Bridging the Gap between Software Development and Information Security

Kenneth R. van Wyk
Cigital and KRvW Associates

Gary McGraw
Cigital

Traditionally, software development efforts in large corporations have been about as far removed from information security as they were from human resources or any other business function. Software development has also had the tendency to be highly distributed among business units and thus not even practiced in a cohesive, coherent manner. In the worst cases, busy business unit executives trade roving bands of developers like Pokémon cards in a fifth-grade classroom (in an attempt to get ahead). Suffice it to say, none of this is good.

The disconnect between security and development has ultimately produced software development efforts that lack any sort of contemporary understanding of technical security risks. Today's complex and highly connected computing environments trigger myriad security concerns, so by blowing off the idea of security entirely, software builders virtually guarantee that their creations will have way too many security weaknesses that could—and should—have been avoided. This article presents some recommendations for solving this problem. Our approach is born out of experience in two diverse fields: software security and information security. Central among our recommendations is the notion of using the knowledge inherent in information security organizations to enhance secure software development efforts.

Don't stand so close to me

Best practices in software security include a manageable number of simple activities that should be applied throughout any software development process (see Figure 1). These lightweight activities should start at the earliest stages of software development and then continue throughout the development process and into deployment and operations.

Although an increasing number of software shops and individual developers are adopting the software security touchpoints we describe here as their own, they often lack the requisite security domain knowledge required to do so. This critical knowledge arises from years of observing system intrusions, dealing with malicious hackers, suffering the consequences of software vulnerabilities, and so on. Put in this position, even the best-intended development efforts can fail to take into account real-world attacks previously observed on similar application architectures. Although recent books[1,2] are starting to turn this knowledge gap around, the science of attack is a novel one.

Information security staff—in particular, incident handlers and vulnerability/patch specialists—have spent years responding to attacks against real systems and thinking about the vulnerabilities that spawned them. In many cases, they've studied software vulnerabilities and their resulting attack profiles in minute detail. However, few information security professionals are software developers (at least, on a full-time basis), and their solution sets tend to be limited to reactive techniques such as installing software patches, shoring up firewalls, updating intrusion detection signature databases, and the like. It's very rare to find information security professionals directly involved in major software development projects.

Sadly, these two communities of highly skilled technology experts exist in near complete isolation, yet their knowledge and experience bases are largely complementary. Finding avenues for interdisciplinary cooperation will likely bear fruit in the form of fielded software that's better equipped to resist well-known and easily predicted attacks. A secondary benefit of any interdisciplinary cooperation is gaining information security personnel with a much better understanding of the applications that they're tasked with protecting.

Every silver lining's got a touch of gray

A complete description of every software security best practice is far beyond this article's scope, but we

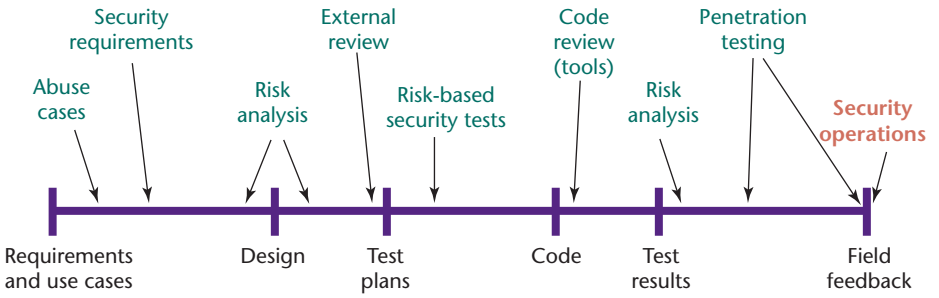

Figure 1. Software security best practices or "touchpoints." Although the software artifacts are laid out according to a traditional waterfall model, most organizations follow an iterative approach today: best practices are cycled through more than once as the software evolves.

can provide a high-level description of the most effective software security touchpoints available today.

Requirements: Abuse cases

The concept of abuse case development is derived from use case development. In an abuse case, the developer considers software's deliberate misuse and ponders its corresponding effect. When addressing user input, for example, the developer can construct a series of abuse cases that describes in some detail how malicious users can and will attempt to overflow input buffers, insert malicious data, and so on. An abuse case depicts these scenarios as well as how the software should respond to them. As with their use case counterparts, each abuse case then drives a requirement and corresponding test scenario for the software.

