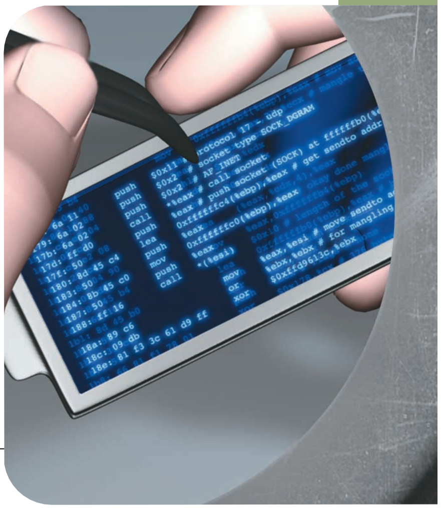
Inside the Slammer Worm



DAVID MOORE
Cooperative Association for Internet Data Analysis and University of California, San Diego

VERN PAXSON
International Computer Science Institute and Lawrence Berkeley National Laboratory

STEFAN SAVAGE
University of California, San Diego

COLLEEN SHANNON
Cooperative Association for Internet Data Analysis

STUART STANIFORD
Silicon Defense

NICHOLAS WEAVER
Silicon Defense and University of California, Berkeley

The Slammer worm spread so quickly that human response was ineffective. In January 2003, it packed a benign payload, but its disruptive capacity was surprising. Why was it so effective and what new challenges do this new breed of worm pose?

Slammer (sometimes called Sapphire) was the fastest computer worm in history. As it began spreading throughout the Internet, the worm infected more than 90 percent of vulnerable hosts within 10 minutes, causing significant disruption to financial, transportation, and government institutions and precluding any human-based response. In this article, we describe how it achieved its rapid growth, dissect portions of the worm to study some of its flaws, and look at our defensive effectiveness against it and its successors.

Slammer began to infect hosts slightly before 05:30 UTC on Saturday, 25 January 2003, by exploiting a buffer-overflow vulnerability in computers on the Internet running Microsoft's SQL Server or Microsoft SQL Server Desktop Engine (MSDE) 2000. David Litchfield of Next Generation Security Software discovered this underlying indexing service weakness in July 2002; Microsoft released a patch for the vulnerability before the vulnerability was publicly disclosed (www.microsoft.com/security/slammer.asp). Exploiting this vulnerability, the worm infected at least 75,000 hosts, perhaps considerably more, and caused network outages and unforeseen consequences such as canceled airline flights, interference with elections, and ATM failures (see Figure 1).

Slammer's most novel feature is its propagation speed. In approximately three minutes, the worm achieved its full scanning rate (more than 55 million scans per second), after which the growth rate slowed because significant portions of the network had insufficient bandwidth to accommodate more growth.

Although Stuart Staniford, Vern Paxson, and Nicholas Weaver had predicted rapid-propagation worms on theoretical grounds,[1] Slammer provided the first real-world demonstration of a high-speed worm's capabilities. By comparison, Slammer was two orders of magnitude faster than the Code Red worm, which infected more than 359,000 hosts on 19 July 2001,[2] and had a leisurely 37 minutes of population doubling time.

While Slammer had no malicious payload, it caused considerable harm by overloading networks and disabling database servers. Many sites lost connectivity as local copies of the worm saturated their access bandwidths. Although most backbone providers appeared to remain stable throughout the epidemic, there were several reports of Internet backbone disruption. For a single snapshot of the activity, see www.digitaloffense.net/worms/mssql_udp_worm/internet_health.jpg. Additionally, Tim Griffin of AT&T Research, has plotted Internet routing data (an overall view of Internet routing behavior) that shows a substantial perturbation in network connectivity resulting from Slammer's spread, (www.research.att.com/~griffin/bgp_monitor/sql_worm.html).

If the worm had carried a malicious payload, attacked a more widespread vulnerability, or targeted a more popular service, its effects would likely have been far more severe.

For more on how we tracked Slammer, see the "Network telescope operations" sidebar.

