How To Hook Worms

Because a computer network cannot ward off every last Internet worm, it must sound an alarm the minute one slithers inside

By **JAMES RIORDAN, ANDREAS WESPI AND DIEGO ZAMBONI** Posted 2 May 2005 | 19:21 GMT

They were 376 bytes that shook the world.

At 5:30 a.m. Greenwich Mean Time on the morning of 25 January 2003, the Sapphire worm began dispatching copies of itself to the Internet. The worm, also known as SQL Slammer, began infecting computers running a very popular Microsoft database program, Microsoft SQL Server.

To infect a computer, the worm first sent itself to a specific communications port of the computer, one the SQL Server used to send and receive requests. When the computer attempted to process the "request," the worm caused a data buffer in the computer to overflow. The overflow in turn caused the computer to install Sapphire, which then sent copies of itself over the Internet. And so it went, computer after computer, with astonishing speed and efficiency.

Photo: Christopher Harting

The virus began infecting a widening circle of computers in a contagion that zoomed around the world, doubling every 8.5 seconds. By 5:40 a.m., just 10 minutes after it was unleashed, Sapphire had spread to at least 70 000 computers—90 percent of all the vulnerable machines in the world. The worm's paltry few hundred bytes carried no malicious payload and so deleted no data or software. But the sheer torrent of data coursing over the Internet consumed nearly all available capacity, crashing networks, bank ATMs, and flight-scheduling systems.

After the dust settled, a few days later, a London computer security firm, Mi2g Ltd., estimated that Sapphire had caused about US $1 billion in damages, related mostly to lost productivity.

Incredibly enough, Sapphire was at that time only the ninth most costly computer attack on record, according to the London firm. And it was an unfortunate harbinger of things to come: in February 2004 alone, Mi2g estimates, various malevolent attacks caused upwards of $68 billion in damages worldwide, much of it due to worms, such as MyDoom and several others that rampaged through the Internet that month. Ever since the first worms were released on the world in the late 1980s, those who write them and those who fight them—including the developers of computer intrusion-detection systems—have been engaged in a sort of arms race [see sidebar, Worm Evolution"].

At IBM Zurich Research Laboratory, we're working on a remedy for worms that differs from other approaches in targeting worms specifically rather than trying to prevent all breaches of computer security. Our system, called Billy Goat, does just one thing but does it extremely accurately.

Protection of a computer system begins with good locks, in the form of hardware and software barriers. But just as homeowners often keep watchdogs to sniff out a burglar even after he has gotten past a locked door, so do many of today's systems monitor suspicious activities that take place inside a computer.

To understand the problem , it helps to review the history of intrusion detection. The field is generally considered to have started in 1987 with a paper by Dorothy E. Denning, a computer scientist then at SRI International, Menlo Park, Calif. In "An Intrusion-Detection Model," published in "IEEE Transactions on Software Engineering", she described how to model the statistical characteristics of a system operating normally so that deviations from the model could be taken as evidence that intruders were present.

Intrusion-detection systems attempt to detect things that are "wrong" in a computer network or system. Because legitimate and illegitimate activities often look alike, the diagnosis depends heavily on the context. For instance, a single Web request sent to a computer may be innocuous. If the same request, however, is sent to a large number of computers across the planet, it may be part of reconnaissance performed to find potential attack targets.

The problem of noticing when things are amiss is compounded by the need to operate in an adversarial environment. Attackers pride themselves on foreseeing the responses of common detection systems and taking pains to sidestep or even exploit them.

Computer systems face a variety of threats—not only from worms but also from viruses and other forms of attack. Each of these threats is best detected by a different method.

A virus, for example, is a program that embeds itself within another program. It executes when that program executes, typically causing some mischief, like deleting data, altering a display, or scrolling a message. Just as human viruses need a cell to reproduce, computer viruses need a host program. They cannot spread from one computer to another on their own; usually, they get into a computer when a naive user runs an infected program, often by opening an infected e-mail attachment. Viruses can often be detected by observing changes in the files stored on the machine.

