The Worm FAQ: Frequently Asked Questions on Worms and Worm Containment

Stuart Staniford
The Worm Information Center
www.networm.org
stuart@silicondefense.com

Contents

1. Administrativia 3
   1.1. Who wrote this FAQ? Who is Stuart Staniford anyway? 3
   1.2. Where do I send suggestions, complaints, etc? 3
   1.3. When was this document last updated? 3
   1.4. What should I do if I don’t know enough security terminology to understand this document? 3

2. Worm Basics 3
   2.1. What is a worm? 3
   2.2. Is a worm the same as a virus? 3
   2.3. Where did the term worm come from? 4
   2.4. What are some famous worms? 4
   2.5. What are some famous things that aren’t worms? 4
   2.6. Where can I find out more about viruses? 4
   2.7. What payloads can a worm have? 4
   2.8. What papers should I read to find out more about worms? 5

3. How Worms Spread 5
   3.1. What is a random scanning worm? 5
   3.2. What is saturation of a worm? 5
   3.3. What is subnet scanning? 6
   3.4. What is a Warhol worm? 6
   3.5. What is co-ordinated permutation scanning then? 6
   3.6. What is a flash worm? 6
   3.7. What is a topological Worm? 7
   3.8. What is a metaserver worm? 7
   3.9. What is firewall penetration? 7

4. Famous Worms 7
   4.1. Please Grandad, tell me about the original Internet Worm. 7
   4.2. What happened with Code Red? 8
   4.3. What happened with Nimda? 9
   4.4. What happened with Slammer? 9
   4.5. TCP worms couldn’t be nearly as fast as Slammer, right? 9
   4.6. What happened with Blaster and Welchia? 10
5. The Future of Worms

5.1. How fast could a worm compromise the Internet? .................................................. 10
5.2. How fast could a worm compromise my enterprise? .................................................. 10
5.3. How long can worms last? ......................................................................................... 10
5.4. Are the worms to date any good? .............................................................................. 11
5.5. Why do people write worms? .................................................................................... 11
5.6. How hard is it to write a worm? Do you need a BS/CS or a PhD? .............................. 11
5.7. I want to write a worm. Can you help me? ................................................................. 11
5.8. Is it legal for me to launch a good worm? ................................................................. 11
5.9. How much do worms cost society? .......................................................................... 12
5.10. What is the Uberworm? How bad could a worm be? .............................................. 12
5.11. Is the power grid vulnerable to worms? ................................................................. 12
5.12. Is Al Qaeda writing worms to destroy civilization? ................................................. 13
5.13. Is Country X writing worms to destroy civilization? ............................................... 13
5.14. Why would a country release a worm - wouldn’t it hurt them just as much? ............ 13
5.15. Aaagh - what can I do to protect my critically important network? ...................... 13
5.16. I’m a journalist or a policymaker. What are some ideas for solving the worm problem? 13

6. Worm Containment

6.1. What is worm containment? .................................................................................. 14
6.2. Shouldn’t the vendors just fix their software and the problem would go away? .......... 14
6.3. Doesn’t anti-virus software do worm containment? ................................................. 15
6.4. Do firewalls help with worms? Are they enough? ..................................................... 15
6.5. What about internal firewalls? Router/Switch ACLs? .............................................. 15
6.6. What about intrusion detection systems? Intrusion prevention? .............................. 15
6.7. What are some ways worms can get inside my enterprise? ...................................... 15
6.8. Vendor ABC told me their system would prevent all worms entering my network. What should I think? ................................................................. 16
6.9. What is this vulnerability density you keep talking about? ....................................... 16
6.10. What is the epidemic threshold for a containment system? .................................... 16
6.11. Can scanning worms be contained? .................................................................... 17
6.12. So what’s the bad news? ....................................................................................... 17
6.13. Is it better to do worm containment on end systems or in network devices? ......... 17
6.14. What’s a cell? ....................................................................................................... 18
6.15. What about host intrusion prevention systems - do those contain worms? .......... 18
6.16. Are there products that can help with this? ......................................................... 18
6.17. I should concentrate my worm containment systems in front of key servers, right? 19
6.18. So what are some places I should concentrate my worm containment systems? .... 19
6.19. Can’t I just have a single IDS on my network that tells switches to turn off ports, and contain worms that way? ................................................................. 19
6.20. How do I design enterprise deployment of worm containment systems? ............... 20
6.21. How do I know my containment system will really work? I don’t want to loose a real worm on my network to test it? ................................................................. 20
6.22. Worm containment is too complicated. Is there an alternative? ............................. 20
6.23. What’s the prospect for worm containment on the Internet itself? ......................... 21
6.24. What’s a network telescope? ................................................................................... 21
6.25. I want to study my MS/PhD thesis on worms/worm containment. Where should I study? ................................................................. 22
6.26. What papers should I read to find out more about worm containment? ............... 22
1. Administrativia

1.1. Who wrote this FAQ? Who is Stuart Staniford anyway?

This FAQ was written by Stuart Staniford, president of Silicon Defense. Silicon Defense is an innovative Internet Security firm that sells worm containment solutions and does research on worms and worm containment. Stuart is an expert on worm spread and worm containment who has authored a number of widely cited research papers on the subject.

The FAQ covers worms and worm containment. While they represent our opinionated view of things, we try to keep them free of sales pitch and just give useful information. We even mention other vendors products favorably! Silicon Defense has a product suite, CounterMalice that does worm containment, and you can go to the CounterMalice web page if you would like same sales pitch ;-) 

1.2. Where do I send suggestions, complaints, etc?

stuart@silicondefense.com

1.3. When was this document last updated?

October 15th, 2003

1.4. What should I do if I don’t know enough security terminology to understand this document?

Following at least some portions of this document will require a general sense of how the Internet works, and a rough understanding of network security. Eg. we throw around terms like ”port”, ”IP address”, ”exploit”, ”vulnerability”, or ”syn packet” fairly freely. If this is gobbledegook to you, you might try this over view of Internet Security. We don’t assume you know much about worms.

2. Worm Basics

2.1. What is a worm?

A Worm is a computer program which, when it runs, finds other computers that are vulnerable and breaks into them across the network. It then copies itself over, starts itself running on the new hosts, and does the same thing from there. Thus it can spread exponentially like an epidemic of human disease, or a nuclear chain reaction amongst fissionable atoms.

The worm has several important aspects - a spread algorithm for finding other hosts, one or more exploits allowing it to break into other computers remotely, and a payload, which is what it does to your computer after it’s broken into it, other than just using it to spread.

2.2. Is a worm the same as a virus?

No. However they are both malicious code that propagates around the network. The boundary between worms and viruses is a little gray, and there is not a consensus in the security industry on where it lies. For the purpose of this FAQ, we define the difference as follows

- If the malicious code can break into another computer and start itself running there immediately with no human intervention, then it’s a worm.

- If the malicious code gets carried around in some other content and then may or may not start running on other computers depending on when and whether humans decide to process that content, then it’s a virus.

In short, we make the distinction based on whether or not the malicious code is self activating. By this definition, Code Red, Slammer, and Blaster are worms. I Love You and SoBig are viruses. Nimda had both viral and worm spread algorithms.
From an operational perspective, the biggest difference is that worms can spread significantly faster, which has strong implications for defenses against them. Viruses are more common, however. By and large, existing anti-viral defenses are adequate against viruses as long as people deploy and update them properly (of course, they don’t always do this). However, antiviral defenses are fairly useless against worms (at least during the initial spread of the worm).

2.3. Where did the term worm come from?

It was coined by researchers at Xerox Parc who used benign worms to do system maintenance tasks. They were apparently inspired by a John Brunner novel “The Shockwave Rider” which featured a “tapeworm” program.

2.4. What are some famous worms?

The Internet Worm of 1988 put worms on the map by disrupting the Internet for several days, and overloading many systems. This was back when there were only 60,000 hosts on the Internet.

The modern era of worm research began with Code Red in 2001, a rapidly spreading worm which exploited a vulnerability in Microsoft’s IIS Web Server. This was followed by Nimda, a highly sophisticated worm/virus that spread very rapidly through multiple modes and was the first worm to have a dedicated firewall tunneling capabilities.