How Slammer chooses its victims

The worm's spreading strategy uses random scanning—it randomly selects IP addresses, eventually finding and infecting all susceptible hosts. Random-scanning worms initially spread exponentially, but their rapid new-host

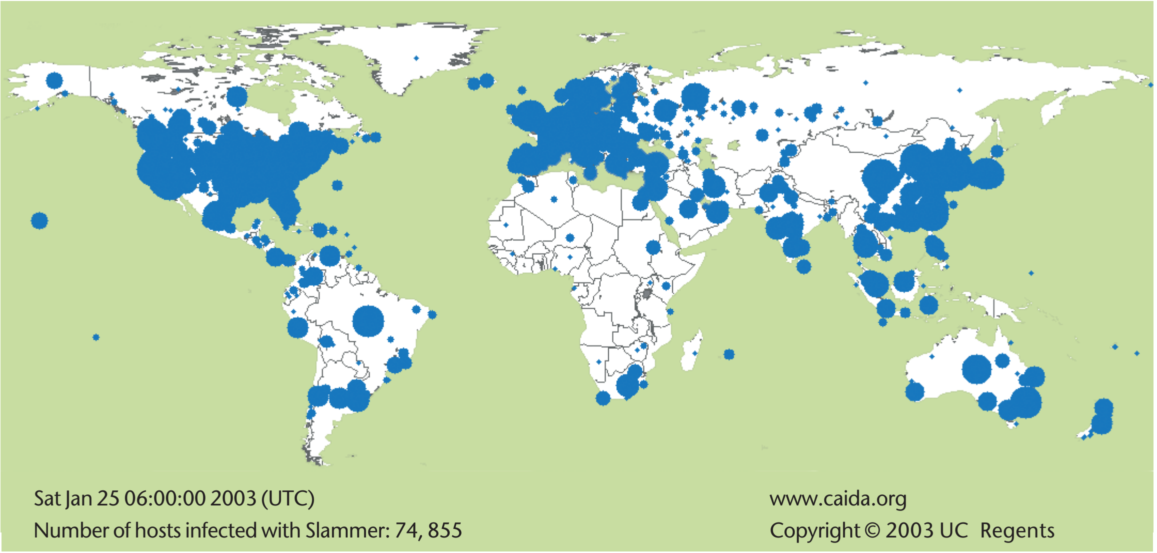

Sat Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Slammer: 74, 855
www.caida.org
Copyright © 2003 UC Regents

Figure 1. The geographical spread of Slammer in the 30 minutes after its release. The diameter of each circle is a function of the logarithm of the number of infected machines, so large circles visually underrepresent the number of infected cases in order to minimize overlap with adjacent locations. For some machines, we can determine only the country of origin rather than a specific city.

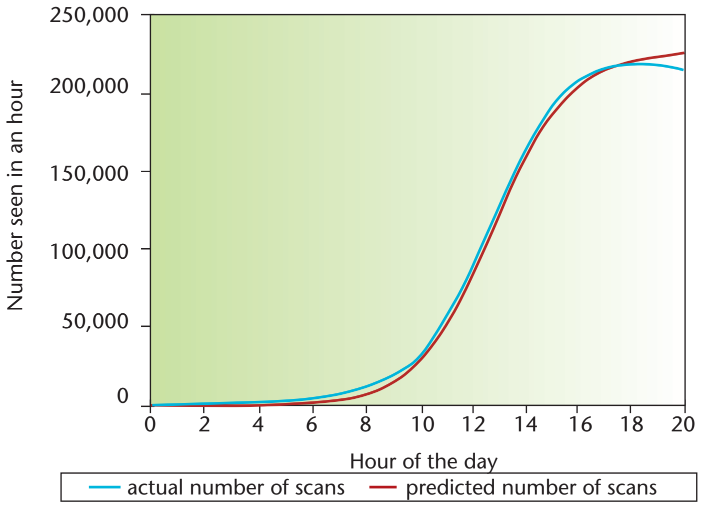


Figure 2. The Code Red worm was a typical random-scanning worm. This graph shows Code Red's probe rate during its re-emergence on 1 August, 2001, as seen on one Internet subnetwork, matched against the random constant spread worm behavior model.

infection slows as the worms continually retry infected or immune addresses. Thus, as with the Code Red worm shown in Figure 2, Slammer's infected-host proportion follows a classic logistic form of initial exponential growth in a finite system.[1,2] We label this growth behavior a *random constant spread* (RCS) model.

