software development The lifecycle starts here.

Intrusion Detection

April 2000

Blocking the Breach: The latest salvo in a decades-old battle for secure systems.

by Alexandra Weber Morales

The assessment is harsh but familiar: "There is little question that contemporary, commercially available systems do not provide an adequate defense against malicious threat. Most of these systems are known to have serious design and implementation flaws that can be exploited by individuals with programming access to the system."

Though the words from this computer security study commissioned by the U.S. Air Force could have been written in 2000, a subsequent sentence puts them in historical context: "While there are growing requirements for interconnecting computer systems into networks, the dimensions of the security problem are unknown." Indeed, in 1972, when James P. Anderson wrote the report–and even a decade later–the military's security specialists could not have predicted the degree to which heterogenous networks would ramble across the globe.

"I don't believe that anyone in defense circles, which are at the root of a lot of what we know about security, could ever have foreseen the impact that the World Wide Web has had. And some folks in defense were blindsided by the whole notion of distributed systems," says Rebecca Bace, a security consultant and researcher in Scotts Valley, Calif. Bace has been in the field since 1989, when she was tapped to lead the National Security Agency's Computer Misuse and Anomaly Detection research program.

Today, numbers describing the expanding dimensions and costs of the computer security problem are readily available. According to the Computer Security Institute, in its annual report with the FBI on what might aptly be termed "e-crime," rates of attack have risen steadily for the past three years. Of the 521 security practitioners surveyed in 1999, 30 percent reported intrusions by outsiders and 55 percent reported unauthorized access from insiders. Faced with such statistics, security professionals are looking past firewalls and virus protection to the latest well-hyped offering: intrusion detection systems.

Playing in Traffic

The most common methods of breaching security are social engineering (fraudulent or deceptive requests for confidential information), viruses or worms, Trojan horses, hostile web content (applets, Active X, JavaScripts), physical access to machines and remote attacks across the Internet. Intrusion detection systems (IDS) sniff out remote attacks in progress as well as the presence of Trojan horses and worms.

Once a breach has been detected (depending on the product), an IDS can generate real-time alerts, reconfigure routers or firewalls on the fly and convert log entries into reports displaying security trends graphically. In addition to host- and network-based IDS, vulnerability and file integrity assessment are two other key approaches.

A host-based IDS processes audit data derived from user, application and system activity (with electronic commerce over the World Wide Web, the interaction between users and applications requires increased monitoring). A network-based IDS, on the other hand, not only watches signs such as device state transitions tracked in network management logs, but it also employs sniffers to read source and destination information contained in packet headers, scanning the data for attack signatures.

These signatures are unique sequences of system commands, such as a move, then a rename, followed by an invocation of the file. Or, at a more complex level, a pattern of bytes on a certain port number can indicate that a Trojan horse such as Back Orifice, a malicious program released in 1998 that allows remote control of a PC running Windows, is functioning. The fidelity of matching attack signatures depends, however, on how comprehensive the database is. Indeed, the prospect of constant updates makes this method of intrusion detection attractive to vendors who watched the growth of the antivirus market.

"Eight years ago, antivirus software was all shareware. Intrusion detection is still in its infancy, but in the next few years we expect it to become as ubiquitous as virus protection is now," says Rob Graham, chief technical officer for Network ICE Inc., a San Mateo, Calif.-based company targeting not only the corporate borderless network market but also, with its $39.95 combination IDS and personal firewall offering, the end-user cable modem and digital subscription line market.

Questions on when, where and how to use IDS abound: Should the IDS sit in front of or behind the firewall? Will a suite of network- and host-based products perform better than an assemblage of best-of-breeds? What is an acceptable incidence of false positives? Will there be a noticeable drain on performance? What is the product’s scalability? How often should the attack signature database be updated? Is it ethical–or efficient–for vendors to use hackers to try to crack their IDS?