In 2003, we had the Slammer worm, a tiny single packet UDP worm on Microsoft’s SQL server. It is the fastest to date, with an early doubling time of less than seconds. Later in the year was the Blaster worm, followed by the Welchia worm which attempted to fix the vulnerabilities used by Blaster but caused more chaos.

There have been many other worms of lesser significance which we don’t note here.

2.5. What are some famous things that aren’t worms?

The Eiffel Tower? Also, viruses such as Chernobyl, Melissa, I Love You, SirCam, Klez, and SoBig.

2.6. Where can I find out more about viruses?

You might try the excellent Virus Bulletin. Also, the anti-virus vendors make lots of good information publically available. Symantec, Network Associates, Trend Micro, and Sophos are all good.

2.7. What payloads can a worm have?

Payloads that have been seen to date on worms include:

- Installing backdoors to later allow control of the computer.
- Defacing websites.
- Installing patches (so called good worms)
- Conducting distributed denial of service (DDOS) attacks against other sites.

On the whole, the worms to date have been remarkably benign, all things considered. Most of the harm they have done has just come from overloading networks with traffic, or rendering the infected computers inoperative for their intended purpose. Some things that we fear as possible payloads:

- Extensive deletion or corruption of data on hard drives
- Damage to the hardware (eg by reflashing the bios of the computer)
- Large scale retargetable DDOS attacks against many important targets simultaneously.
- Search for commercially or militarily significant information on the infected computers.
- Theft of personal information (eg credit card numbers) from infected systems.
- Sale of access to personal computers.
2.8. What papers should I read to find out more about worms?

Here are some suggestions (also look in the bibliography).

- Gene Spafford’s paper on the 1988 worm: The Internet Worm Program: An Analysis
- Our own How to Own the Internet in Your Spare Time (if you’ll indulge the promotion of our own work)
- CAIDA’s study of Code Red
- The Sapphire/Slammer analysis

3. How Worms Spread

3.1. What is a random scanning worm?

Every worm has to have a spread algorithm, and specifically a Target Acquisition Function. This is the part of the worm code that finds the next victim to try and infect. The most popular method is called random scanning: the worm simply picks a random IP address somewhere in the Internet Address space and then tries to connect to it and infect it. There are some variations here: in some cases a TCP worm attempts a full three way handshake with the chosen address using a TCP layer connect() call, or it could send syns to random addresses at high speed, and then only try to complete the handshake and send the exploit in those cases where it gets a syn-ack back. In the UDP case, the exploit and worm may be inside a single UDP packet which gets sent to the randomly chosen address (like the Slammer worm).

Random scanning worms have a characteristic spread pattern. They first spread exponentially, doubling and doubling gradually till there is a decent population of worms. This phases into a stage where the worms infect most of the network in a rapid linear rise. Finally, the worm takes quite a long time to finish finding the last vulnerable machines (saturating) - since just guessing random addresses is not a very efficient when most machines are already infected. The mathematics of random scanning worm spread is covered in more detail in How to Own the Internet in Your Spare Time.

Here’s a picture of the inbound scan rate due to Code Red at one site. The probe rate is proportional to the number of infected worm instances, so this gives a sense of the characteristic way in which random scanning worms spread.

Random scanning worms are very noisy and tend to waste a lot of network bandwidth scanning. This is because the great bulk of the random scans don’t do anything: not many addresses are vulnerable to begin with (the vulnerability density is low) so most scans are wasted even at the start of the worm. Plus the worms usually keep scanning long after everything is infected. When a random scanning worm is scanning on the Internet, everyone’s access link to the net gets deluged with scans. Often much of the harm the worm does comes just from this waste of bandwidth - preventing legitimate network applications from working and crashing routers (which often die if their cpu usage goes to 100% for any length of time). However, random scanning is a simple and robust approach to worm spread, so worm writers keep using it.

3.2. What is saturation of a worm?

Saturation refers to the worm infecting all the systems that were potentially vulnerable to the exploit(s) it has. Once saturation occurs, there is no more value in the worm continuing to try and spread, and things about worms and worm containment are often expressed in terms of saturation: time to saturation and X% saturated. Saturation could either be on the Internet, or with respect to some particular internal network.

In practice, saturation is somewhat fuzzy since vulnerable computers are constantly being turned on and off for reasons that have nothing to do with the worm. Additionally they may get patched prior to infection, cleaned up after infection, and then possibly reinfected. They also change IP address due to dialup lines, DHCP leases expiring, etc. Thus the concept of a static population of vulnerable machines which the worm simply compromises steadily until saturation is reached is a bit cleaner than the real world. However, it’s still a useful approximation for some purposes, especially for very fast worms where other processes don’t have much time to affect the dynamics of the worm spread.
3.3. What is subnet scanning?

Worms such as Code Red or Slammer that scan any old address on the Internet are inefficient in several ways. On the Internet, they are inefficient because the great bulk of scans cross the network core. This means that the scans are slower than they otherwise would be just because of latency, and congestion of the network. On enterprise networks, scanning random addresses is inefficient because most of the address space is not in use behind the firewall.

Thus, we presume, the worm writers came up with subnet scanning to solve these problems. In this approach, the worm differentially picks addresses closer to itself. For example, Code Red II picked a random address within its own class B 3/8 of the time. It picked a random address from its own class A 1/2 of the time, and only picked a completely random address 1/8 of the time. On the Internet, this means less of the worm spread is happening across the core, and more across local networks. In enterprise networks, it means that a worm is likely to compromise the enterprise far more quickly. For example, say the enterprise has two class B networks. A worm that falls into one of them and uses the Code Red algorithm will only fall into the populated space 1 in $2^{15}$ attempts. By contrast, the Code Red II algorithm will pick an address in the local class B (and therefore in the populated space) 3/8 of the time. This will dramatically improve the worm’s ability to saturate the enterprise network.

3.4. What is a Warhol worm?

The Warhol worm was a term made famous by our colleague Nick Weaver for a worm that could spread in less than 15 minutes (thus recalling Andy Warhol’s quote about how everyone could have 15 minutes of fame). The worm is a theoretical design that hasn’t been seen in the wild, but was described in Nick’s original writeup and our subsequent paper.

The Warhol worm relied on three strategies, two simple and one very clever. The clever one was co-ordinated permutation scanning, the subject of the next FAQ item. The first simple one was to use a hitlist: instead of starting at a single location, the releaser of the worm assembles a list of vulnerable machines in advance, and then starts the worm at all those hitlist sites almost simultaneously. This avoids a number of generations while the worm grows to the size of the hitlist, and thus shorten the total spread time considerably. The second simple technique was simply to scan faster. Many of the worms have had a really modest scan rate (the number of IPs per second the worm is able to check), and so this can be improved greatly by a better design.

Back of the envelope calculations suggest that with the implementation of all three techniques, the worm could saturate the Internet in significantly less than 15 minutes.

3.5. What is co-ordinated permutation scanning then?

The ideas is as follows. Suppose all the worms share a random permutation of the Internet Address space. (ie They can all generate the exact sequence of proceeding through all the addresses on the Internet in a random order, but in which each address is only visited once. Such a permutation can be implemented via certain kinds of random number generators (such as linear congruential generators) or via a good encryption cipher. It’s important that all the worms know the same permutation.

Now a worm begins scanning through the permutation. This will still look random to defenders. However, the worm has a second trick up it’s sleeve. When the worm scans an address which is already infected, the second instance responds to the scan in the same way that lets the first instance know the worm has already infected that address (eg by sending back a special magic number in one of the fields of the syn-ack response packet). A worm that realizes an address that it scanned is compromised can safely conclude that another worm instance is scanning through this region of the permutation, and there is not point in continuing. Hence it can switch to another randomly chosen part of the permutation to see if there is unchecked sequences of addresses there.

It turns out that this approach cleans up the last hosts before saturation significantly quicker than pure random scanning. It also gives the worm a reasonably efficient way to know when it is done. Each instance can only switch parts of the permutation three times, say, and then figure out it is done switch to some other more productive activity (whatever the payload is for example). Nick Weaver has shown that this approach does succeed in saturating, but significantly quicker.

3.6. What is a flash worm?

A flash worm is a worm that uses the following hypothetical algorithm (no flash worms have yet been seen in the wild). The worm releaser scans the network in advance and develops a complete hitlist of all vulnerable systems on the network.
The worm carries this address list with it, and spreads out through the list using a precomputed spread map. The first infected machine infects three more (say), and gives each of them 1/3 of the address list. They each infect three machines from their list, and give those 1/3 of the 1/3, and so on.