Slammer's spread initially conformed to the RCS model, but in the later stages it began to saturate networks with its scans, and bandwidth consumption and network outages caused site-specific variations in its observed spread. Figure 3 shows a data set from the Distributed Intrusion Detection System project (Dshield; www.dshield.org) compared to an RCS model. The model fits extremely well up to a point where the probe rate abruptly levels out. Bandwidth saturation and network failure (some networks shut down under the extreme load) produced this change in the probe's growth rate.

Why Slammer was so fast

While Slammer spread nearly two orders of magnitude faster than Code Red, it probably infected fewer machines. Both worms use the same basic scanning strategy to find vulnerable machines and transfer their exploitive payloads; however, they differ in their scanning constraints. While Code Red is latency-limited, Slammer is bandwidth-limited, enabling Slammer to scan as fast as a compromised computer can transmit packets or a network can deliver them. Slammer's 376 bytes comprise a simple, fast scanner. With its requisite headers, the payload becomes a single 404-byte user diagram protocol (UDP) packet. Contrast Slammer's 404 bytes with Code Red's 4 Kbytes or Nimda's 60 Kbytes.

Previous scanning worms, such as Code Red, spread via many threads, each invoking `connect()` to open a TCP session to random addresses. Consequently, each thread's scanning rate was limited by network latency. After sending a `TCP SYN` packet to initiate the connection, each thread must wait to receive a corresponding `SYN/ACK` packet from the target host or time-out if no re-

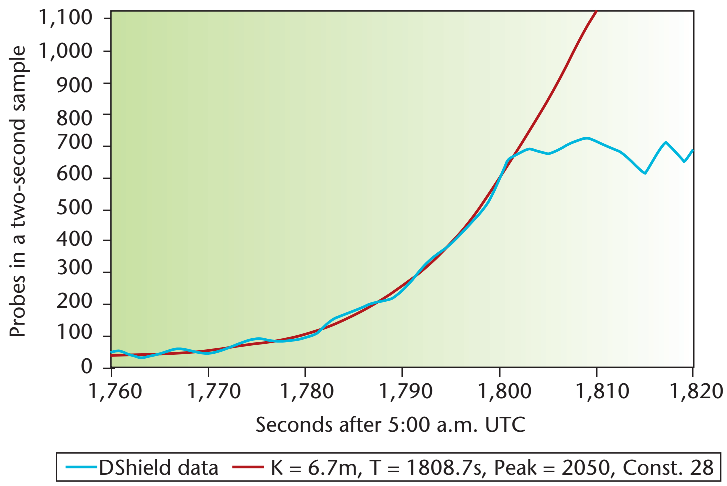


Figure 3. The early moments of the Distributed Intrusion Detection System (Dshield) data set, matched against the behavior of a random-scanning worm.

sponse is received. During this time, the thread is blocked and cannot infect other hosts. In principle, worms can compensate for this latency by invoking a sufficiently large number of threads. In practice, however, operating system limitations, such as context-switch overhead and kernel stack memory consumption, limit the number of active threads a worm can use effectively. So, a worm like Code Red quickly stalls and becomes latency limited, as every thread spends most of its time waiting for responses.

In contrast, Slammer's scanner is limited by each compromised machine's Internet bandwidth. Because a single packet to UDP port 1434 could exploit the SQL server's vulnerability, the worm was able to broadcast scans without requiring responses from potential victims. Slammer's inner loop is very small, and with modern servers' I/O capacity to transmit network data at more than 100 Mbits per second, Slammer frequently was limited by Internet access bandwidth rather than its ability to replicate copies of itself.

In principle, an infected machine with a 100-Mbps Internet connection could produce more than 30,000 scans per second. In practice, bandwidth limitations and per-packet overhead limit the largest probe rate we directly observed to 26,000 scans per second, with an Internet-wide average of approximately 4,000 scans per second per worm during its early growth phase.

Slammer's scanning technique is so aggressive that it quickly interferes with its own growth. Subsequent infections' contribution to its growth rate diminishes because those instances must compete with existing infections for scarce bandwidth. Thus, Slammer achieved its maximum Internet-wide scanning rate in minutes.

Any simple-loop approach creates a bandwidth-limited UDP scanner, so any future single-packet UDP worm probably would have the same property unless its author deliberately limited its spread. While a TCP-based worm, such as Code Red, also could use a bandwidth-limited scanner by sending `TCP SYN`s at maximum rate and responding automatically to any replies in another thread, this would require more effort to implement correctly, as it requires crafting raw TCP packets instead of simply using existing system calls.

