cigital

Software Security

Gary McGraw, Ph.D.

Cigital, Inc.
21351 Ridgetop Circle
Suite 400
Dulles, Virginia
20166
info@cigital.com

www.cigital.com

Software Security

Software security is the idea of engineering software so that it continues to function correctly under malicious attack. Most technologists acknowledge this undertaking's importance, but they need some help in understanding how to tackle it. This paper aims to provide that help by exploring software security best practices.

The software security field is a relatively new one. The first books and academic classes on the topic appeared in 2001, demonstrating how recently developers, architects and computer scientists have started systematically studying how to build secure software. The field's recent appearance is one reason why best practices are neither widely adopted nor obvious.

A central and critical aspect of the computer security problem is a software problem. Software defects with security ramifications – including implementation bugs such as buffer overflows and design flaws such as inconsistent error handling – promise to be with us for years. All too often, malicious intruders can hack into systems by exploiting software defects.[1] Internet-enabled software applications present the most common security risk encountered today, with software's ever-expanding complexity and extensibility adding further fuel to the fire. By any measure, security holes in software are common and the problem is growing: CERT Coordination Center identified 4,129 reported vulnerabilities in 2003 (a 70 percent increase over 2002, and an almost fourfold increase since 2001).[2],[3]

Software security best practices leverage good software engineering practice and involve thinking about security early in the software lifecycle, knowing and understanding common threats (including language-based flaws and pitfalls), designing for security, and subjecting all software artifacts to thorough objective risk analyses and testing. Let's look at how software security fits into the overall concept of operational security and examine some best practices for building security in.

…versus application security

Application security means many different things to many different people. In most cases, it has come to mean the protection of software *after it's already built*. Although the notion of protecting software is an important one, it's just plain easier to protect something that is defect-free than something riddled with vulnerabilities.

Pondering the question, "What is the most effective way to protect software?" can help untangle software security and application security. On one hand, software security is about building secure software: designing software to be secure, making sure that software is secure, and educating software developers, architects and users about how to build secure things. On the other hand, application security is about protecting software and the systems that software runs in a post facto way, after development is complete. Issues critical to this subfield include sandboxing code (as the Java virtual machine does), protecting against malicious code, obfuscating code, locking down executables, monitoring programs as they run (especially their input), enforcing the software use policy with technology, and dealing with extensible systems.

Application security follows naturally from a network-centric approach to security, by embracing standard approaches such as penetrate and patch[4] and input filtering (trying to block malicious input) and by providing value in a reactive way. Put succinctly, application security is based primarily on finding and fixing known security problems after

[1] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley, 2004

[2] J. Viega and G. McGraw*, Building Secure Software*, Addison- Wesley, 2001; www.builingsecuresoftware.com

[3] G. McGraw, "From the Ground Up: The DIMACS Software Security Workshop," *IEEE Security & Privacy*, vol.1, no. 2, 2003, pp.59-66.

[4] G. McGraw, "Testing for Security During Development: Why We Should Scrap Penetrate-and –Patch," *IEEE Aerospace and Electronic Systems*, vol. 13, no. 4, 1998, pp.13-15.

they've been exploited in fielded systems. Software security – the process of designing, building and testing software for security – identifies and expunges problems in the software itself. In this way, software security practitioners attempt to build software that can withstand stack proactively. Let me give you a specific example: although there is some real value in stopping buffer overflow attacks by observing HTTP traffic as it arrives over port 80, a superior approach is to fix the broken code and avoid the buffer overflow completely.

…as practiced by operations people

One reason that application security technologies such as firewalls have evolved the way they have is because operations people dreamed them up. In most corporations and large organizations, security is the domain of the infrastructure people who set up and maintain firewalls, intrusion detection systems and antivirus engines (all of which are reactive technologies).

