Baking in Security During the Systems Development Life Cycle

Kwok H. Cheng
Booz Allen Hamilton

Vulnerabilities in software that are introduced by mistake or poor practices pose a serious threat. Combating threats in today's electronic environment requires a methodical approach in building security into software from the ground up, or baking in security as some of us refer to it. The absence of a planned approach opens the possibility for application flaws which adversaries could potentially exploit. Exploiting application flaws is a likely scenario, considering a Gartner report that states that 75 percent of successful attacks are targeted towards vulnerabilities at the application layer [1].

Traditional systems development lifecycles (SDLCs) used to develop information systems generally exclude security activities. Without explicitly defining security activities during systems development, security may not be completely planned for or even considered. Integrating security activities into the SDLC helps address this issue. Security activities can be included in phases such as requirements, design, build, or test to ensure that governance or industry best practices are satisfied, and that the goal of the system is achieved without conflict.

The Benefits of Integrating Security Into the SDLC

Bolting on security post-development is no longer sufficient in delivering systems on time, under budget, and with the proper level of protection in place. Addressing the issue, the Computer Emergency Readiness Team Coordination Center (CERT/CC) states the following:

> … many security incidents are the result of exploits against defects in the design or code of software. The approach most commonly employed to address such defects is to attempt to retroactively bolt on devices that make it more difficult for those defects to be exploited. This is not a solution that gets to the root cause of the problem and threat. [2]

It is estimated that bolting-on security post-development costs roughly three times more than the cost of built-in security. According to Gartner, if 50 percent of vulnerabilities were removed before production of purchased and internally developed software, enterprise configuration management (CM) costs and incident-response costs could be reduced by 75 percent each [3].

Building in security ensures proper and cost-effective protection. Assets that are identified and categorized can be more appropriately protected in terms of adequacy and cost. For example, when risk analysis is performed during the requirements phase, security risks may be identified which translate into security requirements.

Approaches for Integrating Security Into the SDLC

Regardless of the SDLC model used (e.g. waterfall, spiral, Rational Unified Process), the SDLC represents a phased approach to the development of a system. An appropriate model should facilitate the reanalysis and validation of the plans, requirements, and design at multiple points throughout its life cycle. Whether regarded as a phase or discipline, an SDLC is composed of several common groupings of activities: requirements, design and build, test and deploy, operations and maintenance, and disposal, with full life-cycle support activities such as risk management, CM, and training. Security can be integrated into these different points of the SDLC independent of the model.

Figure 1 describes each phase or discipline of an SDLC with the associated security activities.

Full Life-Cycle Support Activities

Risk Management

Risk management includes performing security risk analyses at different points of the life cycle. Security risk analysis serves to identify and mitigate security-related risks.

The results of the risk analysis feed into the risk management process of identifying, controlling, and eliminating or minimizing uncertain events that may affect the system. Risk analysis should be repeated iteratively throughout the system's life cycle as different activities allow opportunities to identify new or changing risks. For instance, as the project progresses forward and activities shift from requirements development to high-level system design, additional information will be uncovered about the application. This new information may reveal risks not previously identified such as use of vulnerable components or a flawed authentication model. We also know that changes to design during the build phase are almost always certain to occur. That is why it is important to also perform a risk analysis on the system after it has been built.

CM

Inaccurate or incomplete CM may enable malicious developers to exploit the shortcomings in the CM process in order to make unauthorized or undocumented changes to the software. Lack of proper software change control, for example, could allow rogue developers to insert or substitute malicious code, introduce exploitable vulnerabilities, or remove or modify security controls implemented in the software. Good CM practices also prevent the introduction of unintentional flaws into software code. For example, a developer makes a seemingly harmless modification to the application's interface before deployment and is able to bypass the CM process. This change unintentionally gives normal, restricted users elevated privileges to view information they normally would not be allowed to access. Since the CM process was bypassed, this change was not analyzed or tested for its security impact as it normally should have been.

By tracking and controlling all of the artifacts of the system development process, CM helps ensure that changes made to those artifacts cannot compromise the trustworthiness of the software as it evolves through each phase of the process. Coming from a systems development background, I have had the opportunity to practice CM hands on. Now that I am involved in security, I have found that good CM is no different than CM for security. Thus, practicing good CM is good security.