What Slammer's author did wrong

For a random-scanning worm to be effective, it needs a good source of random numbers to select new attack targets. Slammer's random-number generator has some interesting deficiencies that make our analysis difficult and, perhaps, have implications for future worms. Several disassembled versions of the worm's source code are available at the URLs shown in the "Worm guts" sidebar.

Slammer uses a linear congruent, or power residue, pseudo random number generation (PRNG) algorithm. These algorithms take the form: $x' = (x \times a + b) \bmod \mathrm{m}$, where x' is the new pseudo random number to be generated, x is the last pseudo random number generated, m represents the range of the result, and a and b are carefully chosen constants. Linear congruent generators are very efficient and, with appropriate values of a and b have reasonably good distributional properties (although they are not random from a sequential standpoint). Typically, you "seed" a generator's initial value with a source of high-quality random numbers to ensure that the precise sequence is not identical between runs.

Slammer's author intended to use a linear congruent parameterization that Microsoft popularized, $x' = (x \times 214013 + 2531011) \bmod 2^{32}$. However, we found two implementation mistakes. First, the author substituted a different value for the 2531011 increment value: hex 0xFFD9613C. This value is equivalent to −2531012 when interpreted as a twos-complement decimal. So, it seems likely that the conversion to a negative number was an error (the author seems to have forgotten that creating a negative number in twos complement requires inverting and adding 1, not simply inverting), and probably that the author intended to use the `SUB` instruction to compensate for the resulting negative number, but mistakenly used `ADD` instead. The negative constant would be more desirable in the code, as this would eliminate any null (all-zero) characters from the worm's code. The result is that the increment is always even.

The author's second mistake was to misuse the `OR` instruction, instead of `XOR`, to clear an important register. This error left the register's previous contents intact. As a result, the increment inadvertently `XOR`s with the contents of a pointer contained in SqlSort's import address table (IAT). This "salt" value differs, depending on the SqlSort DLL's version, although two common values we observed are 0x77F8313C and 0x77E89B18. eEye Digi-

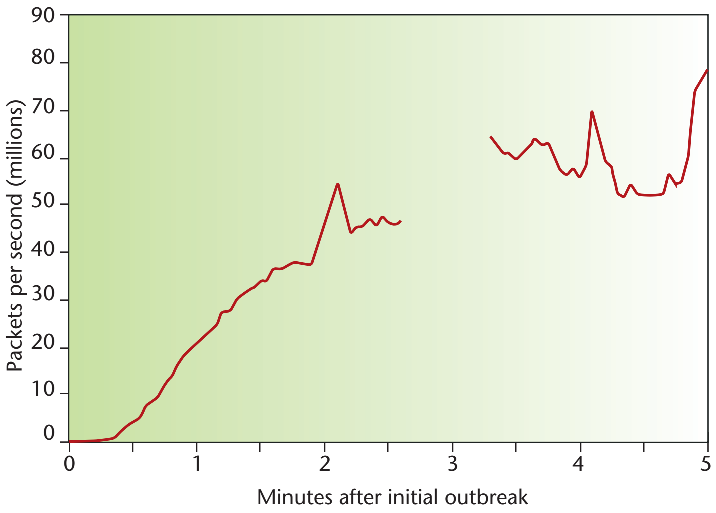


Figure 4. Slammer's early progress as measured at the University of Wisconsin Advanced Internet Lab (WAIL) tarpit, an unused network that logs packet traffic. Scanning rate is scaled to estimate the Internet-wide scanning rate. (A transient data-collection failure temporarily interrupted this data set approximately two minutes and 40 seconds after Slammer began to spread.)

