Sponsored by IBM

Integrating Security into Development, No Pain Required

September 2011

A SANS Whitepaper

Written by: Dave Shackleford

Introduction

The majority of today's information security problems can be traced to flaws in code. Whether these security problems affect operating system components, client applications, web applications or specialized code that runs power generation or other equipment-control systems, the majority of well-publicized vulnerabilities are related to coding errors and implementation issues.

Within the running list of the Top 25 Most Dangerous Software Errors[1] maintained by SANS, three categories emerge: insecure interaction among software components, risky resource management when coding and porous defenses (due to a variety of implementation issues). The biggest question may be why so many application bugs and coding errors continue to cause major security events when we have had decades to deal with these and other common vulnerabilities found in applications today.

The answer is not simple. Coding is an art, and no two developers work the same way, leading to inconsistencies, vulnerabilities and problems with upgrades and code review. Maintaining a code base in a manner that can be checked throughout its life cycle continues to be problematic for developers and security teams alike.

One likely reason for the difficulties is the relationship among developers and security personnel, which has traditionally been perceived as being like oil and water. Although developing bug-free code might be on developers' minds, their priorities are creating the cool factors of applications, meeting deadlines, minimizing time to market and implementing other means to serve a fast-paced business plan. Developers perceive security, with its priority of making sure bad things don't happen to the applications and their critical data, as getting in the way of development priorities.

Clearly, there are many benefits to development teams and security groups working together, ranging from improved code to more efficient development operations and quality-control processes. This paper looks at software development from both the security and development perspectives, and then evaluates what tools and techniques can help integrate security into development cycles without slowing down the process or creating too much overhead.

1 www.sans.org/top25-software-errors

Divergent Points of View

Those who pay attention to vendors' data-breach disclosure announcements and published vulnerabilities know that the current state of security is not great. According to DataLossDB, there were more than 400 data-breach incidents in 2011 (as of this writing) and well over 100 million records accessed, lost or stolen.[2]

Bug announcements, which keep security professionals and IT operations teams busy, are also common occurrences. For the security community, one of the most well-known bug announcement events is Microsoft's monthly "Patch Tuesday." This event produced hundreds of patches in 2011, many of them critical in nature. A wide variety of flaws are also being noted in other operating systems and applications, including browsers, browser plug-ins, web applications and even (sometimes) security applications.

Years of secure application initiatives and frameworks have yielded great improvements in applications and patch cycles. For example, many remotely available services in well-known operating systems and applications are now less susceptible to attacks. In their place, more client-side applications (running on multiple devices) and Internet-facing applications are vulnerable than ever before. SANS listed such applications as the top two priorities in its "Top Cyber Security Risks" report. Although the report notes improvements in operating system software, it also cites the rising number of zero-day attacks against applications as a top concern.[3] The last category, zero-day vulnerabilities, factored prominently in several highly public breaches, including those involving security firm RSA, Google and others.

Vendors have greatly improved the way they react and respond to the security community, particularly the security researchers who find and publicize bugs. Many companies now offer "bug bounty" programs that seek to reward researchers who discover significant security flaws before attackers are able to exploit those flaws. Google offers up to $3,133.70 for identification of severe flaws in its software products, and Mozilla offers up to $3,000 per bug. Facebook recently started a similar program that paid out $40,000 in just over three weeks.[4] Other sites, such as the Zero Day Initiative site,[5] maintain a running chronicle of zero-day vulnerabilities reported by researchers. This site also provides a list of published and upcoming vulnerability announcements from many vendors, along with severity scores for the vulnerabilities.

With all this attention on code flaws and the complex vulnerabilities found in a variety of software applications, security professionals are focusing more of their attention on developers and their methods.

Security teams tend to focus on the three primary objectives, or *pillars of information security* as they are often called: confidentiality, integrity and availability. In most cases, confidentiality and integrity are the primary concerns, with availability coming in third (not always, but commonly). Security teams are evaluated on their ability to protect data confidentiality and integrity, and so they are more concerned with these aspects of development and operations. This is in direct opposition to developers' main priority of availability.

---

2 www.datalossdb.org

3 www.sans.org/top-cyber-security-risks/summary.php

4 http://blog.eset.com/2011/08/30/facebook-bug-bounty-payout-tops-40k

5 www.zerodayinitiative.com

Unfortunately, many, if not most security teams do not have members with coding backgrounds, which can make interactions with development organizations challenging. This situation has led to a significant gap in successful integration of security into development life cycles, primarily due to the way these interactions often occur.