Thus the infection occurs in time that is basically the logarithm of the number of machines to be infected times the latency for each generation of infecting a few machines. This can be potentially very fast: tens of seconds for the Internet, and less than a second for an enterprise. Flash worms are also hard to contain. A more thorough analysis of them is in How to Own the Internet in your Spare time.

### 3.7. What is a topological Worm?

A topological worm is a worm that relies on information it finds on the infected host in order to locate further potential victims to infect. The original Internet Worm of 1988 was a topological worm. Modern worms have all been scanning worms - relying on guessing addresses rather than on using information from the host. To give an example, a topological web server worm might search the pages of the infected web server for URLs of other servers. It would then try and infect them.

Topological worms are probably somewhat intermediate between scanning worms and flash worms in speed, difficulty of containment, and robustness. Scanning is a very simple, robust strategy, but is rather inefficient and very noisy, giving a strong basis for detection and containment of the worm. Flash worms are completely efficient and quick but require elaborate reconnaissance and preparation. Topological worms make far fewer connections per infection than scanning worms, but must search the disk or memory of the infected machine for new links, which may be time consuming. Not all protocols are suited to topological worms; there must be rich enough information about other servers to support a worm spreading well.

### 3.8. What is a metaserver worm?

A metaserver worm is a special case of a topological worm in which the vulnerable protocol is such that a small number of metaservers contain information about the location of all the other vulnerable machines. Some Internet games are structured this way. A worm that can either legitimately query the metaservers for the locations of all the servers, or failing that compromise the metaservers, can then infect everything else in a very short order.

### 3.9. What is firewall penetration?

Some worms have had functionality particularly designed just to get them across the firewall so they can get a start inside. The prototypical case of this was Nimda. Generally, these are viral modes. The worm infects web servers on the Internet and hopes users inside the organizations will browse them and become infected also, or the worm sends infectious email, hoping users behind firewalls will read it. What a worm can potentially do however (as distinct from a virus), is spread on the Internet inside minutes and then begin its firewall penetration before any any-virus update occur.

### 4. Famous Worms

#### 4.1. Please Grandad, tell me about the original Internet Worm.

The Internet Worm of 1988 was the first worm that caused major problems. It was released in the early evening (Eastern US time) of November 2nd, 1988, and spread all across the Internet in the course of the next 24 hours. It led to widespread disruption of computers attached to the public network for several days following. At the time, the Internet only had 60,000 computers, mostly used by researchers and high-tech companies, so the potential damage was much less than in recent worm incidents.

The worm was released by Robert Morris Jr, an graduate student at Cornell University who also happened to be the son of the chief scientist of the National Computer Security Center (then the NSA’s center for research in computer security). Morris was fined and sentenced to community service for releasing the worm, but has since rehabilitate himself and is now a respected computer networking researcher.

The worm was a topological worm that read a variety of system configuration files to find information about other hosts to attack, as well as running utilities to look at current network connections for clues about other machines. Once having found a machine, it had four methods of attack:
- A buffer overflow in fingerd (a once common but now rarely used utility for determining who was active on a given computer).

- Use of the DEBUG command in sendmail (the Unix mail transfer program), which intentionally allowed arbitrary commands to be executed on a machine running sendmail with this option enabled. The option should have been disabled in production use, but often wasn’t.

- Cracking user passwords and then trying them on other machines.

- Exploiting trust relationships that allowed users of one machine to log into another without giving a password (Unix. rhosts and /hosts.equiv files).

The worm was capable of infecting several variants of BSD Unix, and consisted of two pieces - a small bootstrap program passed as a C source code and then compiled and executed, which then pulled over the main worm.

Estimates of the number of systems that got infected vary from about 1000 to 6000; there doesn’t seem to be a reliable basis to these estimates (they come from extrapolating from infection rates at small number of sites). There are no reliable data on the spread progression, but chronologies of events recorded in the literature suggest spread took around 24 hours. The worm had no intentionally damaging payload, but caused denial of service of the computers by overloading them with so many copies of itself that they couldn’t function. The worm was quite ingenious in several ways, but also contained numerous bugs and lots of sloppy programming practices.

The worm led directly to the creation of CERT/CC.

4.2. What happened with Code Red?

The Code Red incident was the biggest worm incident by far after the 1988 Internet Worm. It caused a huge stir because it spread so fast and so widely, and really put worms back on the map. It also sparked lots of research and product development.

There were at least three separate things called Code Red. The first version was initially seen in the wild on July 13th, 2001, according to Eeye Digital Security, who disassembled the worm code and analyzed its behavior. The worm spread by compromising Microsoft IIS web servers using the .ida vulnerability CVE-2001-0500. Once it infected a host, Code-Red spread by launching 99 threads which generated random IP address, and then tried to compromise those IP addresses using the same vulnerability. A hundredth thread defaced the web server in some cases.

However, the first version of the worm analyzed by Eeye, which came to be known as CRv1, hand an apparent bug. The random number generator was initialized with a fixed seed, so that all copies of the worm in a particular thread, on all hosts, generated and attempted to compromise exactly the same sequence of IP addresses. (The thread identifier is part of the seeding, so the worm had a hundred different sequences that it explores through the space of IP addresses, but it only explored those hundred.) Thus CRv1 had a linear spread and never compromised many machines.

On July 19th, 2001, a second version of the worm began to spread. Code Red I v2 was the same codebase as CRv1 in almost all respects – the only differences were fixing the bug with the random number generation, an end to website defacements, and a DDOS payload targeting the IP address of http://www.whitehouse.gov. this was the version that spread so rapidly and globally until almost all vulnerable IIS servers on the Internet were compromised. It stopped trying to spread at midnight UTC due to an internal constraint in the worm that caused it to turn itself off. It then reactivated on August 1st, though for a while its spread was suppressed by competition with Code Red II. However, Code Red II died by design on October 1, while Code Red I has continued to make a monthly resurgence to this day. Code Red followed the theory of a random scanning worm pretty closely.

The Code Red II worm was released on Saturday August 4th, 2001 and spread rapidly. The worm code contained a comment stating that it was "Code Red II" but it was an unrelated code base. It did use the same vulnerability, however. When successful, the payload installed a root backdoor allowing unrestricted remote access to the infected host. The worm exploit only worked correctly when IIS was running on Microsoft Windows 2000; On Windows NT it caused a system crash rather than an infection.

The worm was also a single-stage scanning worm that chose random IP addresses and attempted to infect them. However, it used subnet scanning, where it was differentially likely to attempt to infect addresses close to it. Specifically, with probability 3/8 it chose a random IP address from within the class B address space (/16) of the infected machine. With the probability 1/2 it chose randomly from its own class A (/8 network). Finally with probability 1/8 it would choose a random address from the whole Internet.
Code Red II suppressed the incidence of Code Red 1 v2 once it came out, but both continue to be present on the Internet today in small numbers.

More detail on Code Red can be found in the CERT Advisory, in How to Own the Internet in Your Spare Time, and CAIDA’s excellent analysis.

4.3. What happened with Nimda?

Nimda began on September 18th, 2001, just about exactly one week after the 9/11 incident, and spread very rapidly. It spread extensively behind firewalls, and illustrates the ferocity and wide reach that a multi-mode worm can exhibit. The worm is thought to have used at least five different methods to spread itself:

- By infecting Web Servers from infected client machines via active probing for a Microsoft II Vulnerability (CVE-2000-0884)
- By bulk emailing of itself as an attachment based on the email addresses determined from the infected machine.
- By copying itself across open network shares
- By adding exploit code to Web pages on compromised servers in order to infect clients which browse the page.
- By scanning for backdoors left behind by Code Red II and also the sadmind worm.

There is an additional synergy in Nimda’s use of multiple infection vectors: many firewall allow mail to pass untouched, relying on the mail servers to remove pathogens. Yet since many mail servers remove pathogens based on signatures, they aren’t effective during the first few minutes to hours of an outbreak, giving Nimda a reasonably effective means of crossing firewalls to invade internal networks.

Nimda was also interesting in another light: it contained code to delete all the data on hard drives of infected machines, but that section of the code was turned off.

