COMPUTER SCIENCE

Security Schemes Aspire to No-Fuss System Protection

BALTIMORE—When the most recent security problems surfaced in Netscape, the leading browser for the World Wide Web, the popular press treated them as headline news. But they came as no surprise to the computer-security specialists attending the nation's largest annual meeting of anti-cracker fighters—the 18th National Information Systems Security Conference.* As Padgett Peterson, chief of information security at Martin-Marietta, put it, the wide-open character of the Internet "is a feature, not a bug. It was designed to be that way."

That sets the scale of the challenge these experts face as they try to protect universities, research centers, and industries from intruders who can steal data, corrupt files, or just plain make mischief. Technical fixes like one-time passwords can stop some break-ins, such as those that use dictionary-based programs to search out passwords that are simple names or phrases. Fire walls, devices that filter traffic from the Internet to local computers, can also make intrusions difficult (*Science*, 3 February, p. 608). But although these remedies "can have a big impact, they're often not put into practice," says Sandy Sparks of the Department of Energy's Computer Incident Advisory Capability, a security team based at the Lawrence Livermore National Laboratory. "They make networks harder to use, and people don't like it." Indeed, Steve Bellovin, a widely respected security expert at AT&T Bell Labs, confessed at his Baltimore talk that he had forgotten to print up some of the transparencies the day before, and that his office computer was so secure that he had been unable to retrieve his files with a laptop.

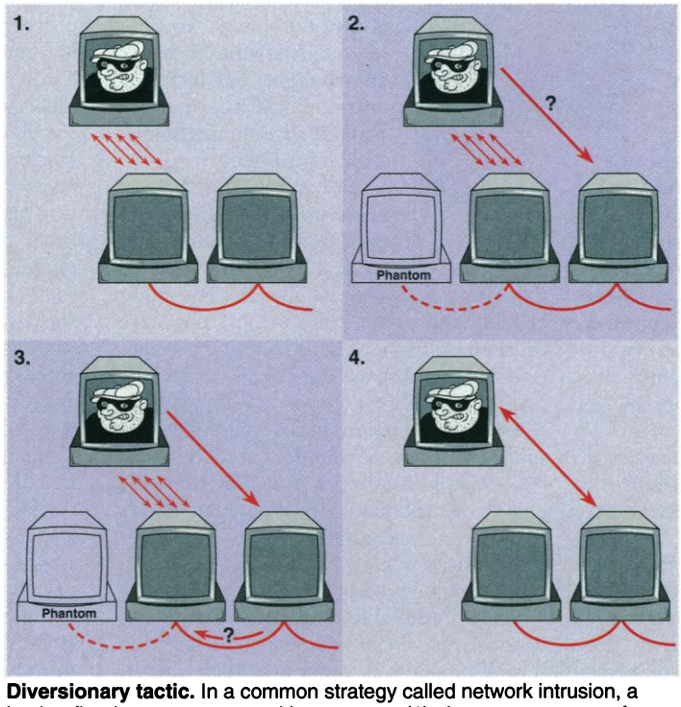
Because of the painful trade-offs between security and usability that come with current protective measures, computer scientists have been investigating new ways to detect and ward off intruders that present fewer obstacles to users and administrators. Two of the most intriguing directions announced at the Baltimore conference were reported by researchers at Purdue University's Computer Operations, Audit, and Security Technology laboratory (COAST). The first was an attempt to harness techniques from artificial intelligence to detect intrusions; the second, a pilot test of what has been called software forensics—an effort to recognize malign programs by identifying the fingerprints of their creators.

Although COAST was created, according to director Eugene H. Spafford, "to develop [security] technologies and pitch them over the wall to industry," the greatest beneficiaries of new security techniques like these may be universities and research centers, which use a wide range of Internet services and must keep their networks open and accessible. Unlike industry, for example, universities and research centers cannot simply shut off risky services like Telnet and FTP, because traveling researchers and outside colleagues need to use them. On top of that, scientists and academics often have little patience with measures such as one-time passwords, which force users to carry around lists of codes, or fire walls, which block off some parts of the network entirely. Even when administrators impose these measures, irate scientists often know enough about computers to bypass them, making them useless. "The scientists are the worst!" one research-center administrator in Baltimore told *Science*. "They not only refuse to be careful; they actually work *against* any security measures we put in."

