An Ecosystem for Continuously Secure Application Software

Mark Merkow, CISSP, CISM, CSSLP, PayPal Inc.
Lakshmikanth Raghavan, CISM, CEH, CRISC, PayPal Inc.

Abstract: A software development ecosystem composed of nine working elements makes it possible to continuously secure application software throughout the entire Software Development Lifecycle (SDLC) and while it's in production use. By orchestrating the activity of these nine elements, organizations and their leadership can reliably and repeatedly produce high-quality software that can stand up to attacks or rapidly recover from intentional or unintentional malicious activity.

Introduction

Provably secure application software can only emerge from a SDLC that treats security as a core element of every phase and in post-deployment. By mandating security within the SDLC itself, management in organizations can rest better at night knowing their infrastructure is continuously working as their defender rather than their enemy. When you address software development as a completed system of phases, tools, activities, and feedback loops, you can bring to life The Rugged Software Manifesto [1] as your deeds match your words.

The U.S. Scheme for software assurance in government and military applications relies on the Common Criteria (CC) Evaluation and Validation Scheme, developed and operated by NIST and the NSA. Critics complain that the CC is too heavyweight and impractical, that it takes too much time, costs too much, and flies in the face of the powerful commercial forces of "time to market" [2].

While CC mechanisms and processes may not be terribly useful for in-house custom developed software applications, many of the concepts and features of the scheme most certainly are. By selectively picking and choosing those software assurance steps from the CC and leading practices in software security, it's possible to build out an infrastructure that produces provably secure application software and provides real-time feedback into the system that forces code with residual vulnerabilities back into the SDLC for rapid remediation and redeployment. A continuously secure ecosystem for software development enables organizations to pay closer attention to building innovative business features and less attention to process or "meta" issues that affect software security and quality.

Catching Errors Sooner Lowers Overall Costs

From the earliest days of software development, studies have shown that the cost of remediating vulnerabilities or flaws in design are far lower when they're caught and fixed during the early requirements/design phases rather than after launching the software into production. Barry Boehm blames late inspection for software errors as the cause of an increase of 40 to 100 times the cost that is required if the errors were caught sooner in the SDLC [3]. Therefore, the earlier you can integrate security processes into the development lifecycle, the cheaper software development becomes over the long haul.

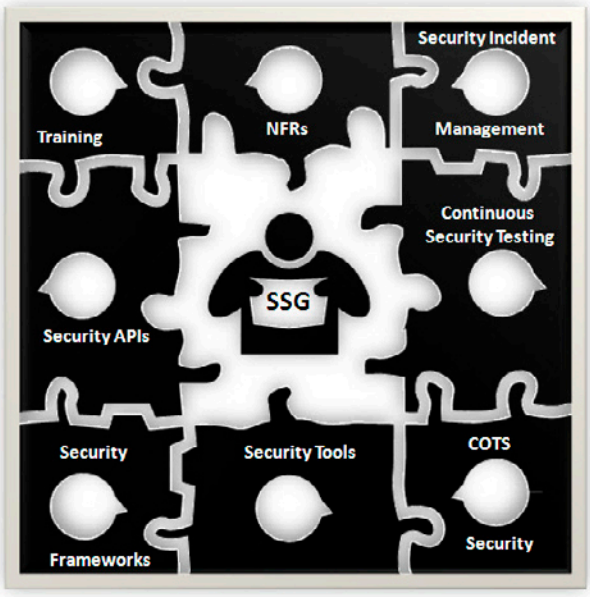
The tricky part is implementing a software development and operational infrastructure in a large enterprise while making it repeatable, scalable, and a natural part of the organization's DNA.

Building Blocks for Continuous Application Software Security

An ecosystem for continuously secure application software requires a robust and reliable infrastructure to make it work. The basic building blocks needed to bring such an infrastructure to life include the following: training and awareness; a Software Security Group (SSG); Nonfunctional Requirements (NFRs); reusable security Application Programming Interfaces (APIs); security frameworks; software security tools; security of COTS software; software security incident management; and continuous security testing.