Worms, unlike viruses, are specifically designed to propagate through a network, usually the Internet, replicating themselves at each machine before jumping to new ones. No human action is needed for a worm to get from one computer to another. In most cases, the frenetic activity and communication the worm causes is itself the point: a worm like Sapphire creates so much network traffic so quickly that it overwhelms routers and other network nodes, in what is called a denial-of-service attack.

Thus, while viruses are an enormous problem, it is worms that generally get the most intense media coverage, because their outbreaks can affect many millions of people at the same time and cause significant economic havoc as well. There are straightforward ways of detecting a worm attack—a flooded network is a pretty good initial indicator—but, as recent experience has shown, many of the defensive measures don't function fast enough to contain the spread of an effectively written worm.

Most Intrusion-Detection Systems today take a dual approach, posting "sentinel" programs, which spot anomalies, both on individual ("host") computers attached to the network and on the network itself [see illustration, "Worms Crawl In but Don't Crawl Out"]. The host-based systems track activity more reliably but require installation on each machine and are therefore harder to manage. Network-based systems use a single device to monitor an entire network, but they are easier for intruders to deceive, and they may not be able to keep up with network traffic during an attack. Data encryption can also prevent a network-based system from analyzing the data stream to determine if it contains an attack.

Even this dual-tier approach has limitations. Like a watchdog, an intrusion-detection system can fail either by sleeping through an attack or by barking at shadows. It is hard to devise a system that does not habitually do one or the other. Most existing detection systems try to detect as many attacks and as many types of attack as possible, thereby sounding many false alarms. This leads users to lose faith in the system and finally to ignore it.

Today it seems clear that intrusion detection is at a critical point in its evolution. The main problems of these systems are the many false alarms they produce, their lack of resistance to both malicious attacks and accidental failures, and the constant appearance of new attacks and vulnerabilities. With respect to worms, we have lately seen evidence of an alliance between spammers and worm writers. This union has resulted in worms that infect host computers and turn them into spam-delivery agents, giving a profit motive to something that had so far been an essentially recreational, albeit perverse, exercise.

Such developments, combined with the advent of large-scale attacks such as Sapphire, have led some to argue, in the words of Richard Stiennon, a network security analyst at Gartner Inc., in Stamford, Conn., that "intrusion detection is dead." We don't subscribe to this view, but it is true that many problems remain and that new techniques are needed to deal with today's problems.

We have taken a different tack, choosing precision over generality. Our Billy Goat is a specialized worm-detection system that runs on a dedicated machine connected to the network and detects worm-infected machines anywhere in it.

We designed Billy Goat to take advantage of the way worms propagate. The project began in 2001 as a "follow your nose" exercise in intrusion detection. We noticed that computers connected to the network often received automated requests from other computers—such as the service requests that were at the heart of the Sapphire episode—that did not correspond to their normal function. Investigating these requests, we found that worms caused a large fraction of them.

The reason is that worms typically find new computers to target by searching through Internet addresses at random. The strategy is effective because the ever-increasing number of infected machines in an attack can soon be generating hundreds of millions of addresses to try; it matters little if most of them are useless. Indeed, an unprotected machine linked by broadband to the Internet will often be compromised within minutes.

To follow up on our investigation, we assigned to our machine a large number of unused and unadvertised addresses. Because no one but us knew of the existence of these addresses, we could assume that requests flowing to the machine would almost surely be illegitimate. Thus was born the first Billy Goat.

Billy Goat functions by responding to requests sent to these unused addresses, presenting what from a worm's-eye view looks like a network full of machines and services. In other words, it creates a virtual environment for the worms.

Such virtualization, by providing feigned services as well as recording connection attempts, helps Billy Goat trick worms into revealing their identity. This subterfuge allows the system to reliably identify worm-infected machines in a network. For example, a Sapphire-infected machine that connects to Billy Goat will be presented with a Microsoft SQL Server that looks real to the worm but which in fact is a fake response provided by the system. When the worm tries to infect Billy Goat, its identity and address get recorded and immediately reported to the network administrator.

