Midgard Worms: Sudden Nasty Surprises from a Large Resilient Zombie Army

Peter Reiher
reiher@cs.ucla.edu
University of California, Los Angeles

Jun Li
lijun@cs.uoregon.edu
University of Oregon

Geoff Kuenning
geoff@cs.hmc.edu
Harvey Mudd College

Abstract

Future network intruders will probably use a zombie army to deliver many different attacks, rather than recruiting a new army per attack. We describe a Midgard Worm, which can build an extremely resilient and scalable overlay network to deliver attack code quickly. The worm's master could disseminate a 1-megabyte exploit or upgrade to a million zombies from any zombie in less than six minutes. Even if 80% of the zombies were disinfected, 70% of the remainder would remain connected and ready to receive new exploits. We discuss the basic design principles behind such a worm and methods of combating this kind of attack.

1. Introduction

Until recently, computer worms seemed to be mainly proofs of concept or idle entertainments for hackers interested in causing random mayhem [Spafford 89, CERT CA-2001-19, Moore 03]. But experts have predicted that worms would begin to recruit armies of machines for purposes beyond merely demonstrating their danger [Staniford 02], and the recent Blaster worm outbreak appears to have been designed to do precisely that [CERT CA-2003-20]. Since a well-designed worm can compromise hundreds of thousands, or perhaps even millions of nodes [Staniford 02], it can serve as an excellent mechanism to recruit vast zombie armies to perpetrate distributed denial of service (DDoS) attacks, serve as spam sources, or perform any other nefarious action that requires large numbers of computers.

While the worm author could put the code for his army to execute in the worm's payload, there are good reasons not to. First, it increases the payload size and may slow down worm propagation. Second, distributing attack code in the worm's payload limits flexibility. The worm can perform that single action, but no other. Third, including the payload might provide defenders with information about the worm, possibly allowing them to limit its spread. Also, defenders will receive a complete copy of the code that the zombie army will execute, usually well before it runs, allowing them to analyze and counteract it. While the attacker can encrypt the payload, he must either include the key with it or distribute it later. If the key is included with the payload, the encryption is of little value. If he distributes the key later, he might as well choose the more flexible option of distributing the payload at the same time. Thus, sophisticated worms are likely to have a more general mechanism for delivering exploit code to their zombie armies.

For example, Sobig.F zombies attempted to download exploit code from 20 different Internet nodes at a particular time [CERT IN-2003-03]. This code seems to have been relatively harmless, but since Sobig.F zombie nodes were totally compromised, the attack could have performed any malicious actions on those computers, from wiping their hard drives to launching massive attacks toward others. However, because Sobig.F used a fairly unsophisticated code distribution mechanism, authorities prevented its zombies from obtaining their code.

More sophisticated code distribution mechanisms are, unfortunately, all too possible. [Staniford 02] introduced a "worm network" concept for direct worm-to-worm communication and programmable worm updates. In this paper, we describe how an attacker could actually implement a worm network that would be highly effective and extremely difficult to counter, and through which an attacker could disseminate arbitrary exploit code to zombies at any time to launch a second-wave attack. We describe a class of worms we call Midgard worms.[1] Midgard

[1] Named after a monstrous serpent in Norse mythology that circled the world and held its own tail in its mouth. At Ragnorak, the Norse end of the world, the Midgard serpent would emerge from the ocean and spit venom over the entire world.

worms build up a highly resilient code dissemination structure based on creating an overlay network of compromised nodes. Midgard worms exploit the existing characteristics of worms (such as repeated attempts to infect already infected machines) to improve the structure's resiliency and speed of dissemination. This class of worms is highly feasible and could be designed by an attacker of no more than moderate sophistication.

The new danger of a Midgard worm is that it offers the attacker a resilient, surreptitious "anytime, anywhere, any flavor" capability: attacks can be launched at a moment's notice, injected at any point in the overlay, and can be crafted to any purpose. Attackers could use a Midgard worm to create a network of compromised machines that would allow them to perform many "useful" activities, such as sending massive amounts of spam to machines all over the Internet, extorting money via threats of devastating DDoS attacks, breaking cryptographic keys by brute force, storing and distributing stolen software or data for fun and profit. A variety of malicious activities could happen at any time in any sequence. The owner of the zombie army could even rent it out to others to allow them to perform any desired activity that required large numbers of nodes.

Our preliminary analysis suggests that a Midgard worm's code dissemination network could distribute a 1-megabyte exploit program to 1 million sites in under 6 minutes.

Midgard worms will appear sooner or later. Most of the component parts of such worms have already been observed in released worms or other malicious code. The remaining parts resemble other popular and well-known programs (such as peer file-sharing networks) and are not tremendously challenging to code. This paper discusses the logical evolution and feasibility of this worm. It also draws attention to the danger posed to the Internet community by Midgard worms and to spur government, industry, and researchers to find solutions to this problem before it becomes a crisis.

In scope and detail, this paper goes beyond existing work that suggests the possibility of building worm networks. We demonstrate how easy it would be to use a worm to automatically create and manage a code dissemination network, and discuss the likely performance characteristics of such a worm and its network, including how long it would take for its master to inject new code into a vast number of nodes, the likely performance impact of doing so on the Internet as a whole, and the resiliency of this dissemination network. Finally, we discuss methods that might be used to discover and counteract such a worm, and discuss what changes in the Internet would make the creation and use of such a worm less likely.

2. Structure of a Midgard Worm

The search and the infection procedures of a Midgard worm are the same as other worms. The unique characteristic of this kind of worm is that it creates a resilient self-organizing overlay of zombie nodes. This overlay allows the attacker to inject arbitrary exploit code at any time and from any point in the overlay, thereby disseminating the exploit code to all the recruited zombies. A Midgard worm can start building its overlay network during the infection phase, but further overlay construction will continue even if it stops propagating. Once a Midgard worm has recruited and organized its zombie army, the overlay network it has built serves as a superhighway for code propagation, even allowing upgraded versions of the worm itself to be distributed.

In order to disseminate exploit code, Midgard worms could build a resilient overlay that is similar to Revere overlays [Li 02b], or adopt any other type of overlays, including trees, hypercubes, butterflies, fat trees, or a random graph. Sophisticated and resilient overlays would make Midgard worms harder to defeat, and these are the focus of this paper.