Once assembled, the complete picture should appear like the one in the diagram below.

Figure 1: Building Blocks for Continuously Secure Application Software



The SSG plays a central role in the ecosystem, providing a crucial source for strategies, implementation, orchestration, and governance for the tools and processes needed to continuously improve the overall security of the software being developed [4].

1) Training and Awareness

While training may not fit directly into any particular SDLC phase, it plays a crucial role in improving the overall security and quality of software. Training is a prerequisite for anyone who has any role anywhere in the software development environment.

All developers and other technical members of the software design/development/test teams should undergo security training that explains the responsibilities of their role, establishes the expectations for their part for security, and provides best practices and guidance for developing and maintaining high-quality software.

2) Software Security Group

A formal SSG, having its primary responsibility be the improvement of security of the SDLC and the software it produces, is a core element. The role of the SSG includes the following: define the software security strategy; develop the processes for integrating security into all the phases of SDLC; roll out software security tools for developers and testers; establish security testing processes; oversee the development of security APIs and frameworks; develop and deliver software security awareness/training sessions; and define, track, and report the metrics related to the success and progress of the overall software security program.

3) NFRs

What software is expected to "do" is described by users in functional requirements. These requirements show up in the early development phases when a group of would-be users collect to describe what they want. NFRs are the quality, security, and resiliency aspects of software that only show up in requirements documents when they're deliberately added. These requirements are the outcome of software stakeholders who meet to discuss the planned software. These stakeholders should include the people who will use the system, the people who will operate it, the people who will maintain it, the people who will oversee the governance of the software development lifecycle, security professionals, and the legal and regulatory compliance groups who have a stake in assuring that the software is in compliance with local, state, and federal laws.

The key to a successful software security program is to establish a requirements analysis process within the SDLC that treats nonfunctional requirements as equal citizens to functional requirements. The SSG should establish processes to assure that this regularly occurs and help the application development organization to put together well defined and reusable NFRs. For example, if your organization follows the agile development methodology, "user stories" [5] are one method for requirements collection that you can quickly apply. Stakeholders could use these methods to capture nonfunctional requirements as well as functional requirements. Some of the key NFRs that must be considered include availability, capacity, efficiency, extensibility, interoperability, manageability, maintainability, performance, portability, privacy, recoverability, reliability, scalability, security, and serviceability.

4) Reusable Security APIs

Application developers have no business writing security functions. Using security controls is different from building them. A better bet is to build and promulgate standardized security APIs for developers to re-use and integrate into their applications. These APIs perform the most important security functions such as validation, encoding/decoding, cryptographic processes like encryption, hashing, authentication, authorization, logging, error handling, etc. The Open Web Application Security Project (OWASP) Enterprise Security API [6] is one such API that any organization can adopt and customize for its software development and operational processes. Developers will need education and training on using these security APIs and should be prevented from developing their own.

5) Security Frameworks

In addition to reusing security APIs in custom development work, security frameworks can help to automatically prevent many well-known attacks like Cross-site Scripting [7], Cross-site Request Forgery (CSRF) [8], and others. These frameworks are built by a centralized application security development under the guidance of the SSG and deployed to all production Web applications. These frameworks automatically provide security functions that counter well-known Web attacks. Many existing frameworks like Spring, Struts, etc., have some security built-in, but controls are frequently missing, incomplete, or wrong. After analyzing the gaps in each framework, some of the typical custom frameworks that can be built and deployed in any application infrastructure include: output encoding framework; input validation framework; and CSRF framework.

Whereas APIs need to be explicitly used by developers, these frameworks work invisibly and require no explicit action from the developers. If developers fail to output encode the parameters sent to an HTML page, these frameworks will automatically do it for them.