There’s more information in the CERT Advisory on Nimda.

4.4. What happened with Slammer?

The Slammer worm occurred at almost exactly 9:30pm (Pacific Time) on Friday January 24th, 2003. The worm exploited a known vulnerability in Microsoft’s SQL server running on port 1434. The worm sent itself in the form of a single UDP packet with 376 bytes of data (404 bytes including headers). This packet included the exploit and the assembly language of the worm itself. When it hit a vulnerable IP/port combination, it would overflow a buffer and immediately begin execution without requiring any further interaction with the infecting machine. As such, it was the smallest worm to date.

It was also the fastest. The worm sat in a tight loop sending out copies of itself to random IP addresses (in classic random scanning worm fashion). We observed scan rates from 3000 packets per seconds to 30000 pps, massively faster than any other worm to date. Later, the worm became bandwidth limited: not all the worm’s packets could fit through networks, and spread slowed down. Still, it was mostly saturated after ten minutes.

The worm had no malicious payload, but caused significant disruption nonetheless, either by blocking networks or by infecting and making unavailable SQL servers that were performing critical tasks. The most notable damage was loss of service from Bank of America’s ATM network for most of a day.

The worm was also called Sapphire (our favorite name), and a more detailed analysis of its spread is here.

4.5. TCP worms couldn’t be nearly as fast as Slammer, right?

Actually, they could be even faster. A properly designed scanner would send out syn packets at near line rate, listen asynchronously for syn-ack responses, and only send the whole worm in the case that the exploit worked. At a typical Internet vulnerability densities of 0.001%-0.01%, the cost of sending out the syn packets considerably exceeds the cost of sending out the worms for a reasonably sized worm. A fast machine will be able to send out 40 byte syn packets considerably faster than 404 byte Slammer packets. (Note that a good implementation is going to write forged packets directly at the link-layer and bypass the stack altogether).

Now, the worm has to do some tricky things to manage congestion, especially when multiple instances are sharing the same link to a site, but it can be done. We’ keep the details to ourselves.
4.6. What happened with Blaster and Welchia?

The Blaster worm began on or about August 11th, 2003. It was a scanning worm that spread via Microsoft’s DCOM RPC mechanism, and thus was potentially able to infect most Windows XP and Windows 2000 systems (a huge population). The worm spread over the course of several days. No detailed analysis of its spread is available at this time, but anecdotal evidence suggests it spread very widely.

The spread algorithm was random-start, sequential search. That is, it picked a random place to begin, but then scanned upwards sequentially through IP addresses. 40% of the time, it picked a start within its own class B, and 60% of the time, it picked a completely random starting place. The mathematics of this is kind of spread aren’t known at this time; it’s likely similar to random scanning in the beginning and then finishes up faster at the end. (However, it’s easier to contain because inbound scan blocking will work against this scan algorithm, whereas it won’t against regular random scanning.

The worm’s main payload was a denial of service attack against windowsupdate.com. However, since the worm gave Microsoft several days before initiating the attack, they were able to avert it. The worm also installed a backdoor command shell that was remotely available. The worm had no dedicated firewall crossing functionality, but nonetheless managed to get into many organization and cause widespread problems. Finally, there is some evidence that the worm may have had a role in the NorthEast power outage of August 2003. See this ComputerWorld story for more detail.

The Welchia worm was an example of an attempted good worm that patched Windows systems vulnerable to Blaster and also removed Blaster. It began about a week after Blaster. In fact, it considerably worsened the harm by taking down networks with excessive traffic - indeed anecdotal evidence suggests that Welchia did more harm than the Blaster worm it was presumably meant to cure. Welchia was also a random start sequential scanner but checked with ICMP whether the IP was live before attempting to infect the address.

5. The Future of Worms

5.1. How fast could a worm compromise the Internet?

The worst case is a flash worm with a precomputed spread map optimized with knowledge of the Internet topology. It could almost certainly saturate the vulnerable population connected at the time in less than thirty second. See How to Own the Internet in Your Spare Time for more information.

The fastest worm to date was Slammer (a random scanner), which saturated in not much more than ten minutes. The TCP random scanning worms have all taken a number of hours (or even days) to saturate, but that’s because their scanners were inefficient: they could be designed to go as fast as Slammer.

5.2. How fast could a worm compromise my enterprise?

The worst cases are a flash worm (assuming someone had gone to the trouble of mapping your enterprise from the inside in advance), or a topological worm where the topological information was in memory (as opposed to on disk). Such a worm could saturate the vulnerable population inside an enterprise in a few hundred milliseconds.

The more common case of a random scanner can vary from seconds to hours, depending on the structure of the address space and the vulnerability density. There’s a discussion of this point in Containment of Scanning Worms in Enterprise Networks. Note that if the worm has a decent guess at the address space (say it first scans the local Class B and that happens to be your whole address space), it need only to take a few seconds to do that at Slammer scanning speeds.

In general, you should assume that a worm can fully compromise your network before you as a human can figure out what is happening, and before any vendor can produce a signature update.

5.3. How long can worms last?

There are still quite a few infected instances of Code Red and Nimda scanning on the Internet now, several years after those worms were released. See this graph (thanks to Vern Paxson at ICSI/LBNL). So certainly worms can potentially become endemic and last for years - as long as the vulnerability lasts.
5.4. Are the worms to date any good?

Some of them show significant hacking skills and cleverness with assembly programming. The 1988 Internet Worm was very innovative, and is still the only worm to use a zero-day exploit (however it also contained many sloppy programming errors, reportedly). Nimda was quite slick and innovative and betrayed a sophisticated author. Code Red II came up with the clever idea of local subnet scanning. However, most of the worms to date have shown a poor grasp of how to spread broadly and quickly, and most have had some significant errors in them. The scanner designs could be much faster (except for Slammer), and they have all achieved far lower penetrations than the number of unpatched systems might lead one to expect - suggesting that the exploits are usually fragile and don’t work on all the putatively vulnerable systems. Plus the payloads have often failed, eg the DDOS of Code Red against the White House was easily circumventable. Blaster was a particularly inept worm - it could have been a devastating attack against Microsoft update, but it spread so slowly that Microsoft had plenty of time to counter it. Overall, the worm writers could do much better if they studied better, worked harder, and tested properly. It’s lazy that they keep using old exploits instead of figuring out new ones.

5.5. Why do people write worms?

Most to date appear to have been written for motivations that lie along the spectrum from “creating graffiti on the Internet” to “carrying out a huge global prank” to “this was my student project”. It’s the lack of serious intent to harm on the part of the worm writers that has saved us from much further damage. The worst of it has been the trojans and backdoors, which can lead to later compromise of personal information and identity theft. Nimda appeared more sophisticated: there were a series of variants, and it had the flavor of someone exploring the technology for later use.

5.6. How hard is it to write a worm? Do you need a BS/CS or a PhD?

Any half competent programmer can write a worm if they put their mind to it. They can get exploits for old vulnerabilities on the net. Event pretty lousy worms will spread at the moment. Having advanced hacking skills: ie the ability to find a novel vulnerability and write a set of highly portable exploits for it, will allow the worm writer to create something that will spread far faster and more widely. Having the kind of discipline that software engineering courses teach will help to ensure the worm is well designed, well written, and properly tested in advance. To engineer a well tested worm capable of scanning fast, breaching firewalls reliably, and causing a global disaster is probably beyond the skills of most amateurs. Having advanced degrees will help to understand the mathematics of worm spread and worm containment, which in turn would allows the creation of a truly superior worm that would spread like lightning everywhere, be too obfuscated to easily reverse engineer, and defeat event the most advanced defenses.

Someone creating the Uberworm would likely put together a team with a scientist who has studies the worm literature, a handful of good software engineers (familiar with operating system internals, networking, and intrusion detection) and a vulnerability researcher capable of developing exploits for novel vulnerabilities. Plus a well equipped lab with a lot of different systems to test against. Given those ingredients, it’s probably only a few months work, with most of the time going on the testing.

5.7. I want to write a worm. Can you help me?

There have certainly been days when we’ve been tempted, either to write worms or help those that do. Pioneering a new field has been frustrating at time. However, the Silicon Defense Core Values run strong in our blood and restrain us. So, no, we can’t help you, and we try to avoid providing details that will mainly be useful to worm writers. However, any aspiring worm writer should certainly study the literature on worms, and this sampling is a good place to start.