Billy Goat presents many such fake services, in an attempt to lure as many different worms as possible. Furthermore, the system can expand its deceptive capabilities with little effort—an important feature, given the continual stream of security updates needed to keep up with newly discovered vulnerabilities.

With the system basing all its feigned services on an infrastructure for virtualization, a single Billy Goat machine can appear as many addresses on the network. At the same time, the virtualization infrastructure allows the use of standard programming tools and interfaces to create new feigned services for Billy Goat. The mountains of data that these virtual services create are stored and cataloged—that is, logged—in a standard relational database.

To ensure that Billy Goat keeps working under heavy worm attack, when the network may slow down dramatically, we have given it a distributed architecture. This means that multiple Billy Goats can coexist in a network. Each one of them can analyze local events and inform administrators of local infections. At the same time, the data from all of them are assembled to provide a global view of the network.

Billy Goat is extremely effective at detecting worm-infected machines in a network. It is currently used in several large corporate intranets, and it is normally able to detect infected machines within seconds of their becoming infected. Furthermore, not only is it able to tell us that there is a worm in the network, it can even tell us the addresses of the infected machines. This makes it considerably easier to remedy the problem.

It is worth contrasting Billy Goat with another type of security tool called the honey pot, one that's been used by the security community for many years. While there is no single universally accepted definition of honey pot, the various conceptions share the presence of the eponymous honey: something to lure attackers, in the hope of gaining insight into their methods.

A honey pot must therefore advertise its existence and allow itself to be partially corrupted to give the attacker the illusion of success. There is much debate within the security community over the use, potential dangers, and legal implications of these tactics. Billy Goat, by contrast, offers no real services, invites no corruption, and does not attempt to trick human beings.

We are currently pursuing two avenues of development based on Billy Goat. The first is creating an active intrusion response system that automatically isolates infected machines, preventing further worm propagation. The second is using Billy Goat as the basis for an early warning system that allows rapid detection of new and emerging threats.

What comes next? The Holy Grail of intrusion detection—a system that never misses an attack or sounds a false alarm—remains distant, and there are serious doubts that it can ever be reached. In any case, we are convinced that great progress can be achieved by doing a few things well rather than trying to do everything.

About the Author

JAMES RIORDAN, ANDREAS WESPI, and DIEGO ZAMBONI are research scientists at the IBM Zurich Research Laboratory, specializing in intrusion detection and other aspects of network security.

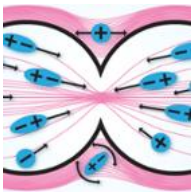
To Probe Further

A historically informed treatment of network security can be found in *Intrusion Detection*, by Rebecca Gurley Bace (Macmillan Technical Publishing, 2000)

For a technical analysis of the Slammer/Sapphire Worm attack, see "Inside the Slammer Worm," by D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, in *IEEE Security and Privacy*, vol. 1 No. 4, July-August 2003, pp. 33;39, August 2003 or visit http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html (http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html)

For Dorothy E. Denning's article, "An Intrusion-Detection Model," see *IEEE Transactions on Software Engineering*, February 1987, Vol. SE-13, no. 2, pp. 222-32.

Recommended For You

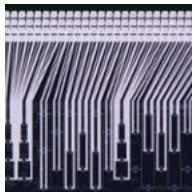


Electric Fields Fight Deadly Brain Tumors

(/biomedical/devices/electric-fields-fight-deadly-brain-tumors)

The Tricks to Launching 100 Satellites on One Rocket

(/tech-talk/aerospace/satellites/the-tricks-to-launching-many-satellites-on-one-rocket)



HPE's New Chip Marks a Milestone in Optical Computing

(/semiconductors/processors/hpes-new-chip-marks-a-milestone-in-optical-computing)

2017: The Year of Self-Driving Cars and Trucks

(/transportation/advanced-cars/2017-the-year-of-selfdriving-cars-and-trucks)

The Grand Ethiopian Renaissance Dam Gets Set to Open

(/energy/policy/the-grand-ethiopian-renaissance-dam-gets-set-to-open)

2017 Is the Break Year Tesla's Gig

(/transport
cars/2017-i
makeorbrea
for-teslas-
gigafactory)