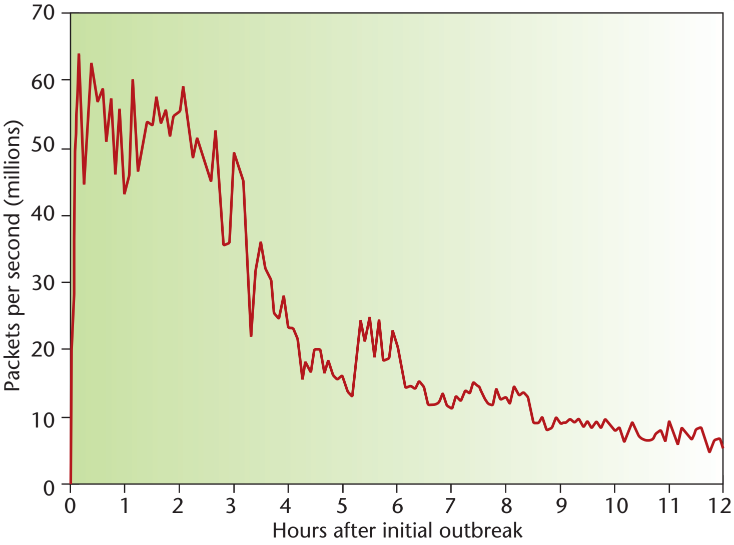


Figure 5. The response to Slammer during the 12 hours after its release, measured in several locations. Scanning rate is scaled to estimate the Internet-wide scanning rate.

tal Security (www.eeye.com/html/Research/Flash/sapphire.txt) also reports seeing 0x77EA094C, which is what the register value needs to be to get what eEye says is their result.

These mistakes significantly reduce the generator's distribution quality. Because b is even and the salt is always 32-bit aligned, the least-significant two bits are always zero. Interpreted as a big-endian IP address (the most significant value in the sequence is stored at the lowest storage address), this ensures that the 25th and 26th bits in the scan address (the upper octet) remain constant in any worm execution instance. Similar weaknesses extend to the 24th bit of the address, depending on the uncleared register's value. Moreover, with the incorrectly chosen increment, any particular worm instance cycles through a list of addresses significantly smaller than the actual Internet address space. Thus, many worm instances will never probe our monitored addresses because none of them are contained in the worm's scan cycle. Combined with the size of our monitored address space,[3] these mistakes prevent us from accurately measuring the number of infected hosts during the first minutes of the worm's spread.

Slammer will or will not include entire /16 address blocks (large contiguous address ranges) in a cycle, because the last two bits of the first address byte never change. We were able to assemble lists of the address blocks in each cycle for each value of the salt (cycle structure depends on salt value). Fortunately, the probability of choosing a particular cycle is directly proportional to the size of the cycle if the initial seed is selected uniformly at random. If we looked at many randomly seeded worms, it is likely that all Internet addresses would be probed equally. Thus, we can accurately estimate the worm's scanning rate during the infection process by monitoring relatively small address ranges. We can estimate the percentage of the Internet that is infected because the probing will cover all Internet addresses.

If not for the higher-quality numbers in the initial seed, these flaws would prevent the worm from reaching large portions of the Internet address space, no matter how many hosts were infected. For the same reason, these flaws also could bias our measurements because even though our data come from several different networks, there is a small chance that those particular networks were disproportionately more or less likely to be scanned. However, the worm uses an operating system service, `GetTickCount`, to seed each generator with the number of milliseconds since a system booted, which should provide sufficient randomization to ensure that across many instances of the worm, at least one host will probe each address at some point in time. We feel confident that the risk of bias in our measurements is similarly minimized. Additionally, the "best" random bits produced by `GetTickCount` are in the least-significant bits, which determine which cycle is selected for a given salt.

An interesting feature of this PRNG is that it makes it difficult for the Internet community to assemble a list of the compromised Internet addresses. With earlier worms, we could collect a list of all addresses that probed into a large network. With Slammer, we would need to monitor networks in every cycle of the random-number generator for each salt value to have confidence of good coverage.

What did we learn about Slammer's author? See the "Who wrote Slammer?" sidebar.

How the Internet responded

By passively monitoring traffic (either by sniffing or sampling packets or monitoring firewall logs) on a set of links providing connectivity to multiple networks, each responsible for about 65,000 IP addresses, we were able to infer the worm's overall scanning behavior over time.

We obtained the most-accurate Slammer early-progress data from the University of Wisconsin Advanced Internet Lab (WAIL), which logs all packet traffic into an otherwise unused network, a "tarpit" (see Figure 4). Because this data set represents a complete trace of all packets to an address space of known size, it lets us accurately extrapolate the worm's global spread. Unfortunately, a transient failure in data collection temporarily interrupted this data set approximately 2 minutes and 40 seconds after Slammer began to spread. Our other sampled data sets are not sufficiently precise for accurate evaluation over short durations.

