

A Brief History of The Worm

Nicholas Weaver 2001-11-26

A Brief History of The Worm

by *Nicholas Weaver*

last updated November 26, 2001

"You will REALLY love it when Word is part of Navigator, and macro viruses will be able to COMMUNICATE!" - The author in electronic chat, quoted in 1997.

Self-replicating malware has been an issue in computer security for many years, dating back at least to Ken Thompson's self replicating code. But in the past few years, with the widespread adoption of the Internet, worms and viruses have become serious pests: spreading around the world in a matter of hours with the capacity to carry highly damaging payloads. Such malware is growing more sophisticated, as the authors of new worms learn from the successes and mistakes of the past. This article will take a brief look at the evolution of worms and other malware, in an attempt to better understand how we got to where we are today.

In order to understand current malware, and to anticipate future developments in malware production, an analysis of some previous major incidents will be useful. Not only does such an analysis provide a model for defenses against current worms, it offers a view of potential future worms. There are two areas of special interest: the means of spreading the worm and the potential malicious content.

Propagation

The existence of self replicating programs dates back to the early Unix era with Ken Thompson's self replicating code, a compiler modification which would trojan login and the compiler itself [see Thompson's excellent Turing award paper, [Reflections on Trusting Trust](#)] Starting in the Apple II era, the conventional virus became a fairly common plague: a bit of executable code would insert itself into other executables. Fortunately the spread of these conventional file viruses is fairly slow, as they required human intervention to transfer themselves from machine to machine. Nevertheless, this mechanism provided the means of spreading for some famous viral non-events such as Michelangelo and Chernobyl.

The first active Internet worm that required no human intervention to spread was the Morris

worm released in 1988. It spread very rapidly, infecting all vulnerable machines in a matter of hours. Most recent active worms use the techniques pioneered by Robert Morris. The Morris Worm infected multiple types of machines (Sun 3s and VAXes), attacked multiple security holes (including a buffer overflow in fingerd, debugging routines in Sendmail, and password cracking), and used multiple streams of execution to improve its throughput when attacking other machines.

Although intended to be a benign proof of concept, the Morris worm had a significant impact due to a bug in the code. When it reinfected a machine, there was a fixed chance that the new infection wouldn't quit, causing the number of running worms on a machine to build up, thereby causing a heavy load on many systems. Even on a modern machine, such bugs would have a similar effect of overwhelming the system. This caused the worm to be quickly noticed and caused significant disruption. Most subsequent worms have mechanisms to prevent this from happening.

In 1996, the first Word macro virus appeared and became quickly widespread. This was due to two reasons: the far greater tendency for people to exchange documents, as opposed to executables, and the accidental inclusion of the virus on at least two Microsoft CDs. For the most part these were just annoyances, but they showed how the blurring of data and programs could create fertile ground for mobile code.

All this changed in 1999 when the Melissa worm appeared. Unlike previous macro viruses, this one would spread in a semi-active manner. When an infected file was opened for the first time, it looked through all Outlook address books and sent a copy of itself to the first 50 individuals. This was the first major e-mail worm and it quickly spread around the globe. Although generally benign in intent (the worm itself only sent a small message and the payload was simply a Simpsons joke,) the process of transferring so many messages overwhelmed some e-mail servers. The Melissa worm clearly illustrated the dangers of mixing code and data: items perceived by the user as benign data could contain malware.

After Melissa, mail worms have become annoyingly common, complete with toolkits. There have been some improvements in social engineering (ILOVEYOU and AnnaKornikova showed how proper subject choice can make a difference in the successful proliferation of a worm,) more comprehensive searches for new addresses, included SMTP routines, and payload (Magistr tries to actively disrupt removal and SirCam demonstrated how worms could be used for espionage.) But otherwise these worms are rather unoriginal. Nevertheless, since they continue to spread,

they have remained a significant problem and must be considered a major threat.

Active worms have recently returned to prominence. The first one that attracted major attention, Code Red, demonstrated how swiftly a relatively simple worm can spread on the current Internet infrastructure: it effectively achieved complete infection in a little over twelve hours, even with the aborted early release of a buggy version. Code Red exploited a recently discovered (but patchable) buffer overflow attack in Microsoft's Internet Information Server. It spread far and fast because of the "on by default" nature of IIS with many versions of Windows NT and 2000. It also included multithreaded scanning routines that improve throughput and effectively keep it from being trapped by tarpits (such as [LaBrea](#)), which are blocks of IP addresses that attempt to slow down scanning by automated tools by seeming to respond to connection requests while actually doing nothing.