In many cases, developers know that security team members do not understand code, so they are somewhat skeptical of security input and guidance. On the other hand, many security professionals think development teams are arrogant and don't care about protecting data because of their focus on code and time to release. The disparity in thinking is even more evident when considering the attitudes held by both groups toward incorporating security into products. In many organizations, security has a "do-or-die" attitude, born of the need to protect critical data and computing assets from exposure. In many development life cycles, security is tagged on as a "consideration" or "toll gate" within the project plan rather than being integrated into the planning, development and production cycles.

Although security teams are usually in the position of having to slow things down and ensure confidentiality and integrity controls are in place, developers usually face pressure from the business units they support to create and update code as quickly as possible. The more critical the application to operational or business needs, the more rigid the mandate for publishing code quickly. Sometimes this brings development organizations into conflict with security teams. As an example, the primary mantra of the Agile development style, which is being adopted more and more frequently, is speed above all other things. If this seems to fly in the face of security teams' normal operations style, it's because it often does.

The key thing to keep in mind, however, is that developers are never incentivized to write bad code. In a blog post titled "Why Programmers Write Bad Code," one developer sums it up adroitly: "… programmers usually don't intend to write bad code. Actually, a programmer's first intention is usually to write code that works, which is where the problem begins."[6] Why the problem? As the post goes on to explain, many developers have been trained to write code to solve a problem or perform a function. When something breaks in that code, the developer removes the bug by writing more code that fixes the issue. In that code, however, lays the possibility of more bugs! And, thus, the cycle continues, leading to more code, more bugs and a repeating cycle that becomes harder and harder to overcome over time.

In addition, with time pressures mounting, many Quality Assurance (QA) cycles are often less in-depth than they should be. Bugs are missed or deprioritized, and fixing them gets pushed further back.

It would be easy to blame programmers for the security problems introduced in the development process. However, programmers, security personnel and upper management all share responsibility. Although lack of training, difficulties in communication and differing priorities hamper programmer–security interactions, upper management can resolve many of these issues by understanding the consequences of maintaining business as usual, instituting training to enable the groups to better understand each other and reprioritizing business and security needs.

6 http://sheriframadan.com/2011/04/why-programmers-write-bad-code

Where Security Fits in Development Processes

Security tends to be associated with a coding project in three major areas. The first is during the QA reviews, where bugs are found, logged and prioritized. Although this is a somewhat logical approach, its success relies on a good working relationship between the development and QA teams to develop functional reviews and proper code analysis, which usually involves more than one technique.

The second major integration point for security tends to be the "security toll gate," where security team members are often one part of a project review committee or planning board. Although the toll gate scenario gives security professionals some insight into the project itself, it is at a high level that may not allow them to be effective. In fact, this approach ultimately relies on security being prioritized in a top-down manner from senior leadership, which is rare.

The third and most comprehensive method integrates security and risk management into the development cycle, either with involvement of security team members or by developing a core security focus on the part of developers.

Software Development Life Cycle

Several well-known *Software Development Life Cycles (SDLCs)* integrate security in different ways. The first is the traditional waterfall SDLC, which has been used for many years. Although many versions of this model exist, the SDLC generally starts with a requirements specification phase followed by design, implementation, testing, deployment and maintenance phases.

Security can be integrated into any (and ideally all) of these phases. In most organizations that use a variant of the waterfall model, security is included with the toll gate style mentioned previously, often at the end of each phase before moving to the next one. It is, however, critically important to ensure that security is prioritized during the requirements specification phase and carried out at every phase, particularly in organizations where developers and QA teams are responsible for policing themselves (see Figure 1).