In general, people responded quickly to Slammer. Within an hour, many sites began filtering all UDP packets with a destination port of 1434 via router or firewall configuration changes. Slammer represents the idealized situation for network-based filtering: its signature easily distinguishes it, it is readily filterable on current hardware, and it attacked a port that is not generally used for critical Internet communication. Thus, almost all traffic blocked by these filters represents worm-scanning traffic. If the worm had exploited a commonly used service vulnerability (for example, DNS at UDP port 53 or HTTP at TCP port 80), filtering could have caused significant disruption to legitimate traffic, with resulting denial of service (DoS) more harmful than the worm itself. Figure 5 illustrates the results of filtering.

Regardless of the optimal filter efficacy conditions in this instance, we must recognize that while filtering controlled the unnecessary bandwidth consumption of infected hosts, it did nothing to limit the worm's spread. The earliest filtering began long after Slammer had infected almost all susceptible hosts.

We recorded all distinct infected IP addresses seen by our monitors in the first 30 minutes of worm spread. We noticed 74,856 distinct IP addresses, spread across a wide range of domains and geographic locations. Slammer infected most of these machines in the first few minutes, but monitoring limitations prevent us from telling precisely when they were infected. We cannot observe all infected machines due to the flaws in Slammer's PRNG, but we can document a lower bound on the number of compromised machines based on the IP addresses we have recorded—the actual infection is undoubtedly larger. Tables 1 and 2 summarize these distributions.

Table 1. Slammer's geographical distribution.

COUNTRY	PERCENT VICTIMS
United States	42.87
South Korea	11.82
Unknown	6.96
China	6.29
Taiwan	3.98
Canada	2.88
Australia	2.38
United Kingdom	2.02
Japan	1.72
Netherlands	1.53

Table 2. Slammer's top-level domain distribution.

TOP-LEVEL DOMAIN	PERCENT VICTIMS
Unknown	59.49
.net	14.37
.com	10.75
.edu	2.79
.tw	1.29
.au	0.71
.ca	0.71
.jp	0.65
.br	0.57
.uk	0.57

Why Slammer caused problems

Although Slammer did not contain an explicitly malicious payload, there were widely reported incidences of disruption, including failures of Bellevue, Washington's 911 emergency's data-entry terminals and portions of Bank of America's ATM network. The 911 and ATM failures were widely reported: Aaron Davis, "Computer Worm Snarls Web" (www.bayarea.com/mld/mercurynews/ 5034748.htm); Robert Lemos, "Slammer Attacks May Become Way of Life for the Net" (http://news. com.com/2009-1001-983540. html?tag=fd_lede2_hed); and Robert O'Harrow Jr., "Internet Worm Unearths New Holes" (www.securityfocus. com/news/2186). Likewise, many routers and switches crashed under the load, including several layer-2 and layer-3 switches in the University of California, Berkeley's Computer Science department.

Inadvertent internal DoS attacks caused the large majority of these disruptions: as one or more infected machines sent out packets at their maximum possible rates. This traffic either saturated the first shared bottleneck or crashed some network equipment. The bottleneck effects are obvious, as a site's outgoing bandwidth is usually significantly less than a Slammer's instance can consume. Thus, the worm's packets saturated Internet links, effectively denying connec-

Network telescope operations

We tracked Slammer's spreading behavior via a network telescope technique—a large address range, which is monitored for unusual activity.[1] Ideally, these address ranges are unused but routed, which eliminates all normal activity, but they can contain actual machines if the normal traffic can be excluded from further analysis. Because Slammer's behavior was highly anomalous, intrusion detection system logs and other sources provided excellent telescopes, even when gathered from active networks.

David Moore, Geoffrey Volkier, and Stefan Savage developed network telescope technology to understand distributed denial-of-service attacks, which generate a considerable amount of "backscatter," or responses to packets with randomly forged source addresses.[2] By monitoring these responses, we can quantify the scope of Internet denial of service activity, including targets and estimated traffic rates. This technique's sensitivity depends on the size and distribution of the monitored address ranges; larger ranges are significantly more precise while distributed ranges are more likely to detect attacks that contain biases in address selection.