Code Red 2 ended up being significantly more disruptive than Code Red even if the change in infection strategy was relatively mild. Instead of searching only randomly selected addresses, Code Red 2 preferentially probed for machines on the same subnet and nearby subnets. As a result, once a single machine within a corporate firewall was infected, it would quickly probe virtually every machine within the firewall and since it was attacking an on-by-default service, Code Red 2 quickly infested entire corporate networks.

But even before the release of Code Red, active worms were spreading, albeit with less publicity. A good example was the March 2000 release of the 911 worm. It spread through Windows shares, a slow but autonomous mechanism. It would search portions of the Net for remotely writeable C drives and then inject a copy of itself into the victim's startup routines. The next time the victim computer started up their system, the worm would begin running. Worms that use only this mechanism are comparatively slow spreading because of the need to have the machine rebooted before infection is complete.

Nimda

The latest worm of note, Nimda, did not really bring anything new to the table. It simply resurrected the idea of multimode operation: it was an e-mail worm, it attacked old bugs in Explorer and Outlook, spread through Windows shares, and an old buffer overflow in IIS. It also borrowed Code Red 2's preference for logically adjacent IP addresses in its scanning routines. The net result was a highly virulent, highly effective worm that revealed that several old bugs can be used even if each hole is patched by most machines: one needs all patches and

vulnerabilities closed to stop a Nimda-like worm. Such a worm is also somewhat easier to write, as one can use many well-known exploits to get wide distribution instead of discovering new attacks.

Nimda proved to be surprisingly effective, for although each hole that it exploited was relatively rare, the combination of attacks proved successful at attacking large numbers of machines. Since worms that can infect a higher percentage of machines actually spread faster, this increased its speed and breadth of proliferation. Of additional interest is how different attacks will circumvent various protection mechanisms. Firewalls are useless when a Web browser deliberately downloads and runs a piece of malicious code; because shared system drives are more common within the confines of a firewalled network, one errant e-mail infection can allow free spread within the firewall.

The biggest lesson from both Code Red and Nimda is the sheer speed with which these fairly simple active worms can spread. By the time humans detect the presence of worms on the Net, through firewall activity or probes of monitoring IP ranges, they may well have spread worldwide. There is a window of a few hours for response, since such worms start slowly, but it is questionable whether this time span is sufficiently long to mount human-mediated defenses.

Worse still, even faster infection strategies have been envisioned (see the description of [Warhol worms](#) and [Flash worms](#) for methods that, with various amounts of preparation, may allow a worm to infect all vulnerable machines in minutes) but have not yet been seen in practice. It is questionable whether someone interested in writing a superworm would need to bother with such techniques, since although significantly faster, greater speed may not be necessary.

Minimizing the Spread

The best way to defend against worms is obviously to prevent infection - once a single machine within a local subnet is infected it may be too late. One obvious defense is diversity, using less common operating systems and servers. Assuming there is no way to directly find vulnerable machines, the rate at which a worm spreads is directly proportional to the number of vulnerable hosts. This is because with fewer potential targets any given random probe is less likely to find a vulnerable machine, limiting the rate of spread. Thus, if one only has a few machines open to the wide internet, using a less common (i.e., non-Windows) OS offers an advantage, as there are less likely to be worms written for less common platforms, and those which are created will spread more slowly.

For example, Linux has had it's own fair share of worms, such as Ramen, Lion, and Cheese. But since the Ramen worm only attacked Red Hat 6.2 and 7.0 systems, it couldn't spread very fast when compared to Code Red, simply because the odds of any particular scan finding a vulnerable target are so much lower when compared to Windows worms. As an additional side benefit, most attempts to construct a worm for the purpose of wholesale information terrorism will target the most common platform, currently Windows, to increase the spread and damage done.

Firewalls are also essential to prevent both active worms as well as human intrusion. But external firewalls are not sufficient for larger institutions. With multimode worms like Nimda and local scanning worms such as Code Red, it becomes critical to split large internal networks with internal firewalls. Such a design is akin to how ships are designed with watertight compartments, the result being that a single intrusion should not allow a worm free reign in the corporate network.