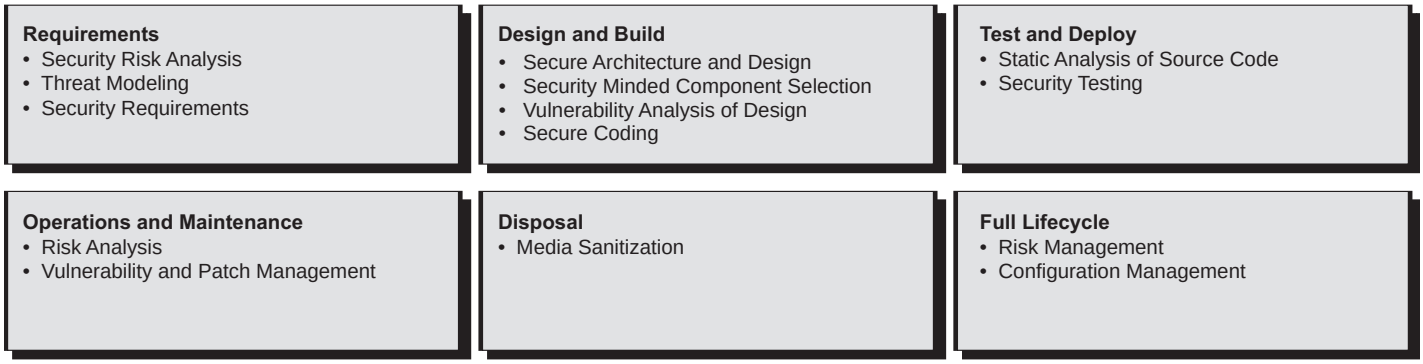

Figure 1: *Phases of an SDLC*

Requirements

During requirements, a risk analysis of the initial functional requirements should be conducted. This initial analysis should form an important source of security requirements for the system and is the best way to start if the project team is having difficulty identifying initial security requirements. This helps address the familiar question *where do I start?*

Performing threat modeling and misuse/abuse cases are also important sources for developing appropriate security requirements as well as secure design. Threat modeling attempts to identify potential threats to the software, estimating the risk vulnerabilities may be exploited by these threats, and then defining countermeasures to mitigate the risks to the application. The development of misuse/abuse cases helps articulate scenarios in which security requirements can be derived. For example, a security requirement for validating user input could be developed to address a misuse/abuse scenario in which a user enters malicious scripting in the application's input field. Other sources for security requirements could be from policy, laws, regulations, standards, or best practices documents such as the following:

- Federal Information Security Management Act [4].
- Defense Information Systems Agency Security Technical Implementation Guides [5].
- National Institute of Standards and Technology (NIST) 800 series Special Publications [6].
- Comprehensive Lightweight Application Security Process [7].
- The Open Web Application Security Project (OWASP) best practices [8].

Additional security requirements discovered during the design, build, and test phases should be incorporated back into the system's requirements specification. Security flaws and defects found during testing should be analyzed to determine whether they originated with the system's requirements, design, or implementation and the root cause should be corrected in order to remove or mitigate the risk associated with that vulnerability. Tracing the origin of security defects is a form of measurement analysis that will help the organization identify where processes or products can be improved. This can tie into an organization's overall process improvement effort (e.g., Capability Maturity Model® Integration, International Organization for Standardization 9001, Six Sigma).

"Risk analysis should be repeated iteratively throughout the system's life cycle as different activities allow opportunities to identify new or changing risks."

Risk analysis can also help prioritize security requirements to focus resources on those components or functions that introduce the greatest vulnerabilities to the system. Specifically, security requirements that help mitigate the most critical risks identified from the risk assessment should be given priority over other requirements. In addition, it aids in striking a balance between security and functionality of the system.

Security requirements should not be treated separately from functional system requirements. Rather, the scope of the requirements development phase should be expanded to include security considerations. Ideally, both security and non-security requirements should exist in a solidified system requirements specification.

Design and Build

Integrating security into the selection or development of the architecture can be seamlessly achieved. When evaluating architectures for use, consider the security aspects of the candidate models or components, such as the following:

- Existing known vulnerabilities.
- Integration with other security products, such as an enterprise-level authentication product.
- Resiliency against certain attacks (e.g., cross-site scripting, structured query language [SQL] injection).[1]
- Ability to meet security requirements.

For example, when selecting commercial off-the-shelf products for integration, a search in the National Vulnerability Database (NVD) can determine if any known vulnerabilities exist for that particular product.

Comparing the access control mechanisms in Java frameworks such as *Struts* or *Spring* might be an example of a security consideration during architecture selection. Assessing how each framework handles security helps in developing the correct architecture – something that is often overlooked.

A secure architecture incorporates overlaps of security controls among components as well as built-in fall backs if the security controls of any one component are compromised. This is commonly referred to as *defense-in-depth*. For example, I typically advise customers to implement three layers of defense against SQL injection attacks. First, client-side input restrictions should be implemented for the purposes of thwarting casual attacks and providing faster response times in generating error messages. Then, input validation should be built into the application as a sec-

ondary line of defense. If the application receives repeated suspicious input at this point, it may be reasonable to assume that an attack is being attempted. Third, database calls should be constructed using parameterized queries to eliminate the possibility of SQL injection and to aid in the manageability of queries. In this architecture, there are fall backs in case a line of defense is compromised. Although all mechanisms provide an additional layer of defense, they also clearly have their own advantages in providing other features such as performance, incident detection, and manageability. Secure architecture of the application should also include countermeasures to compensate for vulnerabilities or inadequate assurance levels in its individual components and inter-component interfaces.