We tracked Slammer by monitoring the worm's probes to address ranges scattered around the Internet. Combined with address range size and distribution, we can estimate the amount of Internet scanning occurring and the worm's early growth rate. With this scanning-rate information, we can estimate the fraction of the vulnerable machines that would be infected.

This technique has some limitations. We cannot determine when a particular machine became infected, because the telescopes only detect an infected machine when it first probes a telescope. Furthermore, our telescopes were more limited than we desired because of relatively small monitored region plus some data set subsampling. As a result, we effectively monitored 1/214 of the complete Internet address space, enough to monitor the worm's behavior and discover most infected machines but not enough to reveal the initial point of infection.

References

1. D. Moore, "Network Telescopes: Observing Small or Distant Security Events," Invited presentation at the 11th Security Symp. (SEC '02), www.caida.org/outreach/presentations/2002/usenix_sec/.
2. D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," *Proc. 10th Security Symp.* (SEC '01), Usenix, Aug. 2001, pp. 9–22, www.caida.org/outreach/papers/2001/BackScatter.

Who wrote Slammer?

After our study, we found very few clues about Slammer's author's identity, location, or motive. No one has claimed authorship and the code does not contain identifying signatures or comments. There also are no variable names in the code, making it impossible to determine the author's native tongue.

Similarly, the author has decent, but not remarkable, x86 coding skill; much of the code was borrowed from a published exploit, the additional code is not very complicated, and the author made three minor mistakes in the random-number generator.

Finally, no one has discovered patient 0, the initial point of infection. Thus, it is impossible to determine where the worm was released, in hopes of tracing the release back to the author.

tivity for all computers at many infected sites.

Equipment failures tended to be a consequence of Slammer's traffic patterns generated by infected machines, although any given equipment's failure details varied. Slammer's scanner produced a heavy load in three ways: a large traffic volume, lots of packets, and a large number of new destinations (including multicast addresses). We feel this combination probably caused most network-equipment failures by exhausting CPU or memory resources.

If attackers can control a few machines on a target network, they can perform a DoS attack on the entire local network by using a program that mimics Slammer's behavior. Because these are "normal" UDP packets, special privileges (such as root or system administrator abilities) are not required. Instead, they need only the ability to execute the attacker's program. Thus, critical networks should employ traffic shaping, fair queuing, or similar techniques, to prevent a few machines from monopolizing network resources.

Although some had predicted the possibility of high-speed worms,[1] Slammer represents the first super-fast worm released into the wild. Microsoft's SQL Server vulnerability was particularly well suited for constructing a fast worm (because the exploit could be contained in a single UDP packet). However, techniques exist to craft any worm with a reasonably small payload into a bandwidth-limited worm similar to Slammer. Thus, we must not view Slammer's extreme speed as an aberration of the exploit's nature or the particular protocol used (UDP versus TCP).

One implication of this worm's emergence is that smaller susceptible populations are now more attractive to attacks. Formerly, attackers did not view small populations (20,000 machines or less on the Internet) as particularly vulnerable to scanning worms because the probability of finding a susceptible machine in any given scan is quite low. However, a worm that can infect 75,000 hosts in 10 minutes could infect 20,000 hosts in under an hour. Thus, vulnerabilities of less popular software present a viable breeding ground for new worms.

Because high-speed worms are no longer a theoretical threat, we need to automate worm defenses; there is no conceivable way for system administrators to respond to threats of this speed.[1,4] Human-mediated filtering provides no benefit for actually limiting the number of infected machines. While the filtering could mitigate the overhead of the worm's continuing scan traffic, a more sophisticated worm might stop scanning once the entire susceptible population was infected, leaving itself dormant on more than 75,000 machines to do harm at some future point. Had the worm's propagation lasted only 10 minutes, it would likely have taken hours or days to identify the attack, and many compromised machines might never have been identified. Thus, it is critical that we develop new techniques and tools that automatically detect and respond to worms.