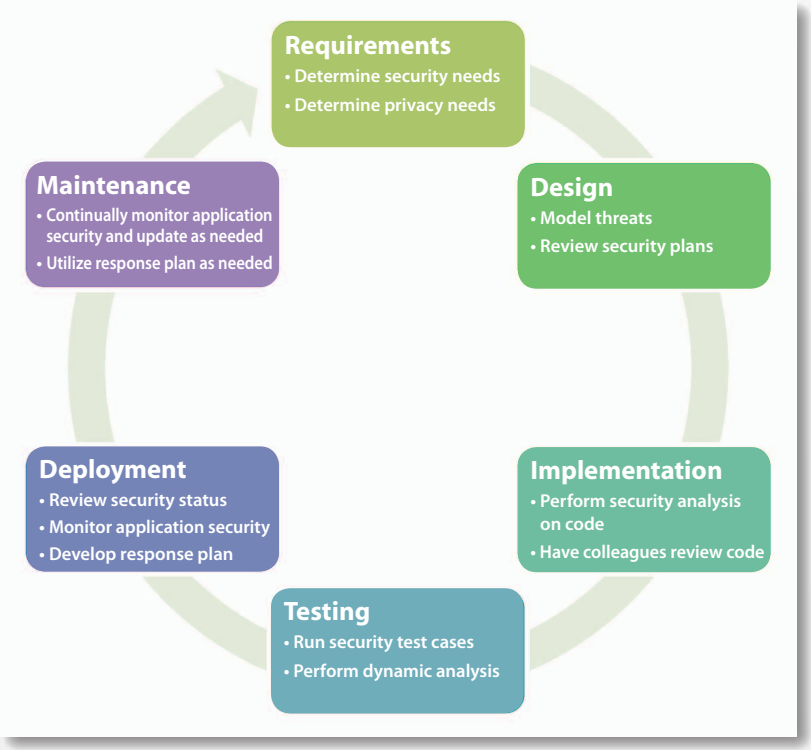

Figure 1. The SDLC with Developers Working with Security

If security drives the software development life cycle process, the responsible parties must work with developers to determine their needs and provide input during every step. This process enables the applications to be built securely while helping development teams to maintain a reasonable schedule.

Security Development Lifecycle

Microsoft is a good example of a company that was ultimately forced to adapt its SDLC due to the large number of vulnerabilities found in its flagship Windows operating system between 1999 and 2003. Microsoft has steadily improved security ever since—and continues to do so using its own development model.

Microsoft uses a popular adaptation of the waterfall SDLC that was adapted for tighter security integration and was dubbed the *Security Development Lifecycle (SDL)*. This model, illustrated in Figure 2, incorporates security training for developers before the requirements specification phase, as well as separate verification (prerelease) and response (postrelease) stages.[7] For large development teams with extensive resources, particularly those in companies that sell software to the masses, this model may make more sense, because security is vitally important throughout the entire coding and QA project, each and every time.

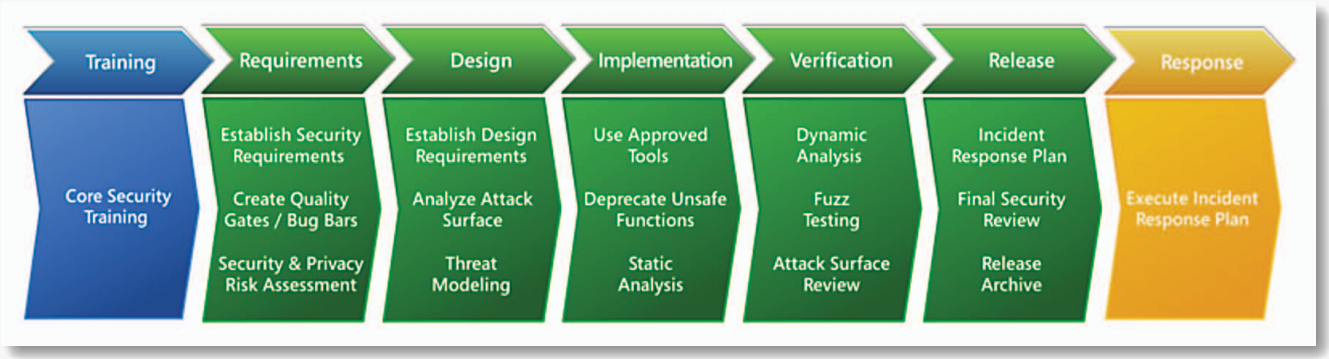

Figure 2. The Microsoft Security Development Lifecycle[8]

Agile Development

Finally, a third widely-used development cycle, known as *Agile*, focuses on personal interactions among development team members and rapid response to change. In the Agile SDLC, shown in Figure 3, pushing code quickly and responding to new feature and code change requests quickly is a priority, as is code and object re-use, and this can come into direct conflict with security principles. In an Agile organization, the most reasonable method for integrating security is either constant involvement from security team members or a dedication to security with well-trained coders on every team.

