Software Realities

Editor: James Bach, Reliable Software Technologies, 21515 Ridgetop Circle, Suite 250, Sterling, VA, 20166; j.bach@computer.org

Software Assurance for Security

Gary McGraw
Reliable Software Technologies

Computer security is taking on new importance as electronic commerce metamorphoses from hype to reality. Large and small businesses alike are reinventing themselves as e-commerce players. The implications for computer security practice are immense. When bits count as money, protecting bits becomes as important as any other aspect of running a successful business.

One essential element shared by every modern information system is the software that determines how the system behaves. Today's software problems lead to spectacular real world failures of many different kinds, including security problems, reliability problems, and safety problems. It is probably only a matter of time before software causes the demise of a large company.

What can we do to combat software bugs lying at the root of these problems, especially in light of the rush to embrace e-commerce and the intense pressure of Internet time? How can we avoid treating security as an add-on feature, when, like dependability, security is really a property of a complete system? This column discusses an approach to security analysis that we have applied successfully over the last several years at Reliable Software Technologies. Our approach is no magic bullet, but it offers a reasoned methodology that has proven to be useful in the trenches.

A VIEW FROM 50,000 FEET

Our methodology, like many useful things, is a mix of art and engineering. The idea is straightforward: Design a system with security in mind, analyze the system in light of known and anticipated risks, rank the risks according to their severity, test to the risks, and cycle broken systems back through the design process.

The process outlined above has one essential underlying goal: avoiding the unfortunately pervasive penetrate-and-patch approach to computer security—that is, avoiding the problem of desperately trying to come up with a fix to a problem that is being actively exploited by attackers. In simple economic terms, finding and removing bugs in a software system before its release is orders of magnitude cheaper and more effective that trying to fix systems after release. The many problems inherent in the penetrate-and-patch approach have been discussed elsewhere, but the main concerns are:

- Patches are rarely applied by overworked system administrators who would rather not tweak their functioning systems;
- patches are rushed out under immense pressure from the market and often introduce new problems of their own; and
- patches are superficial solutions that do not get at the heart of software problems.

Designing a system for security, and testing the system extensively before release, presents a much better alternative.

RISK ASSESSMENT

There is no such thing as 100 percent security. In fact, there is a fundamental tension inherent in today's technology between functionality (an essential property of any working system) and security (also essential in many cases). A common joke goes that the most secure computer in the world is one that has its disk wiped, is turned off, and is buried in a ten foot hole filled with concrete. Of course, a machine that secure also turns out to be useless. In the end, the

security question boils down to how much risk a given enterprise is willing to take on in order to solve the problem at hand effectively. Security is really a question of risk management.

The key to an effective risk assessment is expert knowledge of security. Being able to recognize situations where common attacks can be applied is half the battle. The first step in any analysis is recognizing the risks. This step is most effectively applied to a system's specification.

Once risks have been identified, the next step is ranking the risks in order of severity. Any such ranking is a context-sensitive undertaking that depends on the needs and goals of the system at hand. Some risks may not be worth mitigating, depending, for example, on how expensive carrying out a successful attack might be. Ranking risks is essential to allocating testing and analysis resources further down the line. Since resource allocation is a business problem, making good business decisions regarding such allocation requires sound data.

Given a ranked set of potential risks in a system, testing for security is possible. Testing requires a live system and is an empirical activity requiring close observation of the system under test. Security tests often do not result in clear-cut results like obvious system penetrations, though sometimes they do. More often a system will behave in a strange or curious fashion that tips off an analyst that something interesting is afoot. These sorts of hunches can be further explored.

SOUND SOFTWARE ENGINEERING

One premise of our methodology is the use of sound software engineering practices. Process is a good thing, in moderation. Any system that is designed according to well-understood requirements will be better than a system thrown together arbitrarily. An apt example of a security-related requirement is a requirement that states that some data must be protected against eavesdropping since it is particularly sensitive information.

From a set of requirements, a system specification can be created. The importance of solid system specification cannot be overemphasized. After all, without a specification, a system cannot be wrong, it can only be surprising! And when it comes to running a business, security surprises are not something we want.

A solid specification draws a coherent big-picture view of what the system does and why the system does it. Specifications should be as formal as possible, without becoming overly arcane. Formality is extremely powerful, but it too is no silver bullet. Remember that the essential raison d'etre for a specification is understanding. The clearer and easier to understand a specification is, the better the resulting system will be.

THE IMPORTANCE OF EXTERNAL ANALYSIS

Nobody designs or develops systems poorly on purpose. Developers are a proud lot, and for the most part they work hard to create solid working systems. This is precisely why a security risk analysis team should not include anyone from the design and development team. One essential way in which security testing differs from standard testing is in the importance of preserving a completely independent view of the system, divorced from design influences. In general testing, one person can play dual roles; a design team testing expert to improve testability early on, and an independent tester later in the process. In security testing there is a much greater risk of tunnel vision.

Putting together an external team is important for two main reasons:

- To avoid tunnel vision. Designers and developers are often too close to their systems and are skeptical that their system may have flaws.
- To validate design document integrity. The requirements and specifications that the designers and developers use should be clear enough that an external team can completely understand the system.

Another reason warranting the use of external teams is the expertise issue. Being an excellent programmer and understanding security problems are not the same thing.

