[in]security

by Gary McGraw

How Do Real Bad Guys Break Software?

Most of the people attacking our computer systems today are software people. While script kiddies are part of the problem, the real threat comes from those who use hard-core software tools to take apart programs, write malicious code, and create the attacks that exploit our software.

In this column, we'll focus our attention on attack tools that require a copy of the target software. We're not really limiting ourselves by this constraint, however. Today, we have copies of all kinds of targets that can be exploited, from the OS running on our laptops (the binaries are all there) to the ubiquitous Web server software that can be downloaded for free. Real attackers usually get a copy of their target into the lab, where they wield various surgical tools to dissect it.

Attackers have a well-developed toolkit, with components that run the gamut from standard analysis and testing tools to rootkits and payload collections with no other legitimate use. (We'll look at rootkits and payload collections in another column.)

The first and most important category of these tools work against binary executables. Both decompilers and disassemblers allow humans to understand and analyze target binary code. This means that despite the occasional "big security story" about source code finding its way onto the Internet, source code isn't required for software exploit. Though source code makes an attacker's job easier, it's in no way a necessity. Most attackers use disassemblers and decompilers in concert with other tools to test inputs to the target. They then watch what happens and take things apart as necessary.

Debuggers also make excellent tools for understanding how programs work. The most common debuggers used by attackers are the kernel-level variety, which intercepts all the calls a target program makes to and from the OS. These debuggers allow a program to be meticulously observed, stopped, rewound, started, changed, and so on—all in real time.

Getting in between components of a target program is also a useful technique. API interposition tools let an attacker intercept, record, play back, and change messages sent inside a program as it runs. Similarly, dependency analysis tools provide information about what external functions, utilities, and libraries a program needs to work.

Coverage tools, often used by quality assurance people to look at testing effectiveness, double as attack tools. Knowing what parts of a target program are executed under what conditions can help an attacker hone in on a vulnerability. Other dynamic analysis tools that can be turned to evil ends include control flow and data flow analyzers. These tools all improve understanding about how a program works, what kinds of data it processes, and how the moving parts work together.

Because attackers often want to work at a distance, vulnerabilities that are discovered using such tools are usually scripted into attacks that can be used remotely. Fault injection engines and input generators (called "fuzzers") are very useful when probing the target's input space. Once a vulnerability deep in the system is discovered, the next trick is getting to it through input. Sometimes problems can be discovered accidentally by sending in random input, but tailoring input according to knowledge of the software's guts is always better.

> "It seems that the bad guys have better capitalized on software knowledge than the good guys."

As mentioned earlier, most of the people attacking our computer systems today are software people. Unfortunately, most of the people defending them aren't. In terms of the operator vs. builder problem raised previously in this column, it seems that the bad guys have better capitalized on software knowledge than the good guys.

But it doesn't have to be this way. Every single one of these tools can be used by the good guys just as effectively as they're now used by the bad guys. By learning how these tools work and what to make of their output, security professionals can better understand and counter security attacks.

Gary McGraw is CTO of Cigital, a software quality management consulting provider. He is co-author of *Exploiting Software* (Addison-Wesley, 2004), *Building Secure Software* (Addison-Wesley, 2001), and *Java Security* (Wiley, 1996). Reach him at gem@cigital.com.