Design: Business risk analysis

Assessing the business impact likely to result from a successful software compromise is a critical undertaking. If no one explicitly tackles this issue, a security analysis will fall short in the "who cares?" department. A good risk analysis considers questions of the project's cost to the parent organization sponsoring the software in terms of both direct cost (liability, lost productivity, and rework) and indirect cost (reputation and brand damage).

Design: Architectural risk analysis

Similar to a business risk analysis, an architectural risk analysis assesses the technical security exposures in an application's proposed design and links them to business impact. Starting with a high-level depiction of the design, the analysis team considers each module, interface, interaction, and so forth against known attack methodologies and their likelihood of success. To provide a forest-level view of a software system's security posture, the analysts typically apply such analyses against a design's individual subcomponents as well as to the design as a whole. Attention to security's holistic aspects is paramount: at least 50 percent of all security defects are architectural in nature.

Test planning: Security functionality testing

Just as testers typically use functional specifications and requirements to create test scenarios and test plans (especially those testers who understand the critical notion of requirements traceability), security-specific functionality should be used to derive tests against the target software's security functions. These kinds of investigations generally include tests that verify security features such as encryption, user identification, logging, confidentiality, authentication, and so on. These are "positive" security features for white hats.

Test planning: Risk-driven testing

Thinking like a good guy isn't enough: you have to don your black hat and think like a bad guy. Risk-based test scenarios are the natural result of the process of assessing and prioritizing software's architectural risks. Each architectural risk and abuse case considered should be described and documented down to a level that clearly explains how an attacker might go about exploiting a weakness and compromising the software. Such descriptions can help generate a priority-based list of test scenarios for later "adversarial" testing.

Implementation: Code review

The design-centric activities described thus far focus on architectural flaws built into software design, but they completely overlook implementation bugs that the coders might introduce during coding. Implementation bugs are both numerous and common (just like real bugs in the Virginia countryside) and can include nasty creatures such as the notorious buffer overflow, which owes its existence to the use (or misuse) of vulnerable APIs. 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.

System testing: Penetration testing

System penetration testing, when used appropriately, focuses on human and procedural failures made during the software's configuration and deployment. The best kinds of penetration testing are driven by previously identified risks and are

engineered to probe risks directly to ascertain their exploitability.

Fielded system: Deployment and operations

Careful configuration and customization of any software application's deployment environment can greatly enhance its security posture. Designing a smartly tailored deployment environment for a program requires following a process that starts at the network-component level, proceeds through the operating system, and ends with the application's own security configuration and setup.

Kumbaya (for software security)

With the software security touchpoints we've just listed in mind, let's turn to the issue at hand: how information security professionals can best participate in the software development process. If you're a CISSP, an operational security professional, or a network administrator, this Bud's for you. Let's go back through the activities we just covered and give some recommendations relevant to both software developers and information security practitioners.

Abuse cases

Folding information security into abuse case development is such low-hanging fruit that the fruit itself is dirt-splattered from the latest thunderstorm. Simply put, information security professionals come to the table with the (rather unfortunate) benefit of having watched and dissected years of attack data, built forensics tools,[3] created profiles of attackers, and so on. This might make them jaded and surly, but at least they intimately know what they're up against.

Many abuse case analysis efforts begin with brainstorming or "white boarding" sessions during which the development team describes an application's use cases and functional requirements while a room full of experts pontificate about how an attacker might attempt to abuse the system. Properly participating in these exercises involves carefully and thoroughly considering similar systems and the successful attacks against them. Getting past your own belly button is especially important to abuse case success, so consider other domains that could be relevant to the application under review while you're at it. Once again, real battle experience is critical. Information security people are likely to find (much to their amusement) that the software developers in the room are blissfully unaware of many of the attack forms found daily beyond the network perimeter. Of course, many of the uninformed are also naturally skeptical unbelievers. While converting these skeptics, try to avoid succumbing to the tendency toward hyperbole and exaggeration that is unfortunately common among security types. There's nothing worse than a blustery security weenie on his high horse about some minor skirmish. Don't overstate the attacks you've seen and studied, just stick to the facts and be prepared to back up your statements with actual examples. Knowledge of actual software technology a plus.