Computer users are not the only ones to have trouble balancing security and usability. Many administrators already find it difficult to keep networks running, because they are unplanned bricolages of different computers and operating systems; add in the diverse security problems associated with each component and the need to monitor for the anomalous usage that signals an attack, and you have a job that one administrator characterized as "a nightmare, and also impossible." To relieve the burden, some "sysadmins" have installed intrusion-detection programs that stay on the job 24 hours a day. Unfortunately, these programs represent a single point of attack—disable them, and mischief-makers can walk in freely. In addition, the programs themselves must be constantly augmented to take account of newly discovered security problems, so that they grow constantly in size and complexity and impose an ever-larger load on the system.

Autonomous agents. Because of these drawbacks, Spafford and Mark Crosbie, one of his students, have proposed inoculating networks with "autonomous agents"—small, independent programs that, like the components of the human immune system, collectively monitor the system for intruders. "Instead of one type of immune cell with many functions," explains Crosbie, "you have a multitude of them, all specialized, working in concert." In classic artificial-intelligence style, the hope is that the summed efforts of these little, comparatively simple agents will be more effective than a single, large program that has many complex functions. In addition, the independence of each agent from the others means that subverting any one of them will leave the rest of the system capable of guarding against other types of intrusion. "And if you find a new security problem," says Crosbie, "you simply add or remove an agent, instead of reworking the entire program."

In their pilot test of this strategy, Crosbie and Spafford created agents to detect "network intru-



Diversionary tactic. In a common strategy called network intrusion, a hacker floods one computer with messages (1), then requests entry from a second computer while posing as a legitimate part of the network (2). The second computer asks the first to authorize the connection (3). The first computer is too overwhelmed to respond, so the second computer accepts the request, and the hacker gains access to the system (4).

* The 18th National Information Systems Security Conference, held 10–13 October in Baltimore, sponsored by the Computer Systems Laboratory of the National Institute of Standards and Technology and the National Computer Security Center of the Department of Defense.

sions"—a type of attack in which an invader swamps a computer with so many high-priority incoming messages that the system is temporarily unable to fend off the potential break-in (see illustration on p. 1113). Network intrusion first made headlines last January when hacker Kevin Mitnick allegedly used it to break into the home system of Tsutomu Shimamura, a security guru at the San Diego Supercomputing Center. Mitnick's supposed approach was simple: While flooding one of Shimamura's computers, Mitnick requested access to a second one in Shimamura's system, claiming that his own computer was "trusted" by Shimamura's first computer. The second Shimamura computer queried the first, which was too overwhelmed to respond; then, because the second computer's default setting was to trust a machine unless explicitly told not to, it apparently allowed Mitnick inside.

"You can read about [this type of attack] in papers from 15 to 20 years ago," Crosbie says. "It's annoying that people aren't learning from past mistakes." Even today, he says, "the simplest attacks are often the best because people forget to defend against them. That's one thing our project might help prevent."

To develop their agents, Crosbie and Spafford used a technique called "genetic programming," a term coined 3 years ago by Stanford University computer scientist John Koza to describe the random generation of hundreds or thousands of programs, from which the researcher selects a group that comes closest to performing the desired task. After that, Crosbie explains, "you recombine [the most successful programs], sticking branches from one program onto another, with an eye toward improving it. Then you compare the new group of programs with the desired result. And you do this again and again, trying to move toward an optimal program." With this approach, researchers avoid much of the long process of testing and retesting that usually accompanies software development, because many randomly different versions of the program are being tested at once.