The basic step in forming a Midgard overlay network is to incorporate a single new zombie into an existing zombie overlay. Except for the very first infected machine, every zombie needs to find some others already in the overlay and attach itself to them as a child. Here, a zombie can employ various methods to discover other overlay members. Obviously, this zombie was originally infected by some other machine, so it has no difficulty with the bootstrapping step of finding the first parent. It would be possible for that parent to pass the new child a list of other zombies who might serve as additional parents. It would also be possible for the child to simply wait for other zombies to attempt to infect it. Many of the methods that worm designers use to probe for susceptible nodes will cause multiple infected nodes to probe the same potential target. If a zombie is still looking for parents, each probing node is a candidate, so if an infected machine simply waits long enough, it is likely to hear from as many parents as it needs.

A zombie becomes another's child through a three-way-handshake procedure, initiated by either the child or the parent. When a zombie infects a new host or tries to infect an already infected machine, the parent-initiated handshake would be used, which would look like this:

P: Will you be my child?

C: Yes.

P: Great. You are now hooked up.

If initiated by the child, it would look like:

C: Can I be your child?

P: Yes, confirm if you still want me.

C: Yes. You are now my parent, and I am now your child.

The probe messages sent to find new infectable nodes serve as the parent's initial step in the parent-initiated handshake. If the node contacted is not yet a zombie and is infectable, after infection it will return a message representing the second step in the handshake. If this node already is a zombie, it will decide if it wants to add the contacting node as a parent. If so, it continues the three-way handshake. If not, it simply ignores the message.

A child-initiated handshake could happen when an already infected zombie wants to connect itself with more parent zombies, improving its efficacy of receiving new exploit code.

During the handshake, the parent and child exchange enough addressing information to be able to communicate again in the future. The child does not necessarily need to save information about its parents, but may do so if it wants to replace existing parents with better ones in the future. The parent and child might also exchange other useful information, such as hardware characteristics that will allow them to make better decisions in the future.

A three-way handshake is not strictly necessary; a two-way exchange is sufficient to form an overlay. In the parent-initiated case, the third message ensures that the child has an accurate parent list, which is important if the number of parents is deliberately limited. In the child-initiated case, the third message prevents a defender from overloading a parent with large numbers of false children at spoofed addresses by insisting that the child verify its ability to communicate.

Aggressive worms will contact nodes many times to see if they can be infected, so a zombie would expect to hear from many other zombies. It need not accept all of them as parents, and it need not choose to switch existing parents for more recently advertised ones. Instead, it will try to select "best" parents.

Parent selection has major goals, some conflicting:

- Ensure low latency for delivering exploit code (this roughly equates to minimizing the diameter of the network).
- Avoid causing network congestion during exploit delivery.
- Ensure a resilient overlay by choosing parents unlikely to be simultaneously disconnected.
- Provide the attacker with multiple injection points for exploit code.
- Avoid keeping long lists of other infected nodes.

Zombies could select parents simply by becoming the child of any Midgard zombie that contacts them. If the worm designer used a purely random algorithm for generating the IP addresses that are probed in a search for new victims, eventually every zombie would try to infect every other. In the long term, the overlay would stabilize to a completely connected graph, which would tend to lead to high congestion when attack code is to be disseminated. It would also make it easy for defenders to obtain a complete list of the overlay's members by inspecting one or a few infected machines.

Instead, the worm designer could specify that a particular number m of parents be maintained for each zombie. The first m unique infecting zombies that contact a newly infected machine could become its parents. Of course, if those m zombies are cleaned up or removed from the network, the child would become disconnected. Since other zombies are still contacting it with superfluous attempts to reinfect it, the child could use these infection attempts as liveness indicators, swapping in recently heard-from zombies for older parents. However, if this were the only method, a defender could disconnect zombies from the overlay by sending them false infection probes at a higher rate than the real worm. Leaving aside the possible congestion effects and questionable legality of this defense approach, the worm designer could cripple this defense by maintaining a small number of more permanent parents. For example, the zombie that actually infected the child could be kept

as one of the permanent parents. Whether the parent, the child, or both have the right to alter the selection is a matter of the design of a particular Midgard worm.

Since modern worms often search only a restricted portion of the IP address space, and may pay particular attention to the local network, the overlay network would have limited resiliency if the worm depended only on direct contacts with other zombies to form links. Another way to improve the resiliency would be to exchange a random subset of each worm's parent or child list each time a zombie contacts a previously infected machine. A further improvement would be to make such an exchange only when the two zombies are "far apart" in the IP address space, encouraging wide separation of linked nodes.

Unlike previous worms, Midgard worms have a motivation for maintaining links between already infected zombies, so it is likely that the designer of such a worm will use somewhat different techniques for generating IP addresses to probe for infectable machines.

Midgard worms could use arbitrarily sophisticated methods to select and maintain their parent lists. For example, they could use heartbeat messages to determine liveness of parents and children, or they could maintain path vectors to allow them to choose parents that offer greater resiliency [Li 02b]. Given that existing defense techniques would have a hard time handling even unsophisticated parent maintenance schemes (see Section 4 below), we omit further discussion of more complex approaches.

The above discussion is couched in terms of parents and children. Midgard worms could also be built as peer networks. Doing so would necessitate storing some information on both sides of the virtual connection between two zombie nodes, but they may need to store such information anyway.

3. Disseminating Code Through an Overlay Network

Once an overlay network has been set up as part of a Midgard worm's propagation, the attacker can inject code into the overlay to disseminate arbitrary programs. Depending on details, parents can push code to children, or children can pull it from their parents. A push-based design is advantageous because it doesn't require the children to engage in polling that might reveal a zombie's existence, so we will concentrate on that model here.

The attacker could choose to inject the code to be disseminated at a single point in the overlay network, or he could release it nearly simultaneously in multiple places. The attacker could either manually inject the code, or use an automated program at some other site to contact the injection points. Note that because every node in a Midgard worm's overlay network is highly likely to be an ultimate descendant of every other node in the network, any infected node is a feasible dissemination point. Barring interference, code injected at any node in a Midgard worm's overlay network will be propagated quickly to every other node in the overlay (Section 5.1).

Each injection point will then push the code to its immediate children which will in turn, push the code to its own direct descendants. Since each infected node has multiple parents, a zombie is likely to eventually receive a copy of the code from each of them. Only one copy is needed, so extra copies of the same code will be discarded, just like Usenet messages [Horton 87]. A careful attacker who wants to use his zombie overlay network repeatedly will include features to identify the most recently disseminated code, such as cryptographically signed version numbers or expiration dates. These features can also reduce congestion by suppressing transmission of duplicate or outdated exploits.