Though very simple, Slammer represents a significant milestone in the evolution of computer worms. It is sobering that worms such as Slammer preclude human-time response and, to date, deter attribution and prosecution. It clearly demonstrates that fast worms are a reality and we should consider them as standard tools in an attacker's arsenal. □

Acknowledgments

We thank Dave Plonka (University of Wisconsin-Madison and the Wisconsin Advanced Internet Lab), Pat Wilson and Brian Kantor (University of California, San Diego), and Johannes Ullrich and the SANS Institute for contributing data sets for our analysis. Thanks also to Lea Kissner (Carnegie-Mellon University) and David Wagner (University of California, Berkeley) for help with the pseudo random number generator. We are grateful to Ryan Koga (CAIDA) and Jeffrey Brown (University of California, San Diego) for geographically mapping the worm's spread. The US National Science Foundation, Silicon Defense, Cisco Systems, AT&T, US National Institute of Standards and Technology, the Defense Advanced Research Projects Agency, and CAIDA members provided support for this work.

References

1. S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time," *Proc. 11th Security Symp.* (SEC 02), Usenix Assoc., 2002, pp. 149–167.
2. D. Moore, C. Shannon, and J. Brown, "Code-Red: A Case Study on The Spread and Victims of an Internet Worm," *Proc. 2nd ACM Internet Measurement Workshop*, ACM Press, 2002, pp. 273–284.
3. D. Moore, "Network Telescopes: Observing Small or Distant Security Events," Invited presentation at the 11th Usenix Security Symp. (SEC 02), Aug. 2002, www.caida.org/outreach/presentations/2002/usenix_sec.
4. David Moore et al., "Internet Quarantine: Requirements for Containing Self-Propagating Code," to appear in *Proc. 2003 IEEE Infocom Conf.*, IEEE Press, Apr. 2003.

David Moore *is a principal investigator and assistant director of the Cooperative Association for Internet Data Analysis (CAIDA). He is a computer science PhD candidate at the University of California, San Diego. His research interests are high-speed network monitoring, denial-of-service attacks and infrastructure security, and Internet traffic characterization. Contact him at dmoore@caida.org.*

Worm guts

You can inspect some disassembled versions of Slammer's source code at the following URLs:

- www.nextgenss.com/advisories/mssql-udp.txt
- www.boredom.org/~cstone/worm-annotated.txt
- www.eeye.com/html/Research/Flash/sapphire.txt

Vern Paxson *is a senior scientist with the International Computer Science Institute (ICSI) Center for Internet Research (ICIR), a non-profit research institute in Berkeley, Calif. He is also a staff computer scientist at the Lawrence Berkeley National Laboratory (LBNL). He received a PhD in computer science from the University of California, Berkeley. His research focuses on network intrusion detection, Internet attacks, and Internet measurement. He serves on the editorial board of* IEEE/ACM Transactions on Networking*, and as chair of the Internet Research Task Force, as well as program co chair of ACM SIGCOMM 2002, program chair of Usenix Security 2003, and on the steering committee of the Internet Measurement Workshop. Contact him at vern@icir.org.*

Stefan Savage *is an assistant professor of computer science and engineering at the University of California, San Diego. He received a PhD from the University of Washington. In addition to network security, he is interested in the design, implementation, and evaluation of networks and distributed systems. He is program chair of the 2003 ACM Workshop on Rapid Malcode (WORM) and coprogram chair of the 2004 ACM Symposium on Networked Systems Design and Implementation (NSDI). Contact him at savage@cs.ucsd.edu.*

Colleen Shannon *is a staff researcher at the Cooperative Association for Internet Data Analysis (CAIDA). Her research interests focus on network security, including traffic measurement, characterization, and network application identification. She recently completed a study of the prevalence and characteristics of IP packet fragmentation, the results of which were published in* IEEE Transactions on Networking *in December 2002. Contact her at cshannon@caida.org.*

Stuart Staniford *is founder and president of Silicon Defense, which focuses on Internet security research and commercial products. He received an MSc in Computer Science and a PhD in physics from the University of California, Davis. His research interests include worm containment, intrusion correlation, and intrusion detection. Contact him at stuart@silicondefense.com.*

Nicholas Weaver *is currently completing his PhD at the University of California, Berkeley, having previously completed a BA in Computer Science and Astrophysics at Berkeley. His primary research areas are field-programmable gate array (FPGA) architectures, applications, and tools, as well as computer security. His FPGA work, which forms the core of his dissertation, is a novel FPGA architecture which supports fast routing, pipelined interconnect, and fixed-frequency operation. His security work focuses on Internet scale attacks and potential automatic defenses. Contact him at nweaver@cs.berkeley.edu.*