5.8. Is it legal for me to launch a good worm?

A worm by definition breaks into computers. That is now a crime in almost every jurisdiction. Therefore launching a worm, with any purpose whatsoever, is likely to be committing a widespread crime in many places at once. You knew it was going to break into systems, so you had a criminal intent. Sometimes people are tempted to write “good” worms that will patch systems, rather than causing harm. Don’t do it:
- It’s illegal. If you are caught, you could go to jail for a long time.
- The network traffic from your worm could disrupt critical infrastructures, even if the worm itself has no malicious payload. Eg. see Welchia which was a supposedly good worm that took out the US-Navy-Marine Intranet.
- Patches sometimes destabilize the computer or cause other side-effects. It’s up to the owner of the computer to decide whether they want to take that risk.

5.9. How much do worms cost society?

Needless to say, this isn’t easy to measure. However, the market research firm Computer Economics produces widely cited estimates of the total cost of major worm and virus incidents. Here are their figures for the recent worms:

<table>
<thead>
<tr>
<th>Worm</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Red</td>
<td>$2.62 billion</td>
</tr>
<tr>
<td>Nimda</td>
<td>$0.64 billion</td>
</tr>
<tr>
<td>Slammer</td>
<td>$1.25 billion</td>
</tr>
<tr>
<td>Blaster[1]</td>
<td>$2.0 billion</td>
</tr>
</tbody>
</table>

[1]: Blaster cost includes the cost of the near simultaneous Sobig.F virus.

5.10. What is the Uberworm? How bad could a worm be?

The Uberworm is the official Silicon Defense term for the really big bad worm that is going to cause major widespread harm someday, and that we hope to mitigate by developing worm containment technology and educating people about worms before it happens.

Thinking changes on what the Uberworm might do. A while back, DARPA asked us to study what the worst reasonably likely worm incident could do. This resulted in the Worst Case Worm report. In that, we investigated what a terrorist group or a nation state could do if they wanted to attack us with a worm. Our answer was roughly:

- use a three stage worm with fast scanning spread on the Internet, firewall penetration, and then topological or scanning worm spreading on enterprises.
- robust portable exploit affecting a broad range of Windows systems
- wipe out the data on a sizeable fraction of all the hard drives on the country
- damage the hardware on a smaller fraction of the computers.

That would be bad.

However, we thought that was the worst case when we used to think that the power grid was probably invulnerable to worms.

5.11. Is the power grid vulnerable to worms?

It rather appears that it might be. Many Internet security experts used to assume that the SCADA systems that control power generation and transmission equipment were a foreign world that didn’t interact with our world, and couldn’t be affected by worms and other problems of the Internet. Following the Blaster incident however, which overlapped with the 2003 power grid outage on the east coast, it emerged that some SCADA systems were in fact running on top of Windows DCOM (the service vulnerable to the worm), and thus were potentially vulnerable to the worm if it once got into the intranets in question (and it’s hard to keep a worm out with complete certainty). There’s also some indication that worm traffic interfered with communication between the players during the outage.

There’s no proof at this point that Blaster played a key role in the 2003 outage, but there does seem to be enough information to conclude that a worm could interfere with operation of the power grid - even a general internet worm not designed specifically for interfering with power. A worm (or series of worms) designed by someone with inside knowledge of power grid information systems could presumably be fairly devastating.

A new story worth reading is this one from Computerworld.
5.12. Is Al Qaeda writing worms to destroy civilization?

There’s been no evidence of this in the open literature. A Washington Post article in 2002 did suggest that Al Qaeda was researching cyber-attacks, but direct attacks on critical infrastructure rather than worms. However, who really knows?

5.13. Is Country X writing worms to destroy civilization?

A number of countries are known to have active programs developing cyberwar attacks. The United States is almost certainly investing the most in this, and is probably the most dangerous adversary to anyone else’s cyber-infrastructure. However, China, Russia, and the major European countries all have given considerable thought to the area. Public details of the strength and philosophy of their capabilities are naturally rather limited. There are some indications that China has picked cyber attacks as one of the major ways in which they might offset US military superiority in conventional forces (they have doctrine fielding the wonderfully named "People’s Information Army").

In general, the state of network defense is so abysmal relative to the capabilities of attackers that any moderately developed country that puts a serious effort into it should be able to develop devastating offensive cyberwar capabilities.

If a single loser can perpetrate a major global worm incident, what could a professional operation do?

5.14. Why would a country release a worm - wouldn’t it hurt them just as much?

Not necessarily. There are some fairly simple techniques for limiting the damage to particular target countries that would work most of the time. The basic observation is that computers generally have a setting for the language of the user, and for the timezone the computer is situated in. If a worm contains code to only execute the payload on computers between 5 and 8 hours behind UTC and with US English as the language, then the worm’s harm will be overwhelmingly confined to the US and Canada. Whereas, if someone wanted to attack France, they’d choose the language to be French and the timezone as UTC plus one hour. There’d be some collateral damage in Africa, but by and large, this would hit the French and not the English, or the Chinese.

It’s also possible to gain geographical information from IP addresses, but this is less reliable, especially with intranets were addresses may often not be the publically routable one managed by the various regional addressing authorities. However, it has the benefit that knowing the geography of IP’s allows the worm to avoid even infecting computers in the wrong countries, rather than infecting them and then just not executing the payload.

5.15. Aaagh - what can I do to protect my critically important network?

Well, that’s the subject of the next section, on worm containment!!

5.16. I’m a journalist or a policymaker. What are some ideas for solving the worm problem?

Anything that causes vendors to shio fewer vulnerabilities, causes users to patch their systems faster, or leads to better technical worm containment defenses would be good. Here are some ideas along those lines. Most of these would make the Internet significantly safer, but would be MUCH LESS FUN than the current modus operandi (and likely less profitable for the various parties also). They will not be politically feasible until the damage from worms has worsened significantly further.

• Make software vendors subject to product liability laws so that shipping flaws becomes much more costly to them.
• Set up a government agency to fine vendors who ship vulnerabilities. Recycle some of the money to independent bounty hunters that find new vulnerabilities and report them to the government agency.
• Require software engineers to be licensed like civil engineers. That way, tyros just out of college won’t be writing critical or widespread applications without a clue.
• Have a government agency scan the national address space. Give fix-it ticketxs to people with vulnerable computers. Fine them if they haven’t patched their system two weeks later.
● More research funding. Oddly, under the Bush administration, there has been a massive contraction in research funding into Internet Security. A lot of the research community that existed three years ago has dried up and blown away. More funding would be good (hopefully HSARPA will step up to the plate here eventually). In particular, worm containment research is still a very new field, and there is a lot more to be done. Funding should be allocated based on merit as determined by peer-review.

● Mandate that ISP’s do not allow scanning out of their network. Also mandate egress filtering.

● Make sites liable for damage caused by compromised machines on their network, so they have an incentive not to get hacked.

● Mandatory disclosure laws for security incidents.

● Mandate worm containment technologies (not that we’d have any financial interest in this last idea!).

6. Worm Containment

6.1. What is worm containment?

Worm containment is the art, science, and engineering discipline of preventing worms from spreading. The worm containment perspective assumes that there will always be vulnerabilities in widespread software, and always be some parties with malicious intent who will release worms, and asks how to ensure that the release of such a worm will not result in a widespread epidemic. The defining characteristic of worm containment, as distinct from anti-virus technology, is that it must be all automated with no human in the loop. Otherwise, it may very well be too slow to be useful.

We can talk about worm containment on the internet, where we assume someone malicious released the worm at one or more places and now we must stop it. We can also study it on the enterprise network, where we assume that somehow the worm got a start on the internal network, and now we must prevent it from infecting everything else in the organization. Most of the rest of the FAQ is concerned with the enterprise case (which is a lot more promising).

Worm containment is also sometimes called worm quarantine.

6.2. Shouldn’t the vendors just fix their software and the problem would go away?

It’s not so simple. Software is written by humans. Humans make mistakes. So software will always initially contain flaws, some of which will be security significant. Experienced programmers working in an engineering culture with a high commitment to quality and a good knowledge of security issues will create fewer security problems, but they still won’t produce perfect software with zero security problems.