When a zombie receives a piece of code from a parent, it will check its authenticity. The simplest way to do so is to disseminate a public key with the original worm code. Each subsequently disseminated piece of attack code will come with a digital signature created with the matching private key, allowing each zombie node to check the authenticity and integrity of the code it receives using purely local operations. Authenticated code received for the first time will be executed (or perhaps scheduled for later execution). Duplicate authenticated code will be discarded. Improperly signed code will also be discarded, and possibly the parent node that sent it will be removed from the list of proper parents and replaced with another candidate.

Arbitrary sophistication could be added to this process. For example, the code could be individually encrypted for each child, using a key that is a function of the child's IP address, or each zombie's code could be slightly altered before encryption, using the same key for each zombie. While providing little secrecy, these measures would ensure

that every copy of the code flowing through the network is represented by different bits, making detection of the widespread dissemination of a single piece of code harder. Disconnected zombies could be aware of their disconnection and initiate a pulling process when they became reconnected, allowing them to find code that was disseminated during their absence [Li 02b].

Midgard worms need not guarantee that all of their zombies receive and execute the desired code, an important fact for defenders to remember. Cleaning up a few zombies does little harm to a Midgard worm, either in reduction of the size of its army or in affecting the connectivity of its overlay network (see Section 5.2). If a few nodes are deceived into not downloading the right code (perhaps by changing their local copy of the public key they use to check authenticity), no great damage is done to the worm's goal. As long as a large percentage of the nodes receive and execute the proper code, its purpose is served. Thus, the overlay network set up by a Midgard worm does not need sophisticated mechanisms to guarantee complete connectivity, assure delivery of code to all zombies, completely conceal their presence from knowledgeable observers, or prevent the insertion of a few imposter nodes set up by defenders. The aim is achieved as long as a substantial fraction of the infected machines is responsive to the attacker. Note that "substantial" does not necessarily mean "majority." As long as the absolute count of responding machines suits the attacker's needs, a Midgard worm will be successful.

4. Defending Against Midgard Worms

We can combat Midgard worms at various points in their life cycle. We can attempt to limit their spread; we can find and disinfect the zombies they have taken over; or we can insulate uninfected machines from the damage that the Midgard worm's zombies try to inflict.

4.1 Limiting the Spread of Midgard Worms

Midgard worms can spread using the same mechanisms as any other worm. Thus, the same defensive measures being developed to combat other worms will work equally well for Midgard worms [Twycross 03], and will not be discussed in greater detail here. Such mechanisms are currently of limited efficacy, and are not expected to be completely effective any time in the near future. So, at best, these mechanisms will cut down on the size of the zombie army recruited by a Midgard worm, but will not eliminate it.

4.2 Finding Midgard Worm Zombies

Again, many methods used to find the zombies created by other worms will work equally well for Midgard worms. However, Midgard worms do have some unique characteristics that may make it easier for defenders to detect their presence.

Searching for Listeners

To receive commands, Midgard worm zombies must listen on some socket, making them potentially detectable by a port scanner. However, deploying such a scan is of questionable ethics, and it could easily be defeated by a number of methods, such as only responding to authenticable probes, using a well-known and commonly open socket in a special way, only responding to probes from known parents, or only opening the socket at prearranged times. There are undoubtedly other ways to defeat a scan, so we do not believe that scanning will be an effective way to find zombies.

Searching for Heartbeats

A wisely pessimistic worm author must assume that the defenders will disinfect a large number of his zombies. To combat this, the author could have the zombies send heartbeats to ensure that their parents and children are still connected. In theory, it would be possible to sniff a network for these heartbeats and thus detect the infected machines. Heartbeats are not necessary if infected machines perpetually contact each other in redundant attempts to reinfect, but such attempts are just as noisy and characterizable as actual heartbeats.

However, as our analysis in Section 5.2 shows, if the attacker sets up a sufficient number of parents for each zombie and suitably randomizes the choice of parents, cleaning up even a high percentage of the zombies will tend to leave the others connected. Thus, heartbeats are not necessarily a good strategy for worm authors, and so a defense strategy based on their presence may be useless.

Traffic Analysis

After initial infection, the one time when a Midgard worm can be readily detected is when a second-wave attack is released. At that time, all zombies will come alive and will exchange data with their parents and children. Assuming that the second wave is a program, the total data involved could be large. This pattern will be very distinctive and obvious (not least because it is likely to cause major interference with legitimate Internet traffic). The attacker can use a trivial encryption scheme to alter the bit contents of the many messages being sent to disseminate his code, but he can do little to hide the fact that many machines are sending large amounts of data of similar size to each other quickly (different amount of padding data could be added to change the size, but this would often clog the network). He can also insert a random delay between a zombie's receipt of new code and when it is forwarded onward, but doing so will slow code propagation, and still will not completely conceal the signature of the transmission.

The existence of this pattern provides an opportunity for defenders. If a Midgard worm is known to have been released, it may be possible to deploy observation code at strategic points. The observers would look for the attack pattern and log the IP addresses of the hosts involved. This approach may be one of the most promising for developing a complete list of infected machines. Unfortunately, the defense may be too late because the attack will already have arrived at many of the zombies before any action can be taken. Only if detection of the second-wave attack and prevention of its propagation is sufficiently fast to keep the attack from reaching a critical mass of the zombies can this approach be considered effective.

Tracing the Overlay

Each Midgard zombie must maintain some information about the other zombies it connects to in the overlay. If that information can be extracted, then perhaps the discovery of one zombie can lead to the discovery of others. They, in turn, can be examined to find yet more zombies. The more highly interconnected the overlay is, the fewer zombies need to be examined to trace the entire overlay.

There are two serious problems with this approach to finding zombies. First, any single defender will only be able to look at a relatively small set of machines that the he has explicit permission to access, since it is generally illegal to examine someone else's infected machine for this information. In most cases, an attempt to trace the overlay will quickly lead to an unexaminable point that terminates the trace.

Second, clever worm designers can obscure or even encrypt information about their overlay. Since the information is only required at the time of second-wave code dissemination, it can be hidden at other times. If it is encrypted, the key can be included as part of the second-wave code, so defenders will be unable to extract such information at any other time.