Business risk analysis

The most important people to consult when assessing software-induced business risks are the business stakeholders behind the software. In organizations that already practice business-level technology analysis, this tends to be quite well understood. (Unfortunately, technological assessment of the business situation stops well before the software level in most of these organizations.) Enhancing a standard approach is easy with a few additional questions: What do the people asking for this software think about security? What do they expect? What are they trying to accomplish that a successful attack might thwart? What worries them about security?

The value information security professionals bring to answering these questions comes from their wealth of experience in seeing security impacts first-hand when similar business applications were compromised. It gives them the opportunity to knowledgeably answer several other security-related questions: What sorts of costs have similar companies incurred from attacks? How much downtime was involved? What was the resulting publicity in each case? In what ways was the organization's reputation tarnished? Information security people can provide input and flesh out a conversation with relevant stories. Here again, take care to not overstate the facts. When citing incidents at other organizations, be prepared to back up your claims with news reports and other third-party documentation.

Architectural risk analysis

Now we're getting to the technical heart of the software development process. For architectural risk analysis to be effective, security analysts must possess a great deal of technology knowledge covering both the application and its underlying platform, frameworks, languages, functions, libraries, and so on. The most effective information security team member in this situation is clearly one who is a technology expert with solid experience around particular software tools. With this kind of knowledge under his or her belt, the information security professional can provide real-world feedback into the process. If the analysis team is discussing a particular network encryption protocol's relative strengths and weaknesses, for example, information security can provide perspective to the conversation. All software has potential weaknesses, but was component X involved in any actual attacks? Are there known

vulnerabilities in the protocol the project is planning to use? Is a commercial off-the-shelf component or platform a popular attacker target? Or does it have a stellar reputation and only a handful of properly handled published vulnerabilities and known attacks? Feedback of this sort is extremely useful for prioritizing risk and weaknesses as well as for deciding on what, if any, mitigation strategies to pursue.

Software developers and information security staff can benefit greatly from the respective experiences of the other.

Test planning

Although test planning and execution are generally performed by quality assurance (QA) and development groups, testing represents another opportunity for information security to have a positive impact. Testing—especially risk-based testing—must not only cover functionality, it should closely emulate the steps that an attacker will take when breaking a target system. Highly realistic scenarios (the security analog to real user scenarios) are much more useful than arbitrary pretend "attacks." Standard-issue testing organizations, if they're effective at all, are most effective at designing and performing tests based on functional specifications. Designing risk-based test scenarios is a rather substantial departure from the status quo and should benefit from the experience base of security incident handlers. In this case, information security professionals who are good at "thinking like a bad guy" are the most valuable resources.

Code review

By its very nature, code review requires knowledge of code. An information security practitioner with little experience writing and compiling software is of little use during a code review. If you don't know what it means for a variable to be declared in a header or an argument to a method to be static/final, staring at lines of code all day isn't going to help. Because of this, the code review step is best left in the hands of the development organization, especially if it's 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 those organizations currently attempting to impose software security on their enterprise through the information security division. Although 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. It's definitely not sufficient to arm the information security team with a static code scanner and expect them to deliver substantive feedback to the coders.

Penetration testing

Although testing software to a functional specification has traditionally been QA's domain, penetration testing is usually the domain of information security and incident-handling organizations. As such, the fit here for information security participation is very natural and intuitive. Of course, several subtleties can't be ignored. Most penetration testing today focuses its attention on network topology, firewall placement, communications protocols, and the like, thus it's an outside→in approach that barely begins to scratch an application's surface. Penetration testing must encompass a more inside→out approach that takes into account risk analyses and other software security results as it's performed. This distinction is sometimes described as the difference between "network penetration testing" and "application penetration testing." Software security is much more interested in the latter.

Also worth noting is the use of various black-box penetration tools. Network security scanners such as nessus, nmap, SATAN, and the like are extremely useful because of the countless ways in which to configure (and misconfigure) complex networks and their various services. Application security scanners are nowhere near as useful, so if by an "application penetration test" you mean running an application security testing tool and gathering the results, you have a long way to go to make your approach hold water. It goes almost without saying that software testing isn't something that a set of canned tests can handle, no matter how large the can. The idea of testing any arbitrary program with, say, a few thousand tests determined in advance before the software was even conceived is ridiculous. The idea of testing any arbitrary program with a few hundred application security tests is just as silly.