6) Software Security Tools

Static Analysis of source code and Dynamic Analysis of runtime modules provide significant value to any development environment that honors secure applications. Developers need regular access to static analysis tools for source code analysis and to report security vulnerabilities. Quality Assurance (QA) (testing) teams also require access to dynamic analysis tools (also called black-box testing tools) for complete code coverage. While we don't recommend any specific vendors, we do recommend that organizations perform an objective evaluation of all available tools and select the ones that would work well with their own environment and processes. Some existing reviews done by NIST SAMATE Project [9] and NAVSEA [10] can be a good starting point to begin the evaluation.

While there are multiple strategies to scan source code using any of the commercially available scanners (e.g., Ounce, Fortify, etc.), we strongly recommend a two-pronged approach to the deployment model: an Integrated Development Environment (IDE)-based version for developers to use at their desktops, and a build process integration for effective governance and management of the SDLC. With an integrated scan that runs automatically when an application is submitted for a QA Environment Build Operation, management can define gating criteria that routes the application to the appropriate channels based on the outcome of the source code scan.

IDE Integration for Developers:

To help developers scan the code they write early enough in the lifecycle, you need to provide them with unfettered access to automated scanning tool(s) so that they can perform scans themselves, via an IDE running at their desktop computer.

Scanning can be performed on a single function or method, on a file or a collection of source code files, or on the entire application system. This self-service scan will provide results that developers can use directly to clean up their code based on the findings. The scan report typically provides generic recommendations on how to fix the identified vulnerabilities as well. The OWASP Web Testing Environment (WTE) Project [11] is one example of testing tools for developers that aims to make application security tools and documentation readily available. The WTE has several other goals too, including to: provide a showcase for popular OWASP tools and documentation; provide the best, freely distributable application security tools in an easy-to-use package; ensure that the tools provided are as easy to use as possible; continue to add documentation and tools to the OWASP WTE; continue to document how to use the tools and how the tool modules were created; and align the tools provided with the OWASP Testing Guide.

Build Integration for Governance:

Build process-based scanning occurs when the entire application (all modules and libraries) are ready to be built and prepared for QA testing. This typically includes source code components originating from different development teams and/or different software development companies (e.g., outsourced development shops). This centralized scanning is meant as a governance and management mechanism and can be used to provide gating criteria before the code is released to the next phase in the SDLC. Typical gating criteria for production movement might include zero high-risk vulnerabilities; no more than five medium-risk vulnerabilities; no more than 10 low-risk vulnerabilities, etc.

A software supply-chain risk can be a defect in the delivered software or in the default installation or configuration that an attacker can exploit [12] and a build-governance software scan can help to uncover and eliminate errors that otherwise would fall through the cracks.

You should use the build process-based scanning not only for planned software releases but also for emergency bug fixes. Since the scanning process is closely integrated with the build process, automation takes care of assuring that source code scanning happens every time. When the assurance level of the automated scanner is high (not too many false positives), then the build server can be triggered to fail the build based on the gating criteria and send the application back for remediation.

Metrics that are useful to track for measuring performance and progress could include: number and percent of applications scanned using IDE scan; number and percent of applications scanned using Build scan; number and percent of applications scanned using Build scan that failed/passed; vulnerability density (vulnerabilities/thousand lines of code); vulnerability severity comparison across projects or development teams; vulnerability category comparison across projects or development teams; vulnerability category-specific trending; average time taken to close high/medium/low-risk vulnerabilities; vulnerability distribution by project; and top 10 vulnerabilities by severity and frequency.

Dynamic Analysis (Black-Box Testing) During QA Testing:

In addition to functional testing of an application in the QA environment, security testing using a black-box scanning tool (like IBM's AppScan) can help to catch any remaining vulnerabilities that fell through all prior safety nets.

A bug priority matrix for the organization should include the definitions of security defects that enable the QA team to create separate security defect records and help classify their priority. The testing carried out by this independent team might also serve as gating criteria for promoting the application from QA testing to the production environment. The results from these test results should be shared with the developers soon after the tests are run, so the developers can develop strategies for remediating the issues that are uncovered. Once the criteria for moving an application to production are met, the QA team should sign off on the security vulnerability testing, along with the other test results. Black-box testing also ensures that other minor feature additions and bug fixes are also tested for security bugs before they too are moved to production. Furthermore, centralized testing yields meaningful metrics as experience with the tools is gained and progress (or regress) can be measured and reported over time.