However, since untalented worm designers are less likely to include such mechanisms and might well code them badly even if they are included, the information might not be obscured in all circumstances, and thus this approach should not be rejected *a priori*.

Having Owners Look for the Worm

Historically, the most effective way to discover infected machines has been for their owners to examine them. That is likely to be equally true for Midgard worms. Unfortunately, it is difficult to motivate large numbers of owners to perform the necessary search and repair.

4.3 Zombie Disinfection

Cleaning the zombie machines that a worm has infected is the preferable way to stop it from doing further damage. Unfortunately, past experience suggests that while many owners of infected machines are eager to disinfect them as soon as the problem is discovered or reported to them, a significant number of them are either unaware of their problem, unable to perform disinfection, or unconcerned by the infection. Since in general only authorized people have the right to make the alterations necessary to remove a worm from a machine, zombies identified by unauthorized people are less likely to be disinfected. There are still large numbers of machines infected with Code Red years after its introduction to the Internet [Chun 03, Costello 02]. Regardless of this problem, any achievable degree of disinfection reduces the damage a Midgard worm can do and is thus desirable.

The disinfection of a machine hosting a Midgard zombie will be similar to cleaning out any other noxious code. The ease of disinfection will depend primarily on the sophistication of the author at concealing his changes and entwining his malicious code into various parts of the system. Since Midgard

zombies might contain hints regarding the identity of other zombies, it would be desirable to look for those during disinfection, but otherwise the process will be similar to handling other malicious code infections.

Some defenders have attempted to use worm techniques to combat worms. In essence, they have sent a worm to kill a worm. Since a Midgard worm maintains an overlay network to distribute code, if one could hijack that network, one could distribute code that cleans out the worm and closes the door behind itself as it moves on to the next zombie node. While this idea is seductive, it has several serious flaws – legal, ethical, and practical [Schneier 03]. Further, for Midgard worms, a sufficiently clever worm author can make it infeasible to use his overlay by deploying the same anti-tracing techniques discussed in Section 4.2. He can also require authentication of any code passed through the overlay, which will be hard or impossible for the author of the anti-worm to provide. On the whole, this approach to cleaning up after a Midgard worm seems infeasible.

4.4 Protecting Uninfected Machines From Worm Zombies

Unless technologies for stopping the spread of worms become extremely successful or the worldwide culture of system administration changes dramatically, we must expect that a well-designed Midgard worm will be able to maintain a connected army of zombies more or less indefinitely. At any moment, the owner of this army can release a new piece of code into his overlay. The unique problem brought up by Midgard worms is that this arbitrary code need not be limited to causing problems for the infected machines. In fact, it probably *won't* cause them problems. Instead, it will most likely be targeted at machines that are not infected with the worm.

Minimizing the number of machines initially infected and maximizing the number disinfected will lessen but not eliminate the threat. Having your own machines protected from the widest array of attacks possible will make it less likely that the Midgard master's new code can cause you trouble, but will not protect against new attacks you don't know about or attacks that closely resemble your ordinary traffic.

If we assume that most or all of the zombies can be identified, there is at least one other defense possible. Since we cannot predict what the attacker will do with his zombie army, those wishing to protect themselves must assume that anything sent by a known zombie node represents an attack, regardless of its character or content. Thus, any site worried about the possible harm emanating from a known Midgard overlay network can choose to *shun* all traffic coming from any infected site by ignoring every packet sent from that machine. Packets can be dropped at firewalls or other points in one's network, as convenient, but all such packets must be dropped before they have a chance to do their mischief. Of course, if the real owner of the zombie machine is also sending perfectly legitimate traffic to your protected network, that traffic will also be dropped by shunning. One must choose between protection at the cost of losing some legitimate traffic, or handling legitimate traffic at the cost of losing protection.

Shunning in one's local network is only somewhat effective, since it must be based on observing the IP address of the packets sent by a zombie. If some zombies remain unidentified, shunning will miss packets sent by the unidentified ones. If the attack is of a class that can use IP spoofing to conceal the identity of the sender, and indeed uses it, shunning will provide no protection. Shunning needs to be used in conjunction with technologies that combat IP spoofing [Li 02a, Park 01, Savage 00] or with technologies that combat the classes of attacks that can be made with spoofed packets [CERT CA-1996-21, Mirkovic 02, Ferguson 00]. There are also issues involved with dynamic IP addresses or NAT that would need to be handled, which might well require shunning entire blocks of addresses related to the zombie's observed address. Unless a finer distinction is possible to pinpoint the address in the block currently assigned to the infected machine, many might find this cure worse than the disease.

One could use shunning more selectively on packets sent by known zombies. Any packets deemed to be particularly safe could be passed from such addresses, or all such packets could be passed through careful scrutiny of an automated security system before they are delivered. However, if the attacker is exercising a zero-day vulnerability, there is a possibility that these measures of extra care could be insufficient. Only shunning all of the zombie's traffic can provide maximum safety.

It is not trivial to enable nodes and subnetworks to shun a Midgard worm's zombie army. Thus, we will not discuss shunning techniques in detail here. Shunning should be regarded as a possible research

approach that might help handle the threat, rather than a solution ready for deployment and use. We plan to examine the use of shunning to protect against Midgard worms in future research. Shunning could be deployed either at the edges of the Internet or at its core. Edge deployments are easier to achieve, but protect only the edge networks that deploy them, and not the core at all. Core deployments are much harder to achieve, and face greater technical difficulties, but provide wide coverage of many systems for relatively few deployment and maintenance points. Both options should be investigated.

5. Performance Analysis

A Midgard worm equips an attacker with a special resilient overlay network for injecting arbitrary exploit code to all the zombies on the overlay. In this section, we analyze how fast a Midgard worm can disseminate exploit code, the resiliency of its overlay, and the traffic volume required for Midgard worms to function.

We have done a simulation study in which a Midgard worm uses random scanning to infect new zombies, and eventually every zombie will have exactly m zombie parents. Note that because of Midgard's self-upgrading capability, a Midgard worm can always adopt the most efficient and/or resilient overlay, implying that we may not know exactly what such an overlay would look like from day to day. Our results thus should be regarded as only a lower bound on the performance of Midgard.

5.1 Speed

In the following, we estimate the latency that Midgard would take to deliver exploit code to a zombie. We assume every zombie has two parents (m = 2) to receive exploit code, and will then forward the code to another three child zombies. (Every zombie will then have a total of five neighbors. "Five" here is a typical number; for example, [Staniford 02] reports the average node degree to be 4~5.5 in a 1M worm network formed through permutation scanning.)