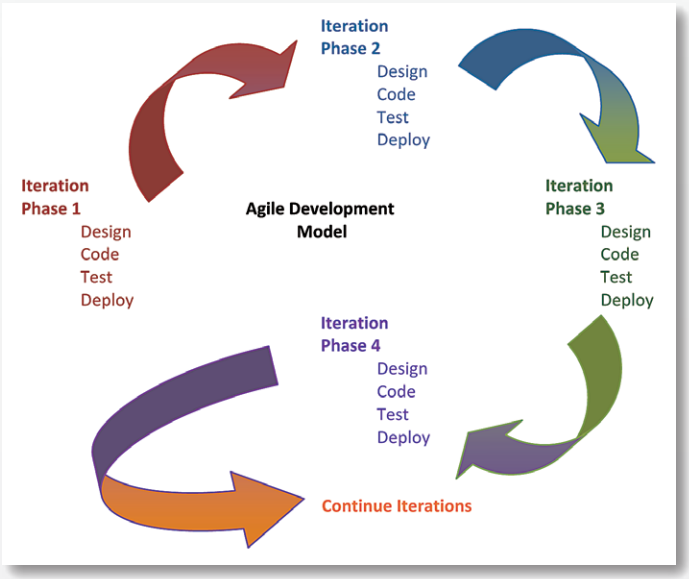

Figure 3. The Agile Software Development Lifecycle

7 www.microsoft.com/security/sdl/default.aspx

8 www.microsoft.com/security/sdl/discover/default.aspx

Development and Security Collaboration

As noted previously, several methods and tools exist for analyzing code and testing applications to find and repair bugs and errors during the life cycle—tools that security personnel and developers should be equally familiar with. Code analysis and application review for functional vulnerabilities should not be painful if done correctly. Policies should include source code analysis and dynamic program analysis to test the code base and functionality of the application. Policies and procedures for the SDLC should include both review methods, as well as a means to continually monitor applications for vulnerabilities and make efficient repairs once the applications are in production.

Source Code Analysis

The first major category of software review tools is source code analysis. Developers use source code analysis tools to inspect code and libraries, often early in the development cycle. They use policies and code signatures to determine whether common coding flaws are present, such as those listed in the SANS Top 25 Most Dangerous Software Errors. Source code analyzers commonly find buffer overflow conditions, improper function calls that could lead to format string issues, use of dangerous functions that cannot be safely called at all, and other such coding errors.

Source code analysis tools are usually linked to bug tracking and reporting mechanisms so development teams can automatically log the identified issues and fix them before they send the code to QA or production. Many code analysis tools also natively tie into Integrated Development Environments (IDEs), such as Eclipse and Microsoft Visual Studio, making it simple for developers to iteratively fix bugs quickly in the development cycle.

Dynamic Program Analysis

Dynamic program analysis, or runtime analysis, is an additional technique used to test software for security vulnerabilities and programming flaws. Unlike static code analysis and review, in which the tools scrutinize code for patterns and signatures that match known flaws, runtime analysis assesses the compiled program's execution flow as it runs, looking for particular interactions and behaviors that may be indicative of security issues.

Runtime analysis inspects how a program interacts with other platforms and code components, such as databases, middle-tier applications and others. The tools come in a number of formats:

- The first format is similar to a debugging approach, in which the code is run through a series of start–stop conditions that pause program execution when a certain condition occurs. From that point, the tester can undertake further analysis. This approach can be very time consuming and relies on the tester having detailed knowledge of the exact testing conditions to create and assess. However, the process can be beneficial when used by developers and QA teams with a more in-depth understanding of the code itself.

- Another type of dynamic code analysis tool is a code scanner. During staging and in production environments, code scanners run scans preconfigured with a security policy against published applications to look for unusual behaviors or program failures. Scanners send a stream of specific input to applications, often probing user input points, program interaction components and other areas to find known weaknesses or potentially unsafe behaviors caused by the scans. Many scanners send short, specific strings of attack data to applications, whereas others perform a technique known as *fuzzing*, usually consisting of unusual quantities or types of traffic sent to applications in the hope that they will crash or fail in some way. Many scanners also provide a number of enterprise-class features that make them attractive to both development groups and security teams. For example:
  - These tools can be set up to scan specifically for custom security policies that teams can configure simply, often from an intuitive user interface.
  - Many tools also come with preconfigured policies and reports that supply critical information for meeting compliance mandates and satisfying well-known industry standards like the OWASP (Open Web Application Security Project) Top 10[9] and the SANS Top 25 Most Dangerous Software Errors list.
  - Scanners don't require a deep level of coding knowledge and expertise to use properly, making them palatable for security teams that lack a strong coding background.

Rugged Software

Another secure development movement that is gaining traction is the Rugged Software Manifesto.[10] Started by Josh Corman, David Rice and Jeff Williams, the Rugged Manifesto recognizes the reality that software is an integral part of our lives today, and security needs to be part of every developer's training and focus when writing new code and fixing existing code. The Rugged Manifesto also emphasizes reliability, survivability and other key attributes that ultimately need to be in place for software to withstand the rigorous conditions on the Internet.