The good news about penetration testing and information security involvement is that it's most likely already underway. The bad news is that information security must up its level of software clue to most effectively perform penetration testing.

Deployment and operations

Many software developers would argue that deployment and operations aren't even part of the software development process. Even if this view is correct, we can't properly address operations and deployment concerns if the software is so

poorly constructed that it falls apart no matter what kind of solid ground we place it on. Put bluntly, operations organizations have put up with some rather stinky software for a long time, which has made them wary. If we can set that argument aside for a moment and look at the broader picture—that is, safely setting up the application in a secure operational environment and running it accordingly—then the work that needs doing can certainly be positively affected by information security. The best opportunities exist in fine-tuning access controls at the network and operating system levels, as well as in configuring an event-logging and monitoring mechanism that's most effective during incident response operations. Attacks will happen, so be prepared to clean up the mess afterwards. This advice is pretty much a "no duh" for information security organizations, which is why their involvement in this step is so important.

Come together (right now)

Even if you accept our recommendations wholesale as worthy, the act of aligning information security and software development is a serious undertaking (and not one for the faint of heart). Close cooperation with the development organization is essential to success. If developers perceive information security people to be the security police or "those people with sticks who show up every once in a while and beat us soundly for reasons we don't understand," you have a problem that must be addressed.

In many cases, developers are more than willing to accept guidance and advice from information security people who know what they're talking about. One problem is that they don't know who to talk to, who might help them, and who might just be a blowhard security weenie. To fix this problem, the first step for any of you information security professionals who want to help out with development efforts should be to reach out to the developers, roll up your sleeves, and offer to assist.

Once you've made the developers aware of your willingness to help, consider taking small steps toward the goals laid out in this article. Rather than trying to become involved in every phase all at once in a giant world-changing endeavor, try one at a time. Be careful not to overwhelm the overall system by attempting to make too many changes at once.

Another positive step is for the information security troops to take the time to learn as much as they can about software development in general and their organization's software development environment in particular. Study and learn about the types of applications your software people develop, why they're doing it (that is, for what business purpose the software is being built), what languages, platforms, frameworks, and libraries are being used, and so on. Showing up with a clue is much better than showing up willing but clueless. Software people aren't the most patient people on the planet, and you often have only one shot at getting involved. If you help, that's great, but if you hinder, it'll be the last time they talk to you.

In the end, success or failure is as likely to be driven by the personalities of the people involved as anything else. Success will certainly not be guaranteed, even with the best of intentions and the most careful planning. Beer helps.

The interesting thing about software security is that it appears to be in the earliest stages of development, much as the field of information security itself was 10 or so years ago. The security activities described here discuss only the tip of the best practice iceberg, but the good news is that these best practices are emerging at all. Naturally, the software security discipline will evolve and change with time, and best practices and advice will ebb and flow like the tides at the beach, but the advice here is likely to bear fruit for some time.

The recommendations in this article are based on years of experience with a large dose of intuition thrown in for good measure. We've presented them in the hopes that others will take them, consider them, adjust them, and attempt to apply them in their organizations. We believe that software developers and information security staff can benefit greatly from the respective experiences of the other, but much work will need to be done before the practical recommendations made here prove themselves to be as useful in practice as we believe they will be. ▢

References

1. G. Hogland and G. McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley, 2004.
2. J. Koziol et al., *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*, John Wiley & Sons, 2004.
3. D. Farmer and W. Venema, *Forensic Discovery*, Addison-Wesley, 2004.

Kenneth R. van Wyk *is a principal consultant at KRvW Associates and director of research at Cigital. His interests include software security and incident-handling. Van Wyk has a BS in mechanical engineering from Lehigh University. Contact him at ken@krvw.com.*

Gary McGraw *is chief technology officer of Cigital. His real-world experience is grounded in years of consulting with major corporations and software producers. McGraw is the coauthor of* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison- Wesley, 2001),* Java Security *(John Wiley & Sons, 1996), and four other books. He has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.*