning activity. DIB:S/TRAFEN could form the basis for a fully automated DShield service.

REFERENCES

1. E. H. Spafford, "The Internet worm: Crisis and aftermath," *Communications of the ACM* **32**, June 1989.
2. D. Moore, C. Shannon, and J. Brown, "Code Red: A case study on the spread and victims of an Internet worm," in *Proceedings of the Second Internet Measurement Workshop (IMW 2002)*, November 2002.
3. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the Sapphire/Slammer worm," technical report, CAIDA, 2003.
4. M. W. Eichin and J. A. Rochlis, "With microscope and tweezers: An analysis of the Internet virus of November 1988," in *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy*, May 1989.
5. S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium (Security '02)*, (San Francisco, California), August 2002.
6. D. Daley and J. Gani, *Epidemic Modeling*, Cambridge University Press, 1999.
7. D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, April 2003. To appear.
8. M. M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," Tech. Rep. 172, HP Labs Bristol, 2002.
9. "Microsoft SQL Sapphire worm analysis," technical report, eEye Digital Security, 2003. Available at <http://www.eeye.com/html/Research/Flash/AL20030125.html>.
10. D. B. Reid, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control* **AC-24**, pp. 843–854, December 1979.
11. F. Baker, "RFC 1812: Requirements for IP version 4 routers," *Request for Comments* **1812**, 1995.
12. J. Postel, "RFC 792: Internet Control Message Protocol," *Request for Comments* **792**, 1981.
13. V. Berk, W. Chung, V. Crespi, G. Cybenko, R. Gray, D. Hernando, G. Jiang, H. Li, and Y. Sheng, "Process query systems for surveillance and awareness," in *Proceedings of the 7th World Multifconference on Systems, Cybernetics and Informatics (SCI 2003)*, (Orlando, Florida), July 2003.
14. R. G. Brown and P. Y. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, John Wiley & Sons, 1983.
15. L. R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proceeding of the IEEE* **77**, **Num. 2**, pp. 257–286, 1989.
16. V. H. Berk, R. S. Gray, and G. Bakos, "Using sensor networks and data fusion for early detection of active worms," in *Proceedings of AeroSense 2003: SPIE's 17th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, (Orlando, Florida), April 2003.

17. M. Liljenstam, Y. Yuan, B. Premore, and D. Nicol, "A mixed abstraction level simulation model of large-scale Internet worm infestations," in *Proceedings of Tenth IEEE/ACM International Conference on Modeling, Analysis and Simulation of Computer and Communications Systems (MASCOTS 2002)*, October 2002.
18. J. Kephart and S. White, "Directed-graph epidemiological models of computer viruses," in *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991.
19. C. C. Zou, W. Gong, and D. Towsley, "Code Red worm propagation modeling and analysis," in *Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS 2002)*, (Washington, DC), November 2002.
20. B. N. Chun, J. Lee, and H. Weatherspoon, "Netbait: A distributed worm detection service." Available at <http://netbait.plain-lab.org/>, 2003.
21. J. Aslam, S. Bratus, R. Peterson, D. Rus, and B. Tofel, "The Kerf toolkit for intrusion analysis." In Preparation., 2003.
22. C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," Technical Report TR-CSE-03-01, University of Massachusetts at Amherst, 2003.
23. W. Fan, M. Miller, S. Stolfo, W. Lee, and P. Chan, "Using artificial anomalies to detect known and unknown network intrusions," in *Proceedings of the First International Conference on Data Mining*, November 2001.
24. D. J. Burroughs, L. F. Wilson, and G. V. Cybenko, "Analysis of distributed intrusion detection systems using Bayesian methods," in *Proceedings of the 21st IEEE International Performance, Computing and Communications Conference (IPCCC 2002)*, April 2002.
25. P. G. Neumann and P. A. Porras, "Experimentence with EMERALD to date," in *Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring*, (Santa Clara, California), April 1999.
26. J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, and R. K. Mehra, "Proactive intrusion detection and distributed denial of service attacks – a case study in security management," *Journal of Network and Systems Management* **10**, June 2002.