Midgard's dissemination of exploit code can be divided into three basic latency components: the processing and queuing delay at each hop, the transmission delay of crossing the wire, and the kernel-space-crossing delay. (Note that a large amount of exploit code or a densely connected Midgard overlay could cause congestion effects and slow down the dissemination.)

Initially, we assume that the exploit code can be encapsulated in a single packet.

First, because the exploit code is delivered hop by hop on an overlay,

$$\textit{Average delivery latency} = \textit{average latency per hop} * \textit{average delivery hop count} \quad (1)$$

Second, denote W as the average wire communication latency per overlay hop, P as the average processing delay at a zombie, and K as the average kernel-space-crossing delay. Then

$$\textit{Average latency per hop} = W + P + K \quad (2)$$

W is actually the point-to-point transmission delay over the Internet. One could conservatively assume that it would take less than 250 ms to deliver a packet across the Internet [Fei 98, Biagioni 00, Internetpulse 03]. Thus,

$$W < 250\ ms \quad (3)$$

Further, we have measured P and K in our test bed, where each machine is equipped with an AMD Thunderbird 1.333 GHz CPU, 1.5GB SDRAM, and a 100 Mbps Ethernet interface. With three nodes to reach, we have empirically found that with 95% confidence,

$$P = 1.30\ ms \quad (4)$$

$$K = 0.67\ ms \quad (5)$$

From (2)-(5),

$$\textit{Average latency per hop} < 250+1.30+0.67 \approx 252\ ms \quad (6)$$

Third, we analyze the delivery hop count. Figure 1 shows that the maximum hop count between any two zombies closely follows a logarithmic trend with respect to the number of zombies. If we extend the trend to the scale of one millions zombies, we will obtain a value of 35 hops. Thus,

$$\textit{Average delivery hop count} <= 35 \quad (7)$$

From (1), (6), (7), for our worst-case,

$$\textit{Average delivery latency} < 8820\ ms \approx 8.8\ secs \quad (8)$$

If the exploit code is large, it will require multiple packets. A 1-megabyte exploit would need about 700 IP packets (MTU=1500 bytes). Noting that the

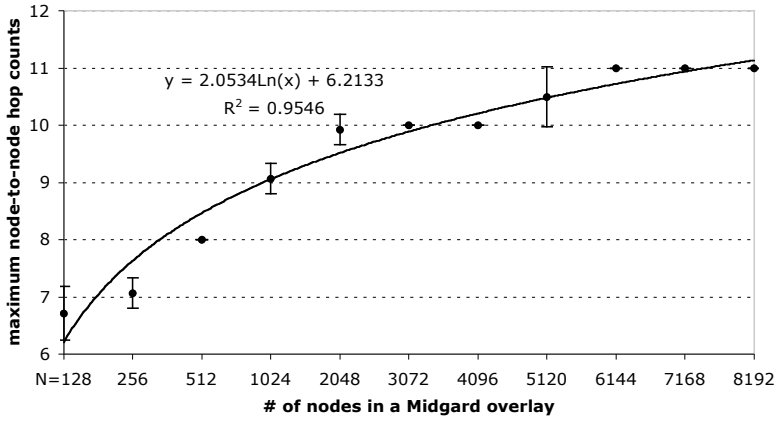

Fig. 1. Maximum hop counts between zombies on a Midgard overlay (confidence level 95%).

dissemination can be pipelined (a zombie can immediately start transmitting the next physical packet after the previous one goes to the wire), from (3) and (8),

Average delivery latency <
*8820 ms + (700-1)*250 ms ≈ 3.1 minutes* (9)

Congestion caused by large amount of exploit code in transit, however, could happen in some cases and slow down the exploit code delivery. For instance, if a zombie receives a 1-megabyte exploit from two parents, P and Q, the worst case is when receiving 700 packets from each parent in the order of following:

$P_1, Q_1, P_2, Q_2, \ldots, P_i, Q_i, \ldots, P_{700}, Q_{700}$
(P_i, Q_i are the ith packet from P and Q, respectively).

The zombie will not have a complete copy of the exploit until it receives P_{700}, the 1399th packets in the sequence, resulting in an almost doubled delivery latency. If this happens at every hop, (9) will become

Average delivery latency <
*8820 ms + (700-1)**500 ms ≈ 6.0 minutes* (10)

This is still very fast! But one can imagine that if every zombie has dozens or hundreds of parents and receives exploits from all of them simultaneously, the latency can become very high. Midgard can introduce congestion avoidance schemes to solve this problem. An easy one would be for the parent of a zombie to send an "I have the update" message rather than the update itself. That way, the zombie could just request the update from a single parent, minimizing congestion while only slightly increasing latency.

In the above extremely conservative analysis, the dissemination time is well under ten minutes. Even if a Midgard worm spreads an order of magnitude slower than we calculated, a new exploit can still be activated in under an hour, particularly since a Midgard worm does not much care about how long it takes to reach every last zombie. It only needs to get to enough zombies to do harm, which is certain to take less time than the above estimate.

5.2 Resiliency

Given a zombie overlay with N nodes, our simulation study of the resiliency of a Midgard overlay aims to answer the following questions:

(1) For a given zombie overlay, if x nodes are randomly disinfected,[2] what percentage of remaining nodes will remain connected? Here, we assume that the Midgard worm does not attempt to reconnect any nodes after the disinfection phase.

(2) What is the impact of a Midgard overlay's size on its resiliency?

In the following, we use the word "overlay" and "graph" interchangeably. We use standard graph terminology: two nodes (zombies) are *reachable* from each other if there is an undirected path between them. A graph is *connected* if any node is reachable from any other; otherwise, the graph consists of

[2] Ideally, the defenders would choose an optimal set of nodes to disinfect. However, we believe that finding the optimal set is NP-hard. Also, the defenders are likely to disinfect every node they can, rather than wasting time choosing a subset, so random disinfection is more reflective of reality.

several *partitions*, with each partition a connected sub-graph. Finally, a graph's *maximal reachability* is the number of nodes that belong to the largest partition of the graph; we express this as a percentage of the total number of nodes.