So then it comes down to testing. Software engineering researchers have found that the number of defects in a given piece of software is, at best, inversely proportional to the amount of time spent testing it (to put some complex results in a simple form). So if you test your software well for ten times longer, it will have 10% as many defects in. If you test it 100 times longer, it will have 1% as many defects in. What is not possible is to eliminate all the defects in any reasonable amount of time.

Given that software vendors operate in a free market where players who are late to market generally get crushed, it’s easy to see why software usually ships with lots of defects in. Even open source systems compete in the sense that if such a system doesn’t evolve new features fast enough, users will switch to something else and the programmers will lose the recognition and sense of meaning that motivated them to write the system. So there too, software faces time pressures that make for limited testing. But even massive amounts of effort on software quality would not eliminate all vulnerabilities.

Since every vulnerability creates the possibility of a worm spreading by exploiting the vulnerability, we can expect worms to be with us for a long time.

Having said that, not all vendors are equal. Some have better engineering cultures than others and produce fewer defects. Also worth noting is the relative size of different applications. Some vendors (notably Microsoft) favor producing extremely large and complex applications and operating systems. No matter how careful such vendors are, very large complex systems inevitably have many defects, so there is value in pressing vendors to produce simpler, better-tested systems. Fewer vulnerabilities would be better than more vulnerabilities, even if we can never get to zero vulnerabilities. Similarly, vendors should provide fixes promptly and make it easy for users to install them, so that when a vulnerability is discovered, the window of time available for large scale exploitation of it is minimized.
6.3. Doesn’t anti-virus software do worm containment?

Anti-virus software works by checking content (executable, attachments) for specific signs that reflect particular viruses—the set of signs for a particular virus is generally known as a signature. When a new virus released, the anti-virus companies obtain copies of it, analyze it, generate a new signature, and disseminate it to their customers. While they have got very good at this, it remains a problem which involves a certain amount of human analysis and decision-making in response to the virus incident as it develops. The takes hours or even days. This was perfectly adequate against most viruses.

The problem worms pose for this approach is their speed. Worms can spread globally in substantially less than an hour, and perhaps even in less than a minute. This is faster than any reasonable human mediated process can produce a new signature and disseminate it. Thus anti-virus systems cannot solve the worm problem (though they remain a very valuable and important part of an organization’s network security defenses).

6.4. Do firewalls help with worms? Are they enough?

Firewalls are critical first line of defense. Without a properly configured firewall on all access links to the Internet, and all links to business partners, worms can freely scan into the enterprise, which makes it very hard to control them. Every address on the network can be hit multiple times from outside the enterprise. It’s essential that firewalls be in position and be correctly configured so that scans can only find a handful of carefully hardened and administered machines in DMZ’s.

However, it’s not likely that firewalls alone can prevent worms getting into enterprise networks. There are too many ways around.

6.5. What about internal firewalls? Router/Switch ACLs?

Excellent ideas. The more you can firewall off pieces of the enterprise network, and the more you can filter traffic, the harder it is for worms to spread across it. In fact, if you get things to the point where every host is only able to see less than one other vulnerable host, then you have an adequate alternative to dedicated worm containment. Hard to create and maintain that, however.

6.6. What about intrusion detection systems? Intrusion prevention?

Intrusion detection systems just detect incidents. They will often detect worms, but by itself that is of limited value since the worm is likely to spread fast enough that merely notifying humans will not cause a useful response until after the worm has completed its spread.

Intrusion prevention systems are basically intrusion detection systems that automatically block the things they detect. If an intrusion prevention system blocks scans, then it can be used as a worm containment device, if suitable deployed (according to the guidance discussed elsewhere in this document). However, general purpose intrusion prevention involves many time-consuming calculations. That requires either running intrusion prevention software on a general processor (in which case the system will be quite slow), or running dedicated hardware developed just for intrusion prevention (in which case the system will be quite expensive). Some intrusion prevention systems have been known to just stop operating and start emitting smoke during the major worm incidents, which isn’t exactly the desired behavior.

Thus there is value in using dedicated worm containment systems, which can be much faster/cheaper, and therefore allow of a broader and finer-grained deployment. Also, worm containment systems are likely to have interfaces and other supporting tools more directly helpful to blocking worms, and less complexity associated with handling other classes of intrusions (which are generally rarer on internal networks).

It may well be useful to combine a worm containment system with a more general intrusion prevention or detection solution in front of key assets of the organization that might attract human attackers.

6.7. What are some ways worms can get inside my enterprise?

- Mobile machines may get infected while connected at home or connected to other networks, and then bring the infestation into the corporate network.
- People may dialup to outside ISP’s while also connected to the internal network (eg to check and alternate email address or circumvent some firewall policy they find inconvenient) and get infected via the ISP connection.
Wireless networks frequently overlap multiple organizations, and may allow people outside the organization to connect to the internal network, and possible infect it (either deliberately or inadvertently).

Alternatively, an internal machine may be misconfigured and connect to an external wireless network from which it can be scanned and infected.

Home machines may be connected to the Internet and the corporate network and cause infections that way.

Poorly configured firewalls and DMZ’s can allow scanning worms from the outside to get a foothold inside the intranet.

Unfirewalled connections (or inadequately firewalled connections) to business partners can allow scanning worms inside the business partner to cross into the enterprise network.

Worms can have viral firewall crossing methods (much as Nimda did). For example.

They can send themselves in email that might be opened by workers inside the organization.

They can infect Internet web sites or other servers with content that will infect browsers inside the organization.

Infected DNS, NTP, or other servers or infrastructure components outside the organization could infect their internal peers when queried.

6.8. Vendor ABC told me their system would prevent all worms entering my network. What should I think?

Vendor ABC lies. It’s beyond the state of the art to reliably detect and block novel worms on all possible ways the worm can get in. This is not to say that perimeter defenses are not useful - good defenses can certainly lower the probability of the worm entering. They’ll keep out the dumbest worms. However sophisticated worms will get in some of the time. This is why it’s worth considering network worm containment on the internal network as part of a defense in depth - if the worm does get in, worm containment may prevent it spreading. Worm containment techniques are not perfect either, but atleast they have quantifiable performance for the most common classes of worms.

6.9. What is this vulnerability density you keep talking about?

The vulnerability density is the proportion of IP addresses on some network that are vulnerable to a particular worm. Note, it’s usually defined as the proportion of addresses, not of computers, or computers with some particular kind of OS or application. The vulnerability density is thus the probability that a random scanning worm scanning exactly that network would succeed in hitting a vulnerable system on the first probe. We can talk about the vulnerability density of the Internet, or the vulnerability density on particular enterprise networks behind their firewalls. In the former case it’s the total number of vulnerable systems divided by 2^32, while in the latter case we divide by the size of the address space the enterprise uses internally.

Observed vulnerability densities have been surprisingly low. For example, Code Red on the internet had a vulnerability density of $8 \times 10^{-8}$ less than 1 in 10,000 addresses were vulnerable. Most other worms have had similar or lower vulnerability densities. The worm may or may not actually succeed in saturating the vulnerable population depending on the worm spread algorithm and any containment measures that are taken.

6.10. What is the epidemic threshold for a containment system?

The epidemic threshold is one of the most critical concepts in worm containment. The worm is trying to spread exponentially. Left alone, each worm instance will find a number of other worm instances to infect, each of which will find further worm instances (at least in the early stages of spread when there are plenty of uninfected vulnerable systems to find). A worm containment system attempts to identify worm instances via some mechanism and then prevent them from spreading. The worm writer hopes that his worm will be able to identify and infect enough other systems before it is contained that the worm will spread.

The epidemic threshold is the condition at which a worm instance, on average, can find 1.0 other vulnerable machines to infect before being contained. Below the threshold, the worm instance can find fewer than 1.0 vulnerable machines, and the worm will not be able to spread. Say we start with four worm instances, but they can only find 0.5 vulnerable machines
before containment kicks in and stops them in their tracks. So the four worm instances will create two children, which will create one grandchild and that will likely be the end of the infection. Contrast that to the situation in which the worm can find 2.0 vulnerable machine. In that case, four worm instances becomes eight, and then sixteen, and then thirty two, and on it goes for a long time with a huge number of machines compromised. In general, if the average number of children is less than one, the total number of infectees will be modest adn there will be no exponential growth. If it’s more than one, the worm will grow exponentially and large numbers of machines will be infected. This is the importance of the epidemic threshold.