Similarly, all executable content which crosses the firewall should be regarded as suspicious, especially e-mail. All macros and programs sent in e-mail should be quarantined to prevent future mail worms from spreading. There should be severe limits on what is allowed to cross firewalls, with a general attitude of "that which is not explicitly allowed is forbidden." If available, operating system options which only allow cryptographically signed code should be exercised, executable stacks should be disabled, and bounds-checking modifications should be used at all times.

Finally, there needs to be continued development of firewalls and anti-virus systems that detect and autonomously respond to new attacks. Since new viruses can spread much faster than humans can respond, the defenses need to be automated. Pure pattern matching methods, suitable against file and macro viruses or human-run exploit scripts, are not sufficient to defend against worms because worms can spread faster than updates are created. For the same reason anti-worms are not generally effective since by the time an anti-worm is ready for release it will be far too late.

Malicious Payloads

All worms need to be considered highly malicious, even if they spread in a seemingly benign manner, as most worms signal that a machine is vulnerable to further attack through its attempts to spread. A good example of the way in which this can be exploited is illustrated at

<http://www.dasbistro.com/default.ida>. This web page responds to a Code Red 2 probe (which tries to access default.ida with a string designed to cause an IIS buffer overflow) by scheduling a counterattack. The counterattack tries to use the Code Red 2 installed security hole and, if successful, disables IIS and resets the infected machine. It could have just as easily deleted all the files, installed Back Orifice, or performed any other malicious activity desired. A modified version could attack Code Red 1 infected machines by using the same buffer overflow exploit that Code Red used and similar scripts could respond to Nimda, Ramen, or other infections.

Apart from revealing that a machine is vulnerable, past malware has included some particularly malicious payloads, including drive erasers, data encrypters, and BIOS reflashers. More recently, Code Red included a buggy DDoS module, Code Red 2 and Nimda opened explicit system-level backdoors, and SirCam provided a crude espionage method. Other payloads have included Magistr's attempts to subvert removal by blocking access to anti-virus sites and the 911 worm's attempt to DDoS the 911 system by dialing the modem.

One malicious payload we haven't seen is cryptographically signed updates. In this scenario, each worm keeps track of the machines it infects and those that try to reinfect it. Then when a new code module is introduced (either to add a malicious attack, to rescind the worm, or to provide another means of infection,) the worm verifies that the code was signed by the appropriate private key, spreads a copy to every other worm it knows about, and then starts executing the new code. The new code would spread to all running copies in a matter of moments, allowing a great degree of flexibility in changing levels of maliciousness, debugging in the field, or seeking out new targets from an existing installed base of worms.

A combination of these techniques would create a particularly destructive attack. This works best with a fast worm, so it can be immediately malicious instead of relying on a long delay timer. Alternatives include a more subtle approach where the worm estimates the extant worm population and only becomes malicious when population levels peak. In either case, a fast worm can start doing significant damage before responses can be initiated.

Fortunately, many explicitly malicious payloads have failed to operate in practice. This is simply because a worm which has bugs in the distribution code will not spread properly, while a bad payload will still spread. But we can't count on future worms not possessing functional payloads, as they simply require proper testing before release. An intelligent adversary will undoubtedly debug malicious payloads before a worm is released.

Conclusions

Once a system is infected, there is not much that can be done to mitigate the damage. Regular backups are critical, as a malicious worm could easily overwrite or corrupt the existing data. Any reflashable BIOSes should be write-protected while software control of voltages, overtemperature set points, and clock rates should be disabled to prevent a malicious program from stressing a CPU through overtemperature and overvoltage conditions. Machines containing particularly sensitive information need to be completely isolated from the Net to prevent a SirCam-style worm or determined hacker from extracting information.

Just as one should have contingency plans in case of fire, those plans must also include worm attacks. There is no theoretically perfect defense against malicious worms: we can take steps to reduce their spread, prevent damage, audit code for security holes, construct solid firewalls, and maintain regular backups, but in the end, we need to accept that it is possible for someone following the lessons of past worms to create something highly disruptive and destructive. One should always assume that, in the end, one's computers may be highly disrupted by an electronic attack. To deny this would be folly.

Nicholas Weaver is a computer science graduate student at the University of California, Berkeley. His primary interest is FPGA and computer architecture, with secondary interests in computer security, molecular computation, and cryptographic implementations.

[Privacy Statement](#)

Copyright 2006, SecurityFocus