7) Security of COTS Software

When COTS software is used by custom-developed systems or offered as an infrastructure service, you may run into problems when you discover vulnerabilities during preproduction black-box testing and penetration testing. In most cases, when problems are found with COTS systems, it's difficult to identify what to do about them or even determine who to contact. As users of COTS products, information protection and risk management professionals are too far removed from the product development activities. Today's state of COTS security testing often leaves software buyers with little ability to gain the confidence they need to deploy business-critical software. In its absence we are forced to develop our own countermeasures and compensating controls to counter these unknown potential threats and undocumented features of the software. As we mentioned in the beginning of the article, because of any actual or perceived shortfalls of the CC, commercial businesses are forced into using various other and related approaches to gaining confidence in the security of COTS products. Here we take a look at two common commercial approaches.

ICSA Labs:

The ICSA Labs certification is based on public, objective criteria that yield a pass/fail result [13]. The criteria–drawn from expertise across the industry–are clearly defined and address common threats and vulnerabilities for each COTS product. The criteria are applicable among products of like kind and can therefore be used for better understanding, comparing, and assessing security products.

Veracode's VerAfied Software Assurance:

Delivered as a cloud-based service, Veracode's VerAfied process yields an analysis of final, integrated applications (binary analysis) and provides a simple way to implement security best practices and independently verify compliance with internal or regulatory standards without requiring any hardware or software [14].

8) Software Security Incident Management

The SSG must work closely with other stakeholders like the Security Operations/Monitoring team, Application Development teams, etc. to put together a well-defined application security incident management process. Even with controls throughout prior phases of the SDLC, bad code still manages to wind up in the production environment. A Security Incident Management Process is needed to analyze, triage, and apply short-term restoration fixes (such as application firewall rule changes), coordinate long-term code level changes, and validate and deploy security fixes.

In his September/October 2010 CrossTalk article, "The Balance of Secure Development and Secure Operations in the Software Security Equation," Sean Barnum advocates a holistic approach that balances secure software development and secure IT operations [15]. By including the appropriate participation from the development community, the incident management process acts as the final safety net to protect the organization from further damage once an incident is declared.

Some key success factors include the following: appropriate stakeholder access to reports from continuous security testing and monitoring tools; a unified bug tracking system that everyone uses and provides end-to-end tracking and closure of identified security bugs; and well-defined processes to identify appropriate source code owners to alert and engage them about production vulnerabilities and to help develop, test, and deploy security fixes.

9) Continuous Security Testing

The last piece of the puzzle completes the picture and pulls together all the elements that compose the ecosystem. Continuous testing using regularly updated black-box scanners set to run automatically can help to assure that new vulnerabilities, and those missed in prior development phases, are caught and acted upon before anyone on the outside has the opportunity to find them and exploit them. Scans can be scheduled based on any number of factors related to the application.

Continuous security testing in the QA Environment of production-released code, along with a well-defined feedback loop that relies on the software security incident management process, can alert the application owners about residual security problems so they can be addressed immediately. As security incidents are opened, the application is forced back into the analysis or design phase of the SDLC and works its way back through the SDLC, helping to assure that software is never released and forgotten.

Conclusion

With a completed puzzle of symbiotic and synergistic elements working in concert, you can implement a well-orchestrated, well-oiled feedback system that over time will improve the SDLC itself as experience is gained and processes and tools are fine-tuned. Meeting the pledge of The Rugged Software Manifesto includes improving the software development environment itself as you improve your own skills. By inculcating security activities and features into the entire SDLC and beyond, you can rest assured that you're doing all you can to address and reverse the scourge of insecure application software.