The good news is that an external team need not be made up of high-priced external experts. Often it is good enough to have a team from your own organization made up of security experts who were not involved in design decisions. The bad news is that security expertise seems to be a rare commodity these days. Determining whether or not to seek help outside of your organization will depend on what the system you are designing is meant to do and what happens if it is broken by attackers. If you are betting your business on a piece of code, it had better not fail unexpectedly.

An experienced team of external analysts considers myriad scenarios during the course of an analysis. Examples of scenarios include decompilation risks in mobile code systems, eavesdropping attacks, playback attacks, and denial of service attacks. Testing is most effective when it is directed instead of random. The upshot is that scenarios can lead directly to very relevant security tests.

SECURITY GUIDELINES

Security is risk management. The risks to be managed take on different levels of urgency and importance in different

situations. For example, denial of service may not be of major concern for a client machine, but denial of service on a commercial Web server could be disastrous. Given the context-sensitive nature of risks, how can we compare and contrast different systems in terms of security?

Unfortunately, there is no golden metric. But we have found in practice that the use of a standardized set of security analysis guidelines is very useful. Our most successful client makes excellent use of security guidelines in their risk management and security group.

The most important feature of any set of guidelines is that they create a framework for consistency of analysis. Such a framework allows any number of systems to be compared and contrasted in interesting ways.

Guidelines consist of both an explanation of how to do a security analysis in general, and what kinds of risks to consider. No such list can be absolute or complete, but common criteria for analysis—such as the Department of Defense's Trusted Computing Systems Evaluation Criteria, commonly called the Orange Book—can be of help.

A COMMON MISTAKE

Much of today's software is developed incredibly quickly under immense market pressure. Internet time now rivals dog years in duration, approaching a 7:1 ratio with regular time. Often the first thing to go under pressure from the market is software quality (of any sort). Security is an afterthought at best, and is often forgotten.

Bolting security onto an existing system is simply a bad idea. Security is not a simple feature you can add to a system at any time. Security is like fault tolerance, a system-wide emergent property that requires much advance planning and careful design.

We have come across many real-world systems (designed for use over protected proprietary networks) that were being reworked for use over the Internet. In every one of these cases, Internet-specific risks caused the systems to lose all their security properties. Some people refer to this problem as an environment problem, where a system that is secure enough in one environment is completely insecure when placed in another. As the world becomes more interconnected via the Internet, the environment most machines find themselves in is at times less than friendly.

It is always better to design for security from scratch than to try to add security to an existing design. Reuse is an admirable goal, but the environment in which a system will be used is so integral to security that any change of environment is likely to cause all sorts of trouble—so much trouble that well-tested and well-understood things fall to pieces.

SECURITY TESTING VERSUS FUNCTIONAL TESTING

Functional testing dynamically probes a system to determine whether the system does what it is supposed to do under normal circumstances. Security testing is different. Security testing probes a system in ways that an attacker might probe it, looking for weaknesses to exploit. In this sense, advanced testing methodologies like software fault injection can be used to probe security properties of systems. An active attack is an anomalous circumstance that not many designers consider.

Security testing is most effective when it is directed by system risks that are unearthed during a risk analysis. This implies that security testing is a fundamentally creative form of testing that is only as strong as the risk analysis it is based on. Security testing is by its nature bounded by identified risks (and the security expertise of the tester).

Code coverage has been shown to be a good metric for understanding how good a particular set of tests is at uncovering faults. It is always a good idea to use code coverage as a metric for measuring the effectiveness of functional testing. In terms of security testing, code coverage plays an even more critical role. Simply put, if there are areas of a program that have never been exercised during testing (either functional or security), these areas should be immediately suspect in terms of security. One obvious risk is that unexercised code will include trojan-horse functionality whereby seemingly innocuous code carries out an attack. Less obvious (but more pervasive) is the risk that unexercised code has serious bugs that can be leveraged into a successful attack.

Dynamic security testing can help ensure that such risks don't come back to bite you. Static analysis is useful as well. Many of today's security problems are echoes of well-understood old problems that can come around again. The fact that 80 percent of 1998's CERT alerts involved buffer overflow problems emphasizes the point. There is no reason that any code today should be susceptible to buffer overflow problems, yet they remain the biggest source-code security risk today.

It is possible to scan security-critical source code for known problems, fixing any problems encountered. Current research is exploring the utility of static source-code scanning. The key to any such approach is a deep knowledge of potential problems.

There are many areas today in which software must behave itself. Good software assurance practices can help ensure that software behaves properly. Safety-critical and high assurance systems have always taken great pains to analyze and track software behavior. With software finding its way into every aspect of our lives, the importance of software

assurance will only grow. We can avoid the band-aid-like penetrate-and-patch approach to security only by considering security as a crucial system property and not as a simple add-on feature.

Computer security is becoming more important because the world is becoming highly interconnected and the network is being used to carry out critical transactions. The environment that machines must survive in has changed radically. Deciding to connect a LAN to the Internet is a security-critical decision. The root of most security problems is software that fails in unexpected ways. Though software assurance has much maturing to do, it has much to offer to those practitioners interested in striking at the heart of security problems. ❖

Gary McGraw, Ph.D. *is Vice President of Reliable Software Technologies and co-author of* Securing Java *(1999) and* Software Fault Injection *(1998). Contact him at gem@rstcorp.com.*