6.11. Can scanning worms be contained?

Yes - this is technically quite feasible. All one needs to do is put in place software/devices that cut off scanning on the network. To ensure that a scanning worm is below the epidemic threshold we have to put in a system which can ensure that scans will generally find fewer than one vulnerable machine on the network. Then a scanning worm cannot spread. Anything that detects and blocks portscans can potentially be used for this purpose if deployed widely enough. Since random scanning worms are quite noisy and inefficient. It’s generally possible to detect and block a scan before it finds a vulnerable machine. Many intrusion prevention systems should be adaptable for this purpose, though there are a few issues.

6.12. So what’s the bad news?

The bad news is this. Scan blocking as a means of worm containment works much better outbound from near the infected machine than it does at preventing a machine across the network from infecting something behind but close to the worm containment system via inbound scans. Consider:

- When a network worm containment system is watching the outbound behavior of an address it is close to, it can see most or all of its behavior. Therefore, it can draw the conclusion that it is scanning early in the scan. If the system is monitoring an address on the other side of the network, it only sees a small fraction of the scanning, and therefore cannot decide it is a scan until a lot of scanning has happened.

- When a containment system is blocking an address, if it is doing outbound blocking of an address it is in front of, it can block most or all of the scanning. If it is blocking a remote address, it only blocks the small amount of scanning that happens to attempt to cross this particular device.

- If containment is inbound, the worm gets at least a few tries at every part of the network, and as many tries as it wants at any parts of the network that don’t have containment defenses in front of them. It has an excellent chance of hitting a vulnerable machine somewhere. Then that one gets to repeat the process. You get the idea. Overall, it’s very hard to get such a system below the epidemic threshold. (Without lots of correlation from all over the network, which has the problem of being too slow acting, and again allowing the worm to escape and propagate before the containment system acts).

- From the perspective of an individual defending system, the worm appears to prohgate all over the network and then hit it from many points at once. The system blocks bad IP’s, but then more and more show up and eventually one is going to get through before it can be blocked.

The other bad news is that worm containment needs to be fairly completely deployed. If much of then network is left with no worm containment defense in front of it, then if that part gets infected, it can site and try to infect the rest of the network as much as it likes, and it’s hard to prevent it from succeeding.

Thus scan-blocking based worm containment needs to separate up the network into cells. and prevent the worms from breaking out of those cells.

6.13. Is it better to do worm containment on end systems or in network devices?

As usual in life, it’s a trade-off. If it’s done on the end systems, there’s such finer grained visibility, no problem with address spoofing, and the possibility of fine-grained response. But deployment is a nightmare (and you need complete deployment), and the mechanism is potentially vulnerable to being disabled by a worm that knew about it. It some sense, this is like network work containment with a cell size of 1 address.

Doing it in the networks makes deployment cheaper but is coarser grained and cruder. The worm will have a harder time just straight disabling the mechanism, but can try to fool it by address munging tricks, scanning within the cell first, etc.
6.14. What’s a cell?

Network layer worm containment operates by preventing worms from spreading from one infected host to others. To do this, it’s necessary that the infected host not be able to get out to the rest of the network by any path that doesn’t have a worm containment device inline. Thus the worm containment devices have to break the network into pieces that are walled off from one another. We refer to these as cells. Then the worm containment prevents escape of the worm from the initial cell to other cells. It’s similar to ships which are designed with a series of watertight internal compartments separated by bulkheads. If one compartment is breached when the ship strikes a rock, the bulkheads prevent all the others from filling with water and sinking the ship.

Designing a deployment involves choosing the size of the cells. There’s a tradeoff here: relatively small cells will give much better protection as the worm will be confined to a very small initial part of the network, and will be much less likely to breach the containment devices. However, this involves deploying, configuring, and managing worm containment at many points in the network which is expensive.

On the other hand, large cells will be much cheaper to deploy and maintain. However, in the event of a worm, the worm can spread throughout the large cell, infecting all the vulnerable systems within it. Additionally, those systems can all try to infect out through the containment devices at the cell boundary. That will result in a higher likelihood of a breach. In general, worm containment will not work (ie keep the worm below the epidemic threshold) at nearly such a large vulnerability densities if the cells are large as it would if they were small.

Cells should not necessarily all be the same size. Where a range of address space contains few systems, or few systems with any services visible, or systems that are otherwise believed to be invulnerable to worms, cells can be large. In areas of the network with densely packed systems with potential vulnerable services turned on, cells should be smaller.

6.15. What about host intrusion prevention systems - do those contain worms?

Yes and no. Host intrusion prevention systems (some of which are now being marketed as anti-worm solutions) are software systems that run on end hosts and attempt to either prevent an attack from succeeding in exploiting a vulnerability, or it can, prevent the compromised process from doing anything it wouldn’t normally do. There are some good techniques for doing this, even without prior knowledge of the specific vulnerability, and these systems are valuable, especially for key servers. There is some hassle in the care and feeding of these systems. Network Computing has a nice review of the space in October 2002.

From the perspective of a system such as this, as worm is just like any other attacker. To the extent the system works, it can prevent or limit the harm the worm does to the systems it runs on. However, it’s likely to be prohibitive for most enterprises of any size to produce a complete deployment of such devices. With an incomplete deployment, the worm can potentially spread on the unprotected but vulnerable systems, and then get frustrated and DOS the hell out of the systems that were protected. Unlike with network worm containment, there’s no fallback to the crude but useful approach of large cells in the event of partial deployment.

Thus host intrusion prevention systems are not a realistic enterprise worm containment strategy by themselves, though they certainly have their place as part of defense in depth.

As usual, there’s a tradeoff between doing worm containment on end systems and doing it in the network. Doing it on the host is theoretically the best way, but TCO is overwhelming and the cruder but cheaper approach of doing it on the network has a place also. Additionally, network systems tend to be less open to subversion by the attacker/worm.

6.16. Are there products that can help with this?

Well of course. In most cases, it’s fairly unclear at this point how the products work, and which one will really work correctly. You’re going to have to test them yourself.

- Silicon Defense has the patent pending CounterMalice worm containment system.
- IBM has announced technology that they will hopefully shortly bring to market. See this story.
- Several intrusion prevention companies, including Tipping Point and Captus Networks mention that their appliances stop worms. Ditto several of the DDOS/traffic management companies such as Arbor and Mazu. Details are very sketchy....
Most recently, ForeScout has announced technology to block worms on the internal network.

6.17. I should concentrate my worm containment systems in front of key servers, right?

No. Remember the bad news about how worm containment works best outbound. This means that instead of concentrating the worm containment devices in front of key servers, what you should actually do is concentrate the devices in front of the worst administered, weakest security, most vulnerable parts of the network.

The one thing you might want to do differently with worm containment in front of key servers is tune it a little loser (set thresholds higher). False positives here will be bad.

6.18. So what are some places I should concentrate my worm containment systems?

Anywhere the vulnerability density is likely to be high. For example, consider focussing in front of:

- Places where addresses are densely used, and many services are on.
- Dial-up modem pools, or other places where remote access devices come onto the network
- Wireless networks
- Connections from small offices with no admin staff
- Connections from business units that don’t take security seriously or are understaffed.
- Connections from business partners.

These are places to make cells smaller. In contrast, if you have parts of the address space with few addresses live, or where every machine has a tight personal firewall showing no services to the rest of the world, you can afford very large cells.

6.19. Can’t I just have a single IDS on my network that that tells switches to turn off ports, and contain worms that way?

While we won’t say this is completely useless - it might work sometime, and it’s at least a way to prevent worm instances from DOSing the network for extended periods, it certainly is not a reliable engineered way to contain worms. There are three problems.

The first problem is that the worm might succeed in scanning a vulnerable host on the network before it scans the IDS enough to trigger a detection (thereby being above the epidemic threshold). So for this to work, the IDS(s) need to be monitoring a cross section of the network that is significantly larger than all the potentially vulnerable hosts put together. This is possible with enough deployment, or perhaps with some routing tricks - if there are enough chunks of unused space on your network, route them all to an IDS and trigger on that (this is called a network telescope).

