Foreword

Software Security and Code Review with a Static Analysis Tool

On the first day of class, mechanical engineers learn a critical lesson: Pay attention and learn this stuff, or the bridge you build could fall down. This lesson is most powerfully illustrated by a video of the Tacoma Narrows Bridge shaking itself to death (http://www.enm.bris.ac.uk/anm/tacoma/tacoma.html). Figure 1 shows a 600-foot section of the bridge falling into the water in 1940. By contrast, on the first day of software engineering class, budding developers are taught that they can build anything that they can dream of. They usually start with "hello world."

Figure 1 A 600-foot section of the Tacoma Narrows bridge crashes into Puget Sound as the bridge twists and torques itself to death. Mechanical engineers are warned early on that this can happen if they don't practice good engineering.

An overly optimistic approach to software development has certainly led to the creation of some mind-boggling stuff, but it has likewise allowed us to paint ourselves into the corner from a security perspective. Simply put, we neglected to think about what would happen to our software if it were intentionally and maliciously attacked.

Much of today's software is so fragile that it barely functions properly when its environment is pristine and predictable. If the environment in which our fragile software runs turns out to be pugnacious and pernicious (as much of the Internet environment turns out to be), software fails spectacularly, splashing into the metaphorical Puget Sound.

The biggest problem in computer security today is that most systems aren't constructed with security in mind. Reactive network technologies such as firewalls can help alleviate obvious script kiddie attacks on servers, but they do nothing to address the real security problem: bad software. If we want to solve the computer security problem, we need to do more to build secure software.

Software security is the practice of building software to be secure and function properly under malicious attack. This book is about one of software security's most important practices: code review with a static analysis tool.

As practitioners become aware of software security's importance, they are increasingly adopting and evolving a set of best practices to address the problem. Microsoft has carried out a noteworthy effort under its Trustworthy Computing Initiative. Many Cigital customers are in the midst of enterprise scale software security initiatives. Most approaches in practice today encompass training for developers, testers, and architects; analysis and auditing of software artifacts; and security engineering. 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.

In my book *Software Security*, I introduce a set of seven best practices called *touchpoints*. Putting software security into practice requires making some changes to the way most organizations build software. The good news is that these changes don't need to be fundamental, earth shattering, or cost-prohibitive. In fact, adopting a straightforward set of engineering best practices, designed in such a way that security can be interleaved into existing development processes, is often all it takes.

Figure 2 specifies the software security touchpoints and shows how software practitioners can apply them to the various software artifacts produced during software development. This means understanding how to

work security engineering into requirements, architecture, design, coding, testing, validation, measurement, and maintenance.

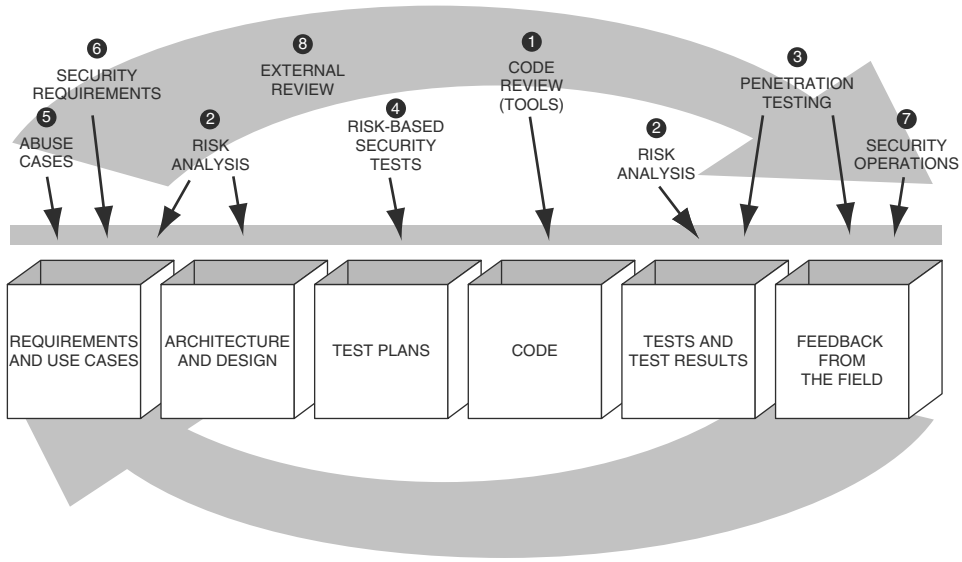

Figure 2 The software security touchpoints as introduced and fleshed out in *Software Security: Building Security In.*

Some touchpoints are, by their very nature, more powerful than others. Adopting the most powerful ones first is only prudent. The top two touchpoints are code review with a static analysis tool and architectural risk analysis. This book is all about the first.

All software projects produce at least one artifact: code. This fact moves code review to the number one slot on our list. At the code level, the focus is on implementation bugs, especially those that static analysis tools that scan source code for common vulnerabilities can discover. Several tools vendors now address this space, including Fortify Software, the company that Brian and Jacob work for.

Implementation bugs are both numerous and common (just like real bugs in the Virginia countryside), and include nasty creatures such as the notorious buffer overflow, which owes its existence to the use (or misuse) of vulnerable APIs (e.g., `gets()`, `strcpy()`, and so on in C). Code review processes, both manual and (even more important) automated with a static analysis tool, attempt to identify security bugs prior to the software's release.

Of course, no single technique is a silver bullet. Code review is a necessary but not sufficient practice for achieving secure software. Security bugs (especially in C and C++) are a real problem, but architectural flaws are just as big of a problem. Doing code review alone is an extremely useful activity, but given that this kind of review can only identify bugs, the best a code review can uncover is around 50% of the security problems. Architectural problems are very difficult (and mostly impossible) to find by staring at code. This is especially true for modern systems made of hundreds of thousands of lines of code. A comprehensive approach to software security involves holistically combining both code review and architectural analysis.

By its very nature, code review requires knowledge of code. An infosec practitioner with little experience writing and compiling software will be of little use during a code review. The code review step is best left in the hands of the members of the development organization, especially if they are armed with a modern source code analysis tool. With the exception of information security people who are highly experienced in programming languages and code-level vulnerability resolution, there is no natural fit for network security expertise during the code review phase. This might come as a great surprise to organizations currently attempting to impose software security on their enterprises through the infosec division. Even though the idea of security enforcement is solid, making enforcement at the code level successful when it comes to code review requires real hands-on experience with code.

The problem is that most developers have little idea what bugs to look for, or what to do about bugs if they do find them. That's where this book, *Secure Programming with Static Analysis*, comes in. The book that you have in your hands is the most advanced work on static analysis and code review for security ever released. It teaches you not only what the bugs are (what I sometimes call the "bug parade" approach to software security), but how to find them with modern static analysis tools and, more important, what to do to correct them. By putting the lessons in this book into practice, you go a long way toward helping to solve the software security problem.

Gary McGraw, Ph.D.
Berryville, Virginia
March 6, 2007

Company: www.cigital.com
Podcast: www.cigital.com/silverbullet
Blog: www.cigital.com/justiceleague
Book: www.swsec.com