However, these people are operators, not builders. Given the fact that they don't build the software they have to operate, it is no surprise that their approach is to move standard security techniques "down" to the desktop and application levels. The gist of the idea is to protect vulnerable things (in this case, software) from attack, but the problem is that vulnerabilities in the software let malicious hackers skirt standard security technologies with impunity. If this were not the case, then the security vulnerability problem would not be expanding the way that it is. Clearly, this emphasizes the need to get builders to do a better job on the software in the first place.

Protecting a network full of evolving software is difficult (even if the software is not patched every five minutes). If software were in some sense self-protecting (by being designed defensively and more properly tested from a security perspective) or at least less riddled with vulnerabilities, running a secure network could become easier and more cost effective.

In the short run, we clearly – desperately – must make progress on both fronts. But in the long run, we must figure out ways to build easier-to-defend code. Software security is about helping builders do a better job so that operators end up with an easier job.

…in the software development lifecycle

On the road to making such a fundamental change, we must first agree that software security is not security software. This is a subtle point often lost on the development people who tend to focus on functionality. Obviously, there are security functions in the world and most modern software includes security features, but adding features such as SSL (for cryptographically protecting communications) does not present a complete solution to the security problem. Software security is a system-wide issue that takes into account both security mechanisms (such as access control) and design for security (such as robust design that makes software attacks difficult). Sometimes these overlap, but often they don't.

Put another way, security is an emergent property of a software system. A security problem is more likely to arise because of a problem in a standard-issue part of the system (say, the interface to the database module) than in some given security feature. This is an important reason why software security must be part of a full-lifecycle approach. Just as you can't test quality into a piece of software, you can't spray paint security features onto a design and expect it to become secure. There is no such thing as a magic crypto fairy dust: we need to focus on software security from the ground up.

As practitioners become aware of software security's importance, they are increasingly adopting and evolving a set of best practices top address the problem. Microsoft has carried out a noteworthy effort under the rubric of its Trustworthy Computing Initiative.[5],[6] Most approaches in practice today encompass training for developers, testers

[5] L. Walsh, "Trustworthy Yet?" *Information Security Magazine*, Feb. 2003; infosecuritymag.techtarget.com/2003/feb/cover.shtml.

[6] M. Howard and S. Lipner, "Inside the Windows Security Push," *IEEE Security & Privacy*, vol. 1. no. 1, 2003, pp. 57-61.

and architects, analysis and auditing of software artifacts, and security engineering. In the fight for better software, treating the disease itself (poorly designed and implemented software) is better than taking an aspirin to stop the symptoms. 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.

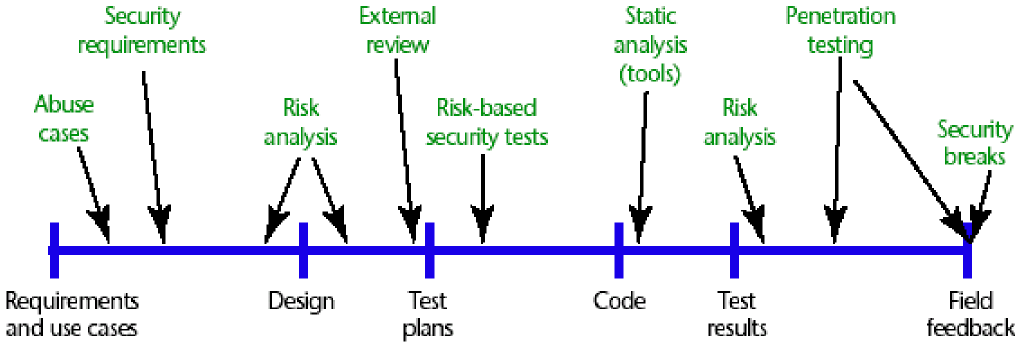

Figure 1. Software security best practices applied to various software artifacts. Although the artifacts are laid out according to a traditional waterfall model in this picture, most organizations follow an iterative approach today, which means that best practices will be cycled through more than once as the software evolves.

Figure 1 specifies one set of best practices and shows how software practitioners can apply them to the various software artifacts produced during software development.