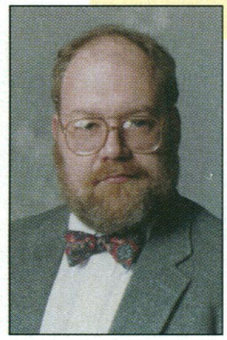
In this case, Crosbie and Spafford "evolved" their agents by simulating network intrusions on a computer with a Hewlett-Packard operating system and iteratively measuring the agents' responses; at the same time, they also exposed the agents to similar but benign phenomena, such as a single researcher making unexpectedly heavy demands on the system. Over many generations, the autonomous agents learned to distinguish between attacks and unusual patterns of use, alerting sysadmins when the former occurred.

After this initial small-scale success, Crosbie has now begun to develop agents for the Solaris operating system, which is used by Sun computers; he also hopes to develop and test a suite of agents for other types of attack. An open question, however, is whether the autonomous agents will impair the performance of their host networks. "I hope to have real-world data [on their impact] in 6 months," Crosbie says.

What's more, the real value of these systems won't become clear until they confront actual intrusions, not the simulated attacks used to develop them, cautions Armand Prieditis of the University of California, Davis. According to Prieditis, who is trying to develop game-playing agents that can entrap intruders, "we don't have good data on what intrusions are really like, because people keep their problems secret. So you have to manufacture thresholds for what anomalous behavior is, and that is always a problem—you end up not knowing if you're able to detect real intruders or can only detect what you wrongly thought might look like an intruder."

"If you were able to identify the authors of programs [used by hackers], you might be ahead of the game."

—Eugene Spafford



Software forensics. In an entirely different approach to computer security, Spafford and Stephen Weeber, a former student, suggested that the creators of malign programs might be identified by the "fingerprints" they leave on their software. The underlying assumption is that most hackers, like Mitnick, are people of modest computer skills who use the automated techniques developed by the small minority of sophisticated hackers. An example is Rootkit, a widely available collection of break-in routines that allows users to gain access to systems and cover their tracks merely by typing the word "make" at the prompt.

"If you were able to identify the authors of programs, or even merely [ascertain] that several programs came from a single source, you might be ahead of the game," Spafford suggests. Indeed, software forensics might someday be used, like traditional criminal forensics, to identify and convict the hacker kingpins who are responsible for most intrusion programs.

Working with a student, Ivan Krsul, Spafford has been trying to exploit what he calls an "intriguing analogy" between the characteristic vocabulary and style that can be used to identify the author of a literary text and characteristic programming techniques and styles, which might reveal the author of a computer program. "Humans work in repeated patterns and tend to reuse the things they've learned well," Krsul says. And computer programmers, for their part, "have particular ways they like to put down lines of code."

Consulting a collection of style rules, programming proverbs, rules for judging software complexity, and other sources, Krsul and Spafford assembled a list of 32 "metrics" that mark individual programming styles, among them the percentage of function names that start with an upper-case letter, the percentage of open curly brackets that end a line of code, and the lines of code per function. Next, they asked 29 students and staffers to anonymously contribute three programs apiece (one contributed four). After testing the metrics, Krsul and Spafford were able to identify the programmers with 73% accuracy. (A more detailed description of both projects can be found at **http://www.cs.purdue.edu/coast/coast.html**.)

The technique was equally able to identify inexperienced students, seasoned programmers, and COAST faculty. Just one programmer was never classified correctly, and the researchers found that he had produced programs that looked so different from one another that the two men suspect cheating. "Either that, says Krsul, "or else he's a complete genius who can program in three completely different styles. In a way, the fact that the program couldn't classify him is a testament to its success."

Krsul adds that the consistency of the other misidentifications suggests that improvement is possible. "There is a pattern there, but the tool we have written is not yet sophisticated enough to detect it," he says. More sophisticated metrics, he believes, could boost the success rate dramatically. Another vehicle for improvement, he says, would be to refine the presently cumbersome method of statistical analysis by using neural networks.

All of these ideas, cautions Sparks of the Department of Energy, do not mean that a total solution to the world's computer security ills is in sight. "In general, the more security methods we develop, the better," she says. "But no one single fix will cure everything, especially if the people using these systems don't pay attention to what they're doing." Krsul agrees. "None of the things we create will help much," he says, "if people keep using their names as passwords."

–Charles C. Mann