Designing for security involves both proactive detection and prevention of attacks, along with minimizing the impacts of successful attacks. Mechanisms such as fail-safe design, self-testing, exception handling, warnings to administrators or users, and self-reconfigurations should be designed into the application itself, while additional prevention, monitoring, and response mechanisms (e.g., application firewalls, intrusion detection systems, and security kernels) should be incorporated as defense-in-depth measures in the application's execution environment. The idea is that security mechanisms should be embedded into the application itself in addition to the traditional prevention, monitoring, and response mechanisms that are implemented around the perimeter of the application.

A fail-safe design, sometimes referred to as fail-secure, is a design that allows the application, in the event of an unrecoverable error, to fail without causing the application to be insecure. An example might be an application defaulting to no-access when a failed connection does not allow it to validate user permissions. At the source-code level, implementation of fail-safe may include having a default case in conditional code to protect the application in a situation where the other conditions are somehow not met. Self-testing is when the application can verify that its own security functionality is working properly. Robust exception handling is critical to security. It helps identify the source of problems and can be used for investigative purposes. Relying solely on exception handling by the Web server is not recommended, as it cannot capture specific actions taken on the application.

Designing for security involves the following key practices:

- Envision potential targets for attacks. What component of the system is most likely to be attacked?
- Analyze attacks from both external and internal sources; do not forget about malicious users inside the network.
- Design and include proven authentication methods, access policies, cryptographic algorithms, or other forms of security controls where appropriate.

Integration of security into the coding phase is relatively straightforward. For custom coding, developers should implement secure coding practices. For example, developers should ensure that user input is validated so that it only contains data that is expected, i.e., no malicious scripting or malformed input. Code should be thread-safe, so that during simultaneous execution by multiple threads, the code functions correctly and does not inadvertently access the data of other threads. Errors should be properly handled and debugging error messages should not be displayed to the user.

"Providing a set of secure coding standards does not guarantee those practices will always be implemented."

Security in implementation of purchased or acquired components focuses on implementing countermeasures and constraints to deal with known vulnerabilities in the individual components and their interfaces.

For custom-developed applications, coding standards are usually defined as part of the overall project effort. The project's coding standards should be augmented with extra standards for secure coding. For example, Sun has a set of security coding guidelines available [9] which would be effortless to integrate into the project's coding standards document. It cannot get much easier than this.

Providing a set of secure coding standards does not guarantee those practices will always be implemented. Therefore, validation of the source code is an essential activity. Much like peer review of source code, a security code review should be performed to assess the correctness and adequacy of the source code, but from a security standpoint. This technique is also referred to as *white-box testing*. The criteria selected to evaluate the source code should mirror the secure coding standards defined, or an agreed upon set of criteria based upon the level of risk an organization is willing to accept. Code analysis discovers subtle and elusive implementation errors before they reach testing or fielded system status. By correcting subtle errors in the code early, software development organizations can save engineering costs in testing and long-term maintenance [10]. Analysis can be performed manually or with automated tools. The Software Assurance Metrics And Tool Evaluation project at NIST attempts to classify software assurance tools and provides a comprehensive listing of code analysis tools [11].

Test and Deploy

Software security testing is not the same as traditional functional testing. The main objectives of software security testing are to identify vulnerabilities in the software and to ensure the secure behavior of software in the face of an attack.

Several techniques can be used for security testing, such as a vulnerability scan, penetration test, and security-oriented fault injection[2]. Vulnerability scans are performed with tools that attempt to detect application-level vulnerabilities (e.g., SQL injection, cross-site scripting) based on known attack patterns. Penetration testing attempts to break the application from the outside (the hacker's perspective). This can be accomplished manually by security specialists or in an automated fashion with tools such as brute-force attack tools.

What if a security code review has already been performed? Is it necessary to perform security testing since it is just another form of validation? Both validation techniques have their benefits and drawbacks. Security testing may simulate real-world attacks and exploitable vulnerabilities. Security testing may also be performed in conjunction with functional testing (assuming the capability is present). However, once vulnerabilities are found

during testing, there is often limited time available to correct the problem. An ideal validation approach would be to complement source code analysis (performed during build) with security testing. With this approach, a minimal number of security flaws can be expected to come out of testing, which allows adequate time to correct them. Of course, a more pragmatic approach would be dependent on the amount of assurance required from the system and a definition of the overall security goal. An organization may choose to perform risk-based security code analysis or testing, assessing limited portions of the application based on risk.