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

At the design and 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 clearly document assumptions and identify possible attacks. At both the specifications-based architecture stage and at the class-hierarchy design stage*, risk analysis* is a necessity – security analysts should uncover and rank risks so that mitigation can begin. Disregarding risk analysis at this level will lead to costly problems down the road. *External review* (outside the design team) is often necessary.

At the code level, we should focus in implementation flaws, especially those that static analysis tools – tools that scan sources code for common vulnerabilities – can discover. Several vendors now address this space, and tools should see market-driven improvement and rapid maturity later this year. As stated earlier, code review is a necessary, but not sufficient, practice for achieving secure software. Security bugs (especially C and C++) can be deadly, but architectural flaws are just as big a problem.

Security testing must encompass two strategies: testing security functionality with standard functional testing techniques, and *risk- based security testing* based on attack patterns and threat models. A good *security test plan* (with traceability back to requirements) uses both strategies. Security problems aren't always apparent, even when we probe a system directly, so standard-issue quality assurance is unlikely to uncover all the pressing security issues.

Penetration testing is also useful, especially if an architectural risk analysis is specifically driving the tests. The advantage of penetration testing is that it gives a good understanding of fielded software architecture in its real environment. However, any black-box penetration testing that doesn't take the software architecture into account probably won't uncover anything deeply interesting about software risk. Software that falls prey to canned black-box testing – which simplistic application security testing tools on the market today practice – is truly bad. This means that passing a cursory penetration test reveals very little about your real security posture, but failing an easy canned penetration test tells you that you're in very deep trouble indeed.

Operations people should carefully monitor fielded systems during us for *security breaks.* Simply put, the attacks will happen, regardless of the strength of design and implementation, so monitoring software behavior is an excellent defensive technique. Knowledge gained by understanding attacks and exploits should be cycled back into the development organization, and security practitioners should explicitly track both threat models and attack patterns.

Note that risks crop up during all stages of the software cycle, so a constant risk analysis thread, with recurring risk tracking and monitoring activities, is highly recommended.

…as a multidisciplinary effort

By and large, software architects, developers and testers remain blithely unaware of the software security problem. One essential form of best practices involves training software development staff on critical software security issues. The most effective form of training begins with a description of the problem and demonstrates its impact and importance. Beyond awareness, more advanced software security training should offer coverage of security engineering, design principles and guidelines, implementation risks, design flaws, analysis techniques, software exploits, and security testing. Each best practice called out earlier is a good candidate for in-depth training.

Software security can and should borrow from other disciplines in computer science and software engineering when developing and evolving best practices. Of particular relevance are:

- security requirements engineering;
- design for security, software architecture, and architectural analysis;
- security analysis, security testing, and use of Common Criteria;
- guiding principles for software security and case studies in design and analysis;
- auditing software for implementation risks, automated tools, and technology developments (code scanning, information flow and so on); and
- common implementation risks (buffer overflows, race conditions, randomness, authentication systems, access control, applied cryptography, and trust management).

Much work remains to be done in each of the best practice areas, but some basic practical solutions should be adapted from areas of more mature research.

As the trinity of trouble – connectedness, complexity and extensibility – continues to impact software security in a negative way, we must begin to grapple with the problem in a more reasonable fashion. Integrating a decent set of best practices into the software development lifecycle is an excellent way to do this. Although software security as a field has much maturing to do, it has much to offer to those practitioners interested in striking at the heart of security problems.

Gary McGraw is chief technology officer at Cigital. His real-world experience is grounded in years of consulting with major corporations and software producers. He serves on the technical advisory boards of Counterpane, Fortify, and Indigo. He also is coauthor of Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001),* Java Security *(John Wiley & Sons, 1996) and four other books. Contact him at gem@cigital.com.*

cigital

Cigital, Inc.
21351 Ridgetop Circle
Suite 400
Dulles, Virginia
20166

T 800 824 0022
703 404 9293
F 703 404 9295
E info@cigital.com

www.cigital.com