Clearly, if the exploit code injection point is in the largest partition, the maximal reachability of a Midgard overlay indicates the percentage of zombies that are reachable by the attacker, and thus is a good metric for measuring the resiliency level of a Midgard overlay. If the maximal reachability is large, it is also very probable that the injection point will be in that partition. (Note that in reality the attacker is not limited to a single injection point, so that even if the overlay has been partitioned, the exploit could be inserted into multiple partitions.)

To answer question (1), in our simulation we randomly cut a certain number of nodes from a graph and then remeasured the maximum reachability of the new graph. This emulates the disinfection of zombie nodes. Here, we denote x as the number of nodes that were cut from the graph.

For every different value of m, we tested twenty 10000-node graphs; for every graph, we tested different values of x; and ten different random cuts were measured for each value of x. Figure 2 shows the maximum reachabilities for those graphs with x nodes cut. For example, when 1024 nodes are randomly cut from a 10000-node Midgard overlay where every zombie has one parent, the maximum reachability will decrease from 100% to roughly 43.30%; but if every zombie has two parents, the maximum reachability will only drop from 100% to 99.80%. (In these tests, we assumed that exploits could also be delivered from a child to a parent if necessary.)

While it is clear that a higher value of m leads to a higher maximum reachability, Figure 2 further shows that a small value of m can already lead to high reachability, and thus resiliency, in a Midgard overlay. The most cost-effective value of m is probably 2, especially if the worm author expects that at most one-third of the zombie nodes might be disinfected. Given that a real Midgard worm could reconnect itself over time, having two zombie parents per zombie clearly already implies a frightening level of resiliency.

Figure 2 also shows that if an attempt to disassemble a Midgard worm's overlay is to succeed, more than 70% of its nodes must be disinfected when every zombie has two parents and more than 80% disinfected when there are three parents – a challenging task for any defender. Even worse, with five parents per zombie, disinfecting 80% of the zombie nodes still leaves about 1400 zombies connected.

To answer question (2), we also studied the impact on resiliency of the size of a Midgard overlay and found that it is minimal. Specifically, we tested Midgard overlays with a constant value of m (m=3) but different size N, where N=1024 to 16384 in powers of two. Furthermore, for each N, we tested twenty overlays; for each overlay, we examined a different percentage of cut nodes; and for each

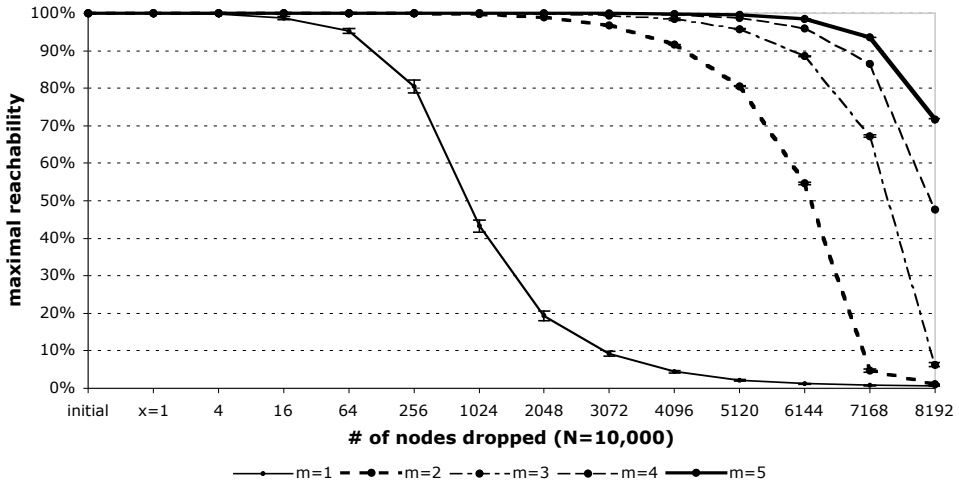

Fig. 2. Maximum reachability decrease vs. zombie node disinfection (confidence level 95%).

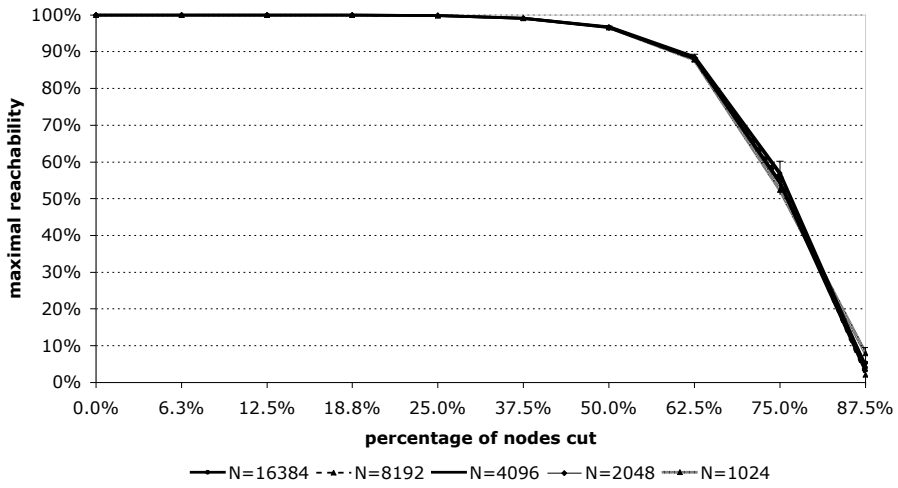

Figure 3. Maximal reachability for Midgard overlays with different size (m=3) (confidence level 95%).

percentage, we measured ten different random cuts. The results are shown in Figure 3 in five curves, where each curve corresponds to a different overlay size and shows the maximum reachability of a graph versus the percentage of nodes cut from the initial graph. These curves are almost the same, indicating the low impact of overlay size. This implies that even Midgard worms that infect a relative modest number of nodes, say 1000, can still build overlays that are extremely hard to disconnect.

In summary, the overlay of a Midgard worm can be easily created using random scanning, but can still achieve high resiliency as well as full connectivity when every zombie has more than one parent. We also found that the size of an overlay has minimal impact on the resiliency.

5.3 Traffic Generated by a Midgard Worm

There are several types of traffic costs that a Midgard worm incurs: the volume of disseminated exploit code (dissemination volume), the volume needed for a zombie to join (join volume), and the ongoing volume for maintaining the overlay network (maintenance volume). The designer of the Midgard worm does not directly care about traffic costs, since he isn't paying for the bandwidth he is using and cares little about any undesirable effect that bandwidth consumption may have on his zombies or the networks they attach to. However, heavy use of bandwidth is at least a signal of his worm's activities, and gross overuse might cause congestion that actually slows dissemination of the attack code. So a Midgard worm designer is likely to pay only modest attention to bandwidth requirements.