Findings from security testing should also be fed back through the change control process as would non-security findings during functional testing.

Operations and Maintenance

Before the application goes into production, a security risk analysis should be performed to ensure that the application (new or updated) does not introduce an unacceptable level of risk. Results of security testing should be fed into the risk analysis, given there are viable threats to the vulnerabilities identified during testing.

Periodic risk assessments should be performed as the threat environment constantly changes. New attacks can uncover previously unseen vulnerabilities. It is also important to conduct risk analysis whenever major changes to the system occur. In the federal government, findings are fed into a plan of actions and milestones, where the mitigation is tracked to closure [12].

Major changes also require validation – a security code review and test. A good approach is to perform security code reviews and then attempt to exploit the more severe findings. This gives organizations a feel for the reality of their findings.

From an operational standpoint, it is important to constantly monitor security alerts and advisories that pertain to the technology implemented by the software since it is not uncommon for systems to fall victim to zero-day attacks[3]. A time-saving approach could be to subscribe to security alerts or feeds from product vendors or cyber-security sites such as the U.S. CERT or the NVD. This eliminates the need and dependency for administrators to constantly check Web sites for security updates. The counterpart to receiving advisories is an approach to address them once they are received. Therefore, it is critical to have a vulnerability and patch management program in place so that corrective action is taken, i.e., patches and fixes are applied in a consistent, timely manner.

Disposal

Security incidents are often seen when equipment is repurposed or disposed without completely eliminating records from hard drives or other data storage devices. Hardware and software should be appropriately sanitized, especially if the equipment will be re-used or repurposed.

Conclusion

Many of the security activities described in this article are simply an expansion of the current activities to include security considerations. Security is not as difficult to integrate into the SDLC as it may appear, and it is vastly more effective than bolting it on at the end. Integrated security ensures that the security mechanisms are adequate and effective, something that bolt-on security cannot boast.

Traditional approaches to security such as firewalls, intrusion detection systems, or server hardening are still important elements of security, but they are in no way the silver bullet for software security. They cannot protect the application itself against compromise. This is quite a paradox considering the application is the most visible of all the aforementioned components. The addition of these activities ensures that adequate security is baked-in to the end product and not sprinkled on, providing the system with resilience against attacks.◆

References

1. Gartner, Inc. "Recommendations for Security Administration, 2006." Qwest Communications 12 Dec. 2006 <www.mediaproducts.gartner.com/gc/webletter/qwest/issue4/gartner2.html>.
2. Carnegie Mellon University. CERT/CC <www.cert.org>
3. Havenstein, Heather. "Baked-In Security." ComputerWorld 21 Mar. 2005.
4. NIST. "Federal Information Security Management Act." <www.csrc.nist.gov/policies/FISMA-final.pdf>.
5. NIST. Security Configuration Checklists Repository. <www.checklists.nist.gov/repository/index.html>.
6. NIST Publications. Computer Security Resource Center <www.csrc.nist.gov/publications/nistpubs>.
7. Viega, John. Building Secure Software. Addison-Wesley Professional, 2001.
8. OWASP. The Open Web Application Security Project <www.owasp.org>.
9. Sun Developer Network. "Secure Coding Guidelines." 2000 <www.java.sun.com/security/seccodeguide.html>.
10. Lavenhar, Steven. "Code Analysis." BuildSecurityIn Portal. National Cyber Security Division. 28 Jan. 2006 <www.buildsecurityin.us-cert.gov/daisy/ bsi/articles/best-practices/code/214.html?branch=1&language=1>.
11. NIST. "Tool Taxonomy." <www.samate.nist.gov/index.php/Tool_Taxonomy>.
12. Daniels, Mitchell E. Jr. "Memoranda 02-01." 17 Oct. 2001 <www.whitehouse.gov/omb/memoranda/m02-01.html>.

Notes

1. SQL injection is a type of security exploit in which the attacker adds SQL code to a Web form input box to gain access to resources or make changes to data.
2. Fault injection is a testing technique where the application is fed anomalous input to reveal behavior.
3. A zero-day attack is an exploit against a vulnerability the same day the vulnerability becomes generally known.

Background

1. Goertzel, Karen Mercedes, et al, Security in the Software Lifecycle: Making Software Development Processes – and the Software Produced by Them – More Secure, Draft Version 1.2 (Aug. 2006), U.S. Department of Homeland Security.

About the Author

Kwok H. Cheng is an associate at Booz Allen Hamilton where he currently assists organizations in implementing security at the application level. He has a background in systems engineering, process improvement, and information assurance. Cheng has a master's degree in information and telecommunication systems, a certificate in information security management, and is a certified information systems security professional.

Booz Allen Hamilton
8283 Greensboro DR
McLean, VA 22102
Phone: (703) 902-3060
E-mail: cheng_kwok@bah.com