But the second problem is latency. Consider that quite a few Slammer infected hosts achieved scan rates of 30000 scans per second and it was a single packet worm. If the vulnerability density is 1 in 1000, that means the worm can find a vulnerable host after scans that make it through the switch in about 30 ms. So to prevent the spread, the IDS has to detect the worm, produce an alert, poll the switch to figure out the right port, and then tell the switch to block the port. All this tends to take several seconds, not 30 ms. So it’s too slow to work reliably.

The third problem is that the worm has a some good workarounds without doing all that much work. One is to scan close to itself first. That way, it has a decent chance of spreading before impinging on the IDS. Another is to pretend to have a lot more IP/Mac combinations than the box normally should. That way, it can spread out the load of scanning over a number of inference units from the standpoint of the IDS, delaying the work of detecting and blocking the right port.

Ideally, worm containment would be done in the switch. In the meantime, doing it well implies deploying worm containment infrastructure with cells as small as practicle.
6.20. How do I design enterprise deployment of worm containment systems?

- If you don’t already know, figure out what address space your organization has, and what the network topology is. If it’s impossible to figure out the latter, at least figure it out enough to know where choke points are.

- Scan your address space on all important ports. A vulnerability scanner will tell you currently known vulnerabilities, but you also need to know all open services in case services that aren’t now known to be vulnerable turn out later to have weaknesses. Figure out the highest open service density of any service (ie this is the worst case vulnerability density - though it seems to be rare in practice for all potentially vulnerable machines to actually be vulnerable).

- Get as many services turned off or firewalled as possible to lower the potential vulnerability density. The lower it is, the better a containment defense will work.

- Figure out the budget and staffing for the project. It may be easier to sell this to management if you it in stages - start out with a modest deployment and then after you’ve proven you didn’t bring the network to its knees, go back for more funds to do a better job. Figure out how many worm containment devices/licenses you can realistically deploy.

- Decide where to drop the devices inline into the network. The goals are
  - Ensure that the network is completely divided into separate cells by the devices.
  - Every cell is closed off by as few devices as possible (so each of them sees as much of the outbound traffic as possible).
  - All cells have roughly equal number of vulnerabilities or potential vulnerabilities in them.

- Deploy and configure the devices. Figure out what the scan threshold is (how many probes get through before the device blocks a scan). You ideally want the following equation to hold. If T is the scan threshold, v is the average vulnerability density on the network, and c is the average number of vulnerabilities in a cell, you would like Tc < 1/v. That will mean the system is below its epidemic threshold.

One natural way to do a coarse grained deployment is to use the WAN as the basis for dividing up the network - every remote office has a device to prevent scanning out of it. Of course, this doesn’t help if the headquarters is 80% of the network. Then it’s a matter of figuring out choke points in the headquarters network. A fine grained deployment would involve putting a worm containment device in front of every switch that connects to wall ports. That will give a very small cell size - only the number of wall ports (plus any less official switches and hubs that hang off the wall ports).

There’s a lot more about the mathematics of scanning spread as a function of cell size, scanning algorithm, etc in Containment of Scanning Worms in Enterprise Networks.

6.21. How do I know my containment system will really work? I don’t want to loose a real worm on my network to test it?

There is a way to test a system’s ability to contain at least scanning worms without actually releasing one. The basic idea is this. We want the system to assure us that the worm will fall below the epidemic threshold, which in turn means that an instance of a scanning worm would not be able to find more than one other vulnerable system to compromise before being blocked.

Follow this sequence of steps:

- Choose a service to test

- Use your favorite scanning tool (eg Nmap is a popular free scanner). Pick a scan speed and pattern (it would be better to use a random scanning pattern than a sequential scan since sequential scans are much easier to detect and stop).

- For each of a sequence of randomly chosen IP’s in the network, generate a scan with that algorithm until it is stopped by the containment system. Count how many machines with a given operating system (or application, if more appropriate) and the chosen service open were seen. Reset the containment system block and repeat.

- Compute the average number of machines with a given OS and service open visible through the scanning system from each location.
Repeat as desired for other services, other scan speeds and algorithms.

For each such trial, if on average, more than one machine with the service open and the same operating system can be seen, the containment system is inadequate in that part of the network, and there is a risk of an epidemic. The system is above the epidemic threshold. If the containment system can routinely ensure that a scan can see an average of fewer than one machine with a particular service/opening system combination, then the containment system is adequate, and a scanning worm with that algorithm will not be able to propagate through it because it is below the containment threshold.

Doing this thoroughly is a bit tricky however, because there’s a lot different scan algorithms the worm could employ, and depending on how the worm containment system and deployment were designed, some might make it through while others are stopped. You really want to test from multiple places in the network also, especially wherever you suspect the weak spots in your containment defenses are.

6.22. Worm containment is too complicated. Is there an alternative?

This will work for scanning worms (not topological worms), but it’s a lot of trouble:

- Keep all system patched fully up to date.
- Turn off or rewall all services that aren’t strictly needed
- Divide your enterprise network thoroughly up with firewalls and/or switch and router ACLs.
- Ensure by this means that every system on the network can only reach the small number of other systems it really need to reach. A system should be able to see fewer other IPs than the inverse of the highest possible vulnerability density on the network.
- Maintain this situation as your organization and network changes.

Or you can just tolerate the occasional worm that makes it through. Do remember about the possible payloads though.

6.23. What’s the prospect for worm containment on the Internet itself?

Well, at a technical level, it’s quite feasible to contain scanning worms on the Internet. However, from a political/business perspective it’s rather challenging. The basic problem arises from the bad news that worm containment works best out outbound and requires broad deployment and a lot of co-operation to work. Security experts have been trying to get people to take the most simple basic security precautions in order to protect the network for years, with very limited success. It’s been striking that we have major virus incidents happening, even though anti-virus companies have been selling excellent defenses against viruses for over a decade.

So I’m not very optimistic that worms will get contained on the internet real soon. I think it will take international treaties and regulation to bring it about, and probably the pain due to worms will have to get significantly higher before that can occur.

In the mean time we can all focus on preventing worm spread on enterprise networks, which is much simpler from a political standpoint (an enterprise can take a rational view of protecting the whole internal network), and is thus likely to present better business models for vendors. Solving this problem will gain us lots of experience that we can later put to work on the Internet itself whenever society is ready to do that.

6.24. What’s a network telescope?

The term (which I believe is due to the guys at CAIDA) refers to being able to monitor a large set of addresses in order to study (or react to) the stray traffic coming into them. This has been very useful for studying both worms and DDOS attacks. Several groups have managed to set up telescopes on the Internet that capture traffic from a whole /8 or more. A distributed telescope is even better - this consists of a number of prefixes all of which get routed to the telescope infrastructure. Telescopes can be used for early detection of worms, as well as spread characterization.

You can set up your own telescope on your internal network. Choose a bunch of unallocated address space and tell you routers to send all traffic to those address ranges to an IDS box in the corner of your office. When it starts to smoke, you know
there’s a worm on the network. Note that this will only ever work for scanning worms however - telescopes are intrinsically incapable of picking up topological worms.

Can flash or topological worms be contained? This is an area of very active research and development at Silicon Defense as well as elsewhere. Since GrIDS, there have been techniques for detecting these kinds of worms, we aren’t aware of current containment systems for these spread algorithms that have a quantifiable performance. Expect them to emerge over the next few years.

6.25. I want to study my MS/PhD thesis on worms/worm containment. Where should I study?

The coolest academic groups working on worms (in our none-too-humble opinion) are:

- The folks at UC San Diego (Stefan Savage and colleagues in the CS department, and David Moore and colleagues at CAIDA).
- Don Towsley’s group at University of Massachusetts at Amherst.
- Also, Karl Levitt, Jeff Rowe and company at UC Davis have made a number of seminal contributions in practical Internet Security, and lately have done more worm work.
- Finally, one could go to UC-Berkeley and then see if it was possible to intern at ICIR with Vern Paxson and Nick Weaver.

6.26. What papers should I read to find out more about worm containment?

We have a bibliography of course, but if you just want to read a few papers we recommend:

- David Moore and co. Internet Quarantine: Requirements for Containing Self-Propagating Code
- Matthew Williamson. Throttling Viruses: Restricting Propagation to Defeat Mobile Malicious Code (we’ll forgive him for calling worms viruses)
- and if you’ll forgive us blowing our own horn, Containment of Scanning Worms in Enterprise Networks.