Our analysis makes the reasonable assumption that every node has five parents.

Dissemination volume. In a single round of dissemination, the inbound dissemination volume per zombie is the size of the exploit code multiplied by the number of parents that forward the exploit. The total amount of volume produced by a Midgard worm is then the volume per zombie multiplied by the total number of zombies. For example, if the payload is 1 kilobyte, every zombie has five parents for forwarding the exploit code, and there are 1 million zombies in all, the total amount of data to be disseminated across the network will be 5 gigabytes. Given that the traffic would be distributed across the entire Internet, this is large but tolerable.

Join volume. A child-initiated three-way handshake normally requires less than 100 bytes of payload. Supposing that a zombie becomes a child of five parents through five successful child-initiated three-way handshakes, then the total join volume for that child is less than 500 bytes. A million nodes would then generate 500 megabytes of aggregate join traffic. Unless all the zombies join within a very short time, the join volume is insignificant.

Maintenance volume. If heartbeat messages are used, if every zombie has five parents, and if every heartbeat is 100 bytes, an overlay with 1 million nodes will then incur 500 megabytes per period, which doubles if the heartbeat is bidirectional (1 gigabyte). Since zombies are scattered all across the

network and are likely to spread their heartbeat traffic over time, unless the period is very short the 500 megabytes will be sufficiently spread both temporally and topologically as to cause little trouble for anyone. Nevertheless, this quantity of data may be sufficient to allow observation of the worm's behavior. Remember, however, that Midgard worms might not need heartbeats at all, in which case there is no maintenance traffic.

6. Existing Worms and Other Related Systems

Many worms and other malicious programs have already been released that exhibit some of the features of a Midgard worm. This observation suggests that building a Midgard worm is merely a matter of some attacker putting all the pieces together. We show here that many of the components of a workable Midgard worm have already been seen "in the wild," and thus it is only a matter of time until a true Midgard worm appears.

A Midgard worm propagates just as any other worm. It can propagate at the same high speed as the Slammer worm [Moore 03] or a Warhol worm [Staniford 02]. But after a Midgard worm builds its own overlay network during propagation, it can disseminate second-wave attack exploit code even faster, since the code does not have to be executed right away and every zombie can simply forward the code to its children without scanning.

Although numerous worms have caused many kinds of damage, most worms still employ relatively simple approaches for infection and propagation, often hard-coded with predefined attacks. For instance, the Blaster worm [CERT CA-2003-20] and Code-Red [CERT CA-2001-19] both tried to launch DDoS attacks toward well-known web sites (Microsoft's windowsupdate.com and www1.whitehouse.gov)., Midgard worms can be designed for flexibility; unlike the current crop of worms, they will also be able to upgrade themselves by disseminating improved versions of themselves to zombie nodes.

A few worms have shown some sophistication. CodeRedII [CERT IN-2001-09] installs a backdoor on every infected machine to allow root-level control on individual computers. The Nachi worm [Symantec 03] attempts to download the Blaster worm patch from Microsoft's windows update web site. Sobig.F provides 20 sites for zombies to download arbitrary exploit code. [Staniford 02] also talks about self-coordinated scanning to infect new machines. Midgard worms will display a greater level of sophistication. While zombies from earlier worms are often self-contained, Midgard zombies will self-organize themselves into a resilient overlay, a fundamental change in allowing sophisticated zombie coordination. Compared to CodeRedII, a Midgard controller will not have to log on to every zombie as root to attack the machine, but instead insert the new code at a single node, with automated dissemination to the entire army. In addition to the advantages of parallelism, the controller need not know the identity of all the zombies that will receive the update code. Sobig.F only used 20 specific nodes for code dissemination points, but Midgard worms will allow injection at any point in the network. Defenders stopped Sobig.F's update dissemination by shutting down 20 sites. A Midgard worm's resilient overlay will effectively deliver code even if thousands of its zombies are shut down.

Other malicious attacks show some control sophistication. For example, the *trinoo* distributed denial-of-service attack tool builds a simple three-layer trinoo network [CERT IN-99-07] on which the attacker(s) control one or more "master" servers, each master controls many "daemons," and the daemons are all instructed to coordinate a packet-based attack against one or more victim systems. Though similar to Midgard's zombie overlay network, a trinoo network is used for a special-purpose attack (DDoS) and does not offer the resiliency that a Midgard overlay provides.

Midgard worms also share some features with legitimate distribution mechanisms, such as broadcasting, multicasting, content-delivery networks, event notification, virus signature distribution, automatic update facilities, etc. Midgard worms are especially similar to legitimate delivery services that are based on overlay networks. There are tree-structured overlays for delivery service, often regarded as application-level multicast, such as Yoid [Francis 00], ALMI [Pendarakis 01], End System Multicast [Chu 00], Scattercast [Chawathe 00], Bayeux [Zhuang 01], Overcast [Jannotti 00], etc. There are also non-tree-structured overlays; for example, Bullet [Kostic 03] provides high-bandwidth data dissemination through an overlay mesh, and Revere [Li 02b] supports large-scale security update delivery through resilient self-organized overlay networks. Although the overlay of a Midgard worm could be organized as a tree, for resiliency reasons this would be a poor choice: as in Bullet or Revere,

every node can have multiple parents. However, unlike Bullet and Revere, which assume that the delivery overlay has a root (often the dissemination source), a Midgard worm can try to inject exploit code from any arbitrary point and typically adopts a different approach for its overlay's resiliency.

7. Conclusions

So far, the Internet community has been fortunate that worms have not been more malicious and dangerous. However, our luck is likely to run out very soon. As hackers and criminals discover the advantages of having large multipurpose armies of zombie machines, we can be sure worms will be created to enlist such armies. This paper demonstrates for the first time how trivial it would be for a zombie army to be directed to new targets using new attacks. We have shown that within six minutes, the general of a zombie army can realign them to a new purpose and use them for whatever malice he has in mind.