ABOUT THE AUTHORS

Mark Merkow, CISSP, CISM, CSSLP works at PayPal Inc. (an eBay company) in Scottsdale, AZ as a manager in the IT Security Department. He has over 35 years of experience in Information Technology from a variety of roles, including Applications Development, Systems Analysis and Design, Security Engineer, and Security Manager. Mr. Merkow holds a master's in Decision and Information Systems from ASU, a master's of Education in Distance Learning from ASU, and a bachelor's degree in Computer Information Systems from ASU. He chairs the Financial Services Information Sharing and Analysis Center Education Committee, serves on the BITS Security Working Group and the Research and Development Committee of the Financial Services Sector Coordinating Council on Homeland Security and Critical Infrastructure Protection. Mr. Merkow has authored or co-authored 10 books, including his latest, "Secure and Resilient Software Development" (2010, Auerbach Publications).

Mark Merkow
9999 North 90th Street
Scottsdale AZ 85258
E-mail: mmerkow@paypal.com
Phone: (480) 862-7391

Lakshmikanth Raghavan
2211 North 1st Street
San Jose CA 95131
Email: lraghavan@paypal.com
Phone: (408) 967-4637

Lakshmikanth Raghavan (Laksh) works at PayPal Inc. (an eBay company) in San Jose, CA as Staff Information Security Engineer in the Information Risk Management area. He has over nine years of experience in the areas of information security and information risk management and has been providing consulting services to financial services companies around the world in his previous engagements. He is a Certified Ethical Hacker and holds the CISM and CRISC certifications from the Information Systems Audit and Control Association. Laksh is the co-author of "Secure and Resilient Software Development" and holds a bachelor's degree in Electronics & Telecommunication Engineering from the University of Madras, India.

REFERENCES

1. Rugged Software. Web. 30 Sept. 2010. <http://www.ruggedsoftware.org>.
2. Merkow, Mark S., and Lakshmikanth Raghavan. Secure and Resilient Software Development. Boca Raton, FL: CRC, 2010. Print.
3. Boehm, Barry W., and Richard Turner. Balancing Agility and Discipline: a Guide for the Perplexed. Boston:MA, Addison-Wesley, 2006. Print.
4. The Building Security In Maturity Model (BSIMM). Web. 30 Sept. 2010. <http://bsimm2.com/index.php>.
5. Introduction to User Stories <http://www.agilemodeling.com/artifacts/userStory.htm>
6. "Category: Enterprise Security API." ESAPI. OWASP. Web. 30 Sept. 2010. <http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API>.
7. "Cross-site Scripting." Wikipedia, the Free Encyclopedia. Web. 30 Sept. 2010. <http://en.wikipedia.org/wiki/Cross-site_scripting>.
8. "Cross-site Request Forgery." Wikipedia, the Free Encyclopedia. Web. 30 Sept. 2010. <http://en.wikipedia.org/wiki/Cross-site_request_forgery>.
9. NIST SAMATE - Static Analysis Tool Exposition. Web. 11 Nov 2010. <http://samate.nist.gov/Main_Page.html> & <http://samate.nist.gov/SATE.html>
10. Software Security Assessment Tools Review. Web. 11 Nov 2010. <https://buildsecurityin.us-cert.gov/swa/downloads/NAVSEA-Tools-Paper-2009-03-02.pdf>.
11. OWASP Web Testing Environment (WTE). Web. 15 Nov 2010. <http://code.google.com/p/owasp-wte>.
12. Ellison, Bob. Supply-Chain Risk Management: Incorporating Security into Software Development. 03 2010. DHS National Cyber Security Division. 15 Nov. 2010 <https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/acquisition/1140-BSI.html>.
13. ICSA Labs. ICSA Labs. Web. 30 Sept. 2010. <https://www.icsalabs.com>.
14. Application Security | Veracode. Veracode. Web. 30 Sept. 2010. <http://www.veracode.com>.
15. Barnum, Sean. The Balance of Secure Development and Secure Operations in the Software Security Equation. Crosstalk – The Journal of Defense Software Engineering. Vol 23 Number 5. Sept/Octorber 2010.