"This is where a lot of IDS vendors are right now," laughs Bace. "If your value-add is that you keep track of all of the hacker threat out there, in that case, yeah, you can afford to hire them. These are folks who are totally enchanted with the notion of finding new trouble. Internet Security Systems and Axent and Network Associates have parlayed that into a wonderful business case–and I’m not slamming that; it’s just that it isn’t sufficient. Hackers aren’t going to enable the consistency of process that’s key to hardening the trustworthiness of your software."

Matters of State

In addition to signature databases, some IDSs come with scripting languages that allow users to define additional events to track. Many also store state information for a given connection, which lets them trigger on conversations as well as single packets. But statefulness isn’t everything.

"Some folks believe intrusion detection is simply a linear pattern-match exercise–and it isn't," says Bace. "When you start dealing with heavily distributed systems, you have Poisson effects associated with the arrival time of messages. Poisson is a traffic analysis concept that says that when events need to be consolidated at a particular point in space, you'll have some randomization of the arrival times of those messages."

Atomic signatures are simple enough to deal with, but when multiple indicators from even a single stream must be analyzed, the detection process becomes much more complicated. The difference between trouble and benign activity often comes down to the ordering of the event stream. In race conditions, for example, a check and an access take place in sequence, with a write in the intervening slice rather than after the access, where it should normally occur.

According to Prof. Eugene Spafford, director of Purdue University's Center for Education and Research in Information Assurance and Security (CERIAS), analyzing event-stream ordering requires recording not only when things start but when they end.

"When opening a file on some Unix systems, we need to record not only when the call to open it was made, but also when the file was actually opened and whether the name was mapped to the underlying system–because the two change in some attacks," Spafford says. "Ordering requires a finer-grained time stamp than what's used. Currently, the system clock doesn't have the right resolution. It's a very minor change to add a sequence field; for every system audit event you just add one to it. That's not done, but it can make a huge difference."

In the end, without measures preventing the surveillance of sensitive settings in the first place, says Bace, a stateful IDS may not make much difference. And cache space and performance limitations place an additional burden on issues of state.

"Some of the early failures that we saw were in statistics rather than attack signatures. You'd have these immense matrices characterizing activity not only for every user on the system but also for every file object, every process. These would grow incredibly quickly until you'd run out of memory," explains Bace.

Unforgiving Hosts

Even today, Spafford dismisses the feasibility of logging all packets, a feature that NetworkICE's Rob Graham boasts allows his product to prevent intruders from avoiding detection by using long, slow scanning.

"It's not viable," Spafford says. "If you've got a network of 15,000 machines (which is not a lot for some organizations), and you're doing e-commerce and office applications, what kind of disk farms do you need to have to save every packet over any length of time? It may work for a very narrow environment, but not in general."

"Many systems really miss the boat," continues Spafford, whose CERIAS ranks as the nation's preeminent security think tank. "They don't catch insider misuse, they don't work on ATM networks, they don't work on hubs that don't broadcast, they don't work with virtual private networks, encrypted traffic or enterprises where you have more than one route into the network for load sharing or fault tolerance. Those kinds of systems have a limited lifetime with the continuing evolution of networks."

For these reasons, many experts prefer focusing on host- and application-level security rather than debating whether network sniffers offer sufficient statefulness.

"You can do a lot of cool things in host-based intrusion detection that you can't do from the network side," Bace says. "When I'm inside the host watching an operating system audit trail, I can tell when people transition identities and see how much resource they've consumed. From a network point of view, you can't determine what the ultimate state is on the endpoint. You can see a hack go across, but you can't lay your hands on evidence of whether it worked or not, and exactly what the reach was. It could easily be blocked at the host level, and you'd never know."

Fail-Safe Basics

The experts agree that IDS is still immature–and it's detecting the failure of software engineers to learn robust programming lessons first taught in the 1970s.

That's why Matt Bishop, associate professor at the University of California, Davis department of computer science, is assembling a compendium of seminal papers in computer security (on CD-ROM and at http://seclab.cs.ucdavis.edu/projects/history/seminal.html). According to Bishop, developers too often overlook the early work and waste time rediscovering problems and solutions.

