by Gary McGraw

How Does Security Fit With Engineering?

As interest in software security increases, savvy security organizations are determining how to integrate security into the way they build software. However, the reality of software development in many organizations is complex. Sometimes development is part of the traditional IT organization, but more often software development teams report to individual business units. Sometimes a standard Software Development Lifecycle (SDLC) is used, but more often each development team has its own way of building software. In almost every case, a particular software process is defended with religious zeal. All these factors make integrating security into software development an interesting challenge.

Fortunately, we can dispense with the SDLC religious war by focusing on the artifacts created during any given SDLC. Every SDLC is guaranteed to create at least one artifact: code. Other possible artifacts include requirements and use cases, design documents, architecture documents, test plans, tests,

There's no substitute for working software security as deeply into the development process as possible.

and test results. Given a set of standard artifacts, we can carry out a number of inline "microprocesses"—that is, touch points or best practices—regardless of the core SDLC. That way, developers can keep building software the way they like and still meet security goals.

SECURITY TOUCH POINTS

Security should begin at the requirements level. Security requirements must cover both overt functional security (say, the use of applied cryptography) and emergent characteristics. One way to cover the emergent security space is to build abuse cases. Similar to use cases, abuse cases describe a system's behavior under attack, providing explicit coverage of what should be protected, from whom, and for how long.

At the architecture level, a system must be coherent and present a unified security architecture that takes into account security principles (such as the principle of least privilege). Designers, architects, and analysts must document assumptions and identify possible attacks. Architectural risk analysis, in which analysts uncover and rank risks, is necessary before mitigation can begin. External analysis outside the design team is often also required.

At the code level, the focus should be on implementation flaws, especially those discovered by static analysis tools such as Fortify Software's Source Code Analysis.

Security testing must encompass two strategies: testing security functionality with standard functional testing techniques, and testing risk-based security based on attack patterns and threat models.

Penetration testing is also useful, especially when it's driven by architectural risk analysis. Note that black-box penetration testing that doesn't take the software architecture into account won't uncover anything interesting about software risk. Software that falls prey to canned black-box testing is bad, and merely passing a cursory penetration test reveals little about security readiness.

Because attacks will happen, monitoring software behavior is an excellent defensive technique for operations people to adopt. Knowledge gained by understanding attacks and exploits should be cycled back into the development organization.

Risks crop up during all stages of the SDLC, so a constant risk analysis thread with recurring risk tracking and monitoring activities is highly recommended.

Practitioners are increasingly adopting and evolving a set of best practices to address software security. Most approaches encompass analysis and auditing of software artifacts, security engineering, and training for developers, testers, and architects. There's no substitute for working software security as deeply into the development process as possible and taking advantage of the engineering lessons software practitioners have learned over the years. Ultimately, the most prudent approach is a process-agnostic one that uses software artifacts as touch points for designing security into the architecture.

Gary McGraw is CTO of Cigital, a software quality management consultancy. 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.