As our results also show, once the resilient overlay for carrying orders to such an army has been established, disconnecting it is extremely difficult. With a perfectly reasonable connectivity of five parents per zombie node, cleaning up over 80% of a 10,000-node zombie army still leaves the general with a connected overlay of 1400 zombies. Anything far short of 100% disinfection will allow the remaining zombies to work in concert. If the original army contained 1 million zombies, 80% disinfection leaves a useful connected army of 140,000 zombies, easily enough to perpetrate a DDoS attack on almost any site, generate enough spam to drown the Internet, or launch a new penetration attack on almost every node in the Internet before human beings have time to respond or even notice.

Since Midgard worms are, at the initial infection stage, just like any other worm, the community's primary line of defense against them will depend on creating better mechanisms to detect the appearance of new worms early in their life cycle and slowing the spread of worms that have been detected. This research merely reinforces the absolute necessity of finding better methods of detecting, slowing, and ultimately stopping worms. Any efforts spent on disinfection through normal or newly discovered methods will at least cut down on the size of the zombie army, even if they don't prevent surviving zombies from receiving new instructions.

But nothing suggests we can change the existing culture that allows many infected machines to go unpatched and remain infected for years. So we must assume that sooner or later some attacker will build a large Midgard worm overlay and that we will have to live with the consequences for the foreseeable future. Those consequences are that the owner of this overlay can enlist large numbers of machines to execute arbitrary code intended to perform as-yet-unknown malice against targets of his choice. The extortion potential is staggering.

So something must be done. Barring near-perfect defense mechanisms to stop the spread of worms and a highly unlikely worldwide enlightenment of computer users to aggressively patch and disinfect machines, our best defense seems to be to detect which nodes are infected and prevent them from spewing their arbitrary poison on the rest of the network. Doing so will not be easy, but we recommend an immediate research program into the feasibility of such an approach, applied both to edge networks that wish to protect themselves and core networks in a position to protect the entire community. If this research is not initiated soon, we will suffer the damaging effects of a Midgard worm before any effective defense is available.

Bibliography

[Biagioni 00] E. Biagioni, P. Hinely, C. Liu, and X. Wang, " Internet Size Measurements," 2000. Available at http://citeseer.nj.nec.com/biagioni00internet.html

[CERT CA-2003-20] Computer Emergency Response Team, "CERT® Advisory CA-2003-20 W32/Blaster Worm," August 2003. Available at http://www.cert.org/advisories/CA-2003-20.html

[CERT IN-2003-03] Computer Emergency Response Team, "CERT® Incident Note IN-2003-03 W32/Sobig.F Worm," August 2003. Available at http://www.cert.org/incident_notes/IN-2003-03.html

[CERT CA-2001-19] Computer Emergency Response Team, "CERT® Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL," July 2001. Available at http://www.cert.org/advisories/CA-2001-19.html

[CERT IN-2001-09] Computer Emergency Response Team, "CERT® Incident Note IN-2001-09 "Code Red II:" Another Worm Exploiting Buffer Overflow In

IIS Indexing Service DLL," August 2001. Available at http://www.cert.org/incident_notes/IN-2001-09.html

[CERT IN-99-07] Computer Emergency Response Team, "CERT® Incident Note IN-99-07 Distributed Denial of Service Tools," January 2001. Available at http://www.cert.org/incident_notes/IN-99-07.html

[CERT CA-1996-21] Computer Emergency Response Team, "CERT® Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks," September 1996. Available at http://www.cert.org/advisories/CA-1996-21.html

[Chawathe 00] Y. Chawathe, "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service," Ph.D. thesis, Elec. Eng. Comput. Sci. Dept., Univ. California, Berkeley, CA, Dec. 2000.

[Chu 00] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," *Proceedings of ACM Sigmetrics*, June 2000.

[Chun 03] B. Chun, J. Lee, and H. Weatherspoon, "Netbait: a Distributed Worm Detection Service," Unpublished manuscript. February 2003. Available at http://netbait.planet-lab.org

[Costello 02] S. Costello, "Nimda, Code Red Still Crawling, Threatening the Net," *IDG News Service*, May 2002. Available at http://www.nwfusion.com/news/2002/0506code.html

[Fei 98] A. Fei, G. Pei, R. Liu, and L. Zhang, "Measurements on Delay and Hop-Count of the Internet," *IEEE GLOBECOM'98 - Internet Mini-Conference,* 1998.

[Ferguson 00] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," *Internet RFC 2827*, May 2000.

[Francis 00] P. Francis, Yoid: Your Own Internet Distribution, April 2000. Available: http://www.aciri.org/yoid

[Horton 87] M. Horton and R. Adams, "Standard for Interchange of USENET Messages," *Internet RFC 1036*, December 1987.

[Jannotti 00] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole Jr, "Overcast: Reliable Multicasting with an Overlay Network," *OSDI* 2000.

[Internetpulse 03] The Internet Health Report. Available at http://www.internetpulse.net/.

[Kostic 03] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *SOSP* 2003.

[Li 02a] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol," *INFOCOM 2003.*

[Li 02b] J. Li, P. Reiher, and G. Popek, Disseminating Security Updates at Internet Scale, *Kluwer Academic Publishers,* November 2002. ISBN 1-4020-7305-4.

[Mirkovic 02] J. Mirkovic, J. Martin, and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," UCLA CSD Technical Report 020018, 2002.

[Moore 03] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, vol. 1, no. 4, July/August 2003.

[Park 01] K. Park and H. Lee. "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," *Proceedings of ACM SIGCOMM 2001.*

[Pendarakis 01] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," *Proceedings of the 3rd Usenix Symposium on Internet Technologies & Systems (USITS)*, March 2001.

[Savage 00] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," *Proceedings of ACM SIGCOMM 2000.*

[Schneier 03] B. Schneier, "Benevolent Worms," Cryptogram newsletter, September 15, 2003, http://www.schneier.com/crypto-gram-0309.html

[Spafford 89] E. Spafford, "The Internet Worm: Crisis and Aftermath," *Communications of the ACM*, vol. 32, no. 6, June 1989.

[Staniford 02] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in Your Spare Time," *11th Usenex Security Symposium*, 2002.

[Symantec 03] Symantec, "W32.Welchia.Worm," August 2003. Available at http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html

[Twycross 03] J. Twycross and M. Williamson, "Implementing and Testing a Virus Throttle," *12th Usenix Security Symposium*, August 2003.

[Zhuang 01] S. Zhuang, B. Zhao, A. Joseph, R. Kata, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination," *Proceedings of NOSSDAV*, 2001.