"Vulnerabilities have been around since the first software. There were two seminal studies in the 1970s about this. One was done by Bob Abbott and a group of people at Livermore Laboratories. They surveyed about 10 systems and built a classification scheme for the vulnerabilities they found. The Information Sciences Institute project, under Richard Busbee, published a similar study two years later. Their goal was to automate the detection of vulnerabilities. They didn't succeed, but they developed a very interesting classification of security flaws."

But that work lay dormant in the 1980s, according to Bishop–abandoned in favor of the belief that the technology at the time allowed the construction of secure systems without any vulnerabilities. It soon became evident that this was not true.

"You can't build systems without vulnerabilities," says Bishop, "and even if you could, you'd have problems with the operators and misconfigurations and such. In 1991, the folks at Purdue came out with a taxonomy of security flaws. I wrote one at the same time. Carl Landware, in the late 80s, had come out with one as well. They all boil down to this: Essentially, nothing's changed. The flaws described in the taxonomies were quite different, but you could basically map them on to one another. I'm coming up with a new one now, and again, I'm not seeing any development in the field. Techniques for writing good code and testing for vulnerabilities are not being used. Intrusion detection will help us spot things, but it's too late. We’ve got to build better code."

Many security professionals, though, are tired of hearing the same harangue about quality code at every conference keynote address. Bishop does acknowledge that "we’re beginning to get a handle on how to scan programs for potential race conditions; once we’ve got a list of potential ones, we can go in and eliminate them." But, he notes, a classic attack from the late 1970s–buffer overflow–is still successfully penetrating systems today.

"As an adversary, I have a much easier task than you have as a security developer," Bace explains. "I only have to find one hole; you have to make sure there are none. A wonderful hacker is not someone who understands how to protect the system. The hacker might be able to engineer workarounds for those areas he knows how to attack, but there’s no guarantee that what he’s seen represents the entire universe of problems."

In her book, Intrusion Detection (Macmillan Technical Publishing, 1999), Bace urges developers to reacquaint themselves with a 25-year-old paper by Jerome H. Saltzer and Michael D. Schroeder ("The Protection of Information in Computer Systems," Proceedings of the IEEE, Vol. 63, No. 9, Sept. 1975), in which the authors enumerate eight fail-safe security design principles. They are:

1. Least privilege. Relinquish access when it’s not required.

2. Fail-safe defaults. When the power goes off, the lock should be closed.

3. Economy of mechanism. Keep things as small and simple as possible.

4. Complete mediation. Check every access to every object.

5. Open design. Don’t attempt "security by obscurity," as Bace puts it; assume the adversary can find your hiding places.

6. Separation of principle. Don’t make privilege decisions based only on a single criterion; use the onion-skin model.

7. Least common mechanism. Minimize shared channels.

8. Psychological ac-ceptability. Make security painless, transparent and ubiquitous.

Both Bishop and Spafford concur with Bace that following these principles would go a long way toward eliminating the "penetrate-and-patch" mentality that prevails among software vendors today.

The current approach–trying to keep pace with all the attacks as the number of weaknesses multiplies–is fundamentally backward, Spafford laments: "The next generation of Windows has 60 million lines of code. How can it possibly be designed to be secure?"

Future Fears

Increasingly complex, monolithic operating systems are not, however, the biggest risk on the horizon. While Internet security has been reinvented by the Java virtual machine, which already emulates IDS in its method of code verification by stack inspection, problems with mobile code only look to get worse. With embedded systems attached to heterogeneous networks, intruders no longer stand to simply gain access to information or financial transactions–now they could conceivably control real-time systems or safety-critical devices. In addition, even the rigidly bureaucratic, stiffly secure Department of Defense has found itself in potentially compromising positions with commercial off-the-shelf components that have been built quickly under market pressures and don’t comply with its iron-clad requirements.

"There’s a huge demand for security software in general, but there hasn’t been enough real research done," says Spafford. "The result is products that aim to meet a market demand. Customers want more features rather than strong security, so vendors introduce more complexity, which leads to a greater likelihood of failure. We’ve seen this in firewalls, and we’re going to start to see it in intrusion detection."