9 www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

10 www.ruggedsoftware.org

Policies and Procedures

More organizations are working toward improving the synchronization between security and development teams, as well as changing the way developers work overall. A paper published in 2007 by SANS introduced a new methodology called *Scalable and Agile Lifecycle Security for Applications (SALSA)*. Although not a true development life cycle in itself, SALSA allows web application developers, in particular, to start strategically incorporating security into the existing life cycles and adapting them for improved code review and analysis. Several key tenets of SALSA include the following policies for bringing security and developers together during the SDL:

- **Educate developers on how to analyze the enterprise attack surface or application, as well as the associated potential threats.** Although this suggestion sounds like more of a job for the security or risk-analysis team, all developers should understand the exposure points of their applications (user inputs, web-facing code, exposed function calls and so on) and take steps to design more secure code wherever possible.
- **Integrate security best practices into application life cycle and development methodologies.** This approach was largely pioneered by the Microsoft SDL mentioned previously, but SALSA also incorporates attack surface analysis and reduction.
- **Integrate security into the automated build process.** This suggestion is excellent practical advice, especially for Agile development teams, because automated builds are a core part of the rapid change-and-response element of the Agile Manifesto. The automated build process should include some general consideration of code complexity (metrics-based or otherwise), as well as automated code and unit testing using the tools discussed previously.
- **Procure security training for developers and some development training for security professionals, depending on their needs.** This training will be more of an ongoing activity that changes from year-to-year.
- **Include automated vulnerability assessments and penetration testing whenever possible.** Such assessment helps significantly with analysis of the application attack surface and provides new insights into potential vulnerabilities that might have been missed in earlier code reviews and testing.
- **Adopt more transparency!** Make it simpler for customers (both internal and external) to submit bugs, ensure all stakeholders have a clear way of getting to project information (phases, tasks, people) and provide feedback to users on progress made in development, particularly with regard to security.

11 www.sans.org/reading_room/analysts_program/brickhouses_Oct07.pdf

Putting It All Together

Whatever tools organizations choose to use, the key to getting security teams and developers onboard is taking a number of small steps, as opposed to taking one big one. This integration can be accomplished by adding a routine source code analysis into a nightly build process, for example, or setting up a scanning toll gate that includes remediation of all critical vulnerabilities before code can be pushed to the next phase.

QA teams can also be useful in this process, particularly as intermediaries between security and development. In fact, many organizations are adopting static code analysis for developers, while QA and security teams work together to use dynamic scanning tools to perform an additional level of security analysis.

Conclusion

Security and development teams *can* work together—they just need to look for common areas in which they can make improvements. Security teams focus on confidentiality and integrity of data, which can sometimes require development teams to slow down and assess code differently. At the same time, business units require developers to produce and revise code more quickly than ever, resulting in developers focusing on what works best instead of what is most secure.

This difference in focus does not mean that either side is wrong. In fact, both teams are doing exactly what they're supposed to do. However, in order to facilitate teams accomplishing both sets of goals (timely release of both functional and secure software) and working together more fluidly, changes to tools and processes are necessary.

To begin, the organization needs to make a commitment to code security and evaluate the tools that can help accomplish that goal. A combination of static and dynamic code analysis tools usually works best, although using multiple tools often costs more and requires more time for training and implementation. These tools, then, should be integrated into development and QA cycles, preferably in a largely automated manner to avoid slowing down development cycles as much as possible. Security teams should be involved in bug report reviews from both kinds of tools, and a continuous feedback loop should be created that allows all stakeholders to participate in development projects as appropriate.

Although this process will take time, the benefits will manifest in the form of a significantly more secure application landscape.

About the Author

Dave Shackleford, founder and principal consultant with Voodoo Security, is a SANS analyst, instructor and course author, as well as a GIAC technical director. He has consulted with hundreds of organizations in the areas of security, regulatory compliance, and network architecture and engineering. He has previously worked as CSO for Configuresoft, CTO for the Center for Internet Security and as a security architect, analyst and manager for several Fortune 500 companies. Dave is the co-author of *Hands-On Information Security* from Course Technology and the "Managing Incident Response" chapter in the Course Technology book *Readings and Cases in the Management of Information Security*. Recently, Dave co-authored the first published course on virtualization security for the SANS Institute. He currently serves on the board of directors at the Technology Association of Georgia's Information Security Society and the SANS Technology Institute.

SANS would like to thank its sponsor: