Feature

# Preventive Technical Controls for Application Security

**Rohit Sethi, CISSP, CSSLP,** is vice president of product development at SD Elements. Sethi is a specialist in building security controls into the software development life cycle (SDLC), and a SANS course developer and instructor on Secure J2EE development.

**Ehsan Foroughi, CISM, CISSP,** is director of research at SD Elements. Foroughi is an application security expert with eight-plus years of management and technical experience in security research and an extensive development and reverse engineering background. He is the founder and chief technology officer (CTO) of TELTUB.

COBIT®,[1] the Payment Card Industry Data Security Standard (PCI DSS) and several other software security maturity models[2, 3, 4] address the security requirements necessary to produce secure applications. In the authors' experience, software development groups rarely address technical application security requirements at the breadth or level of detail found in The Open Web Application Security Project (OWASP)[5] or similar developer guides to security.

Organizational application security efforts tend to focus on automated detective and/or corrective solutions, such as static analysis, binary analysis, run-time testing and web application firewalls.[6] IT control audits that primarily assess detective application security controls tend to bias enterprises' application security efforts toward building software features and subsequently fixing security defects rather than preventing the defects from occurring in the first place.

> "Secure application life cycle management (SALM) systems can help auditors assess organizations for preventive technical security requirements of either procured or in-house developed applications."

An emerging class of in-house and commercial off-the-shelf (COTS) software called secure application life cycle management (SALM) systems can help auditors assess organizations for preventive technical security requirements of either procured or in-house developed applications.

### CHALLENGES WITH DETECTIVE CONTROLS

Detective controls face challenges. Static and/or dynamic analysis systems cannot report on the *absence* of a technical security control mechanism, such as session management, altogether.[7] In practice, this sort of logic flaw is caught by manual source-code review or manual penetration testing. These techniques suffer from scalability and high cost. Few organizations can afford to perform the level of manual testing required for their entire application portfolio. The problem is further exacerbated by domain-specific flaws or bugs such as insufficient authorization, which require not only human expertise, but also an understanding of the software domain to uncover.[8] In one famous example, security researchers without sufficient permission were able to access privacy-protected photographs on Facebook.[9] While the security community and security tool developers already have a strong understanding of insufficient authorization, there is simply no practical method of detecting such a vulnerability using a completely automated mechanism.

### DETECTION IS INEFFICIENT

Detective techniques for flaws are inefficient when compared to preventive techniques as a result of extremely high remediation costs in the software development life cycle (SDLC). Researchers have long studied the sheer cost-effectiveness of planning for and preventing defects upfront rather than finding and fixing them later.[10]

### DEVELOPER EDUCATION

Another preventive technique—developer education—seeks to empower developers with the knowledge to write secure code. Research shows a noticeable positive correlation between education and the quality of application security.[11] Continuous training is crucial. With a single training class as a point-in-time activity, the value of the education diminishes over time unless the developers are continuously in touch with material and are updated on new and emerging techniques. Moreover, given the pressures of building software under strict deadlines, software developers could forget about specific security defects due to cognitive burden.[12] Thus, developer education is important but not sufficient for preventing application security defects.

### UNDERSTANDING SALM

SALM systems seek to close the gaps in the current detection-focused software security space. SALM systems are the security extension of SALM products—tools designed to help manage the process of building software.[13] SALM systems define specific application security defects and their corresponding preventive controls, as relevant to a given application, by rules relating to the application's underlying properties, such as class of application (e.g., web vs. client/server), technology stack (e.g., Java EE, C/C++, Android SDK) and regulatory drivers (e.g., PCI DSS). For example, a SALM system might define a rule that a Structured Query Language (SQL) injection[14] weakness applies to all applications that interact with databases using SQL. SALM systems are produced in-house by some of the most mature application security organizations in security-sensitive industries, such as software development, financial services and e-commerce. Commercial solutions also exist.

> "SALM systems seek to close the gaps in the current detection-focused software security space."

SALM content providers continuously update the database of common software security flaws and corresponding compensating controls tied to rules. Developers or security analysts can, thus, model an application by profiling its technology stack, compliance requirements and other properties. Based on the application's profile, SALM tools generate a series of checklists of preventive controls and corresponding guidelines to follow in various phases of the SDLC.

Each checklist item specifies an underlying security weakness, a succinct discussion of the control and a contextually tailored guideline based on the technology stack specified in the application priorities. With this structure, users can achieve three pillars of application life-cycle management: contextual training, project and progress tracking, and auditability.

### CONTEXTUAL TRAINING

By providing contextually relevant tasks, SALM systems reinforce one of the most effective application security controls—developer awareness training—while reducing the cognitive burden of remembering all relevant security issues.



The training is contextually relevant, thereby increasing its effectiveness.[15] The SALM system may also provide examples of known good source code in the developer's programming language. Moreover, by providing instructional videos on how to test for defects, SALM systems provide contextual training on how an attacker may exploit an underlying weakness.

### PROGRESS TRACKING AND AUDITABILITY

SALM solutions provide auditability into application security posture. Currently, many organizations assess application security posture by manual risk assessments[16] or by testing for security through dynamic and static analysis. Consistent risk assessments can often be difficult to deploy for information security. One practitioner suggests: "Security is more like art, and a security risk really cannot be calculated."[17] Approaches such as penetration testing also have limitations such as cost.[18] SALM solutions detail lists of controls for known software security weaknesses and track completion status. As a result, security auditors can quickly ascertain if an in-house, outsourced or COTS application has appropriately handled security controls by profiling the application in a SALM solution and tracking which security controls are completed.

Knowing security controls upfront allows development teams to build cost estimates and prioritize security issues alongside other priorities at project or iteration inception. Application owners can decide to accept risks at the planning stage. Upfront discussion and risk acceptance have the benefit of side-stepping disagreements later in a development cycle and avoiding a culture of development vs. security.[19]

## CONCLUSIONS

SALM solutions offer the unprecedented ability to achieve auditable and scalable prevention-based application security. Although detection-based controls such as static and dynamic analysis are still critical components of an overall secure SDLC, early evidence shows a clear business case for adopting a SALM solution.

The fact that enterprises in disparate industries have independently developed SALM systems in-house points to a pervasive need. Combining this with the ability to provide visibility into application security risk posture across an enterprise, SALM solutions are indispensable for reducing the risk of common weaknesses of software developed or purchased. IT controls auditors should consider the existence of SALM systems when auditing the effectiveness of an enterprise's application security posture.

## ENDNOTES


1 IT Governance Institute, COBIT 4.1, AI2.4 Application Security and Availability, 2007, *www.isaca.org/Knowledge-Center/cobit/Documents/CobiT_4.1.pdf*

2 BSIMM3, "Intelligence: Standards and Requirements (SR)," *www.bsimm.com/online/intelligence/sr/*

3 Chandra, Pravir; "Software Assurance Maturity Model: A Guide to Security Requirements," *www.opensamm.org/downloads/SAMM-1.0.pdf*

4 Microsoft, Microsoft Security Development Lifecycle (SDL) Process: Requirements, *www.microsoft.com/security/sdl/discover/requirements.aspx*

5 The Open Web Application Security Project (OWASP), *A Guide to Building Secure Web Applications and Web Services, 2.0 Black Hat Edition*, 27 July 2005, *http://sourceforge.net/projects/owasp/files/Guide/2.0.1/OWASPGuide2.0.1.pdf/download*

6 The 451 Group, "The Application Security Spectrum: From Concept to Cloud," *www.the451group.com/reports/executive_summary.php?id=1832*

7 OWASP-ASVS, "V3 - Session Management Verification Requirements," February 2012, *http://code.google.com/p/owasp-asvs/wiki/Verification_V3*

8 Sethi, Rohit; Yuk Fai Chan; "Domain-Driven Security," OpenSAMM, January 2011, *www.opensamm.org/2011/01/domain-driven-security/*

9 Grubb, Ben; "Security Experts Go to War: Wife Targeted," *Sydney Morning Herald*, 17 May 2011, *www.smh.com.au/technology/security/security-experts-go-to-war-wife-targeted-20110517-1eqsm.html*

10 LKP Consulting Group, "The Real Cost of Software Defects," *www.lkpgroup.com/Cost%20of%20Software%20Defects.pdf*

11 Veracode, *State of Software Security Report*, vol. 4, p. 6

12 Jing, Xie; Bill Chu; Heather Lipford; "Idea: Interactive Support for Secure Software Development," In Proceedings of Engineering Secure Software and Systems 3rd International Symposium (ESSoS), Madrid, Spain, February 2011

13 IBM, "Application Lifecycle Management: Effective ALM Helps Capital District Physicians' Health Plan Immediately Increase Efficiency," 2008, *www-142.ibm.com/software/products/us/en/category/SW860*

14 The Open Web Application Security Project (OWASP), "SQL Injection," 2012, *www.owasp.org/index.php/SQL_Injection*

15 Jing, Xie J.; B. Chu; Lipford H. Richter; "Why Do Programmers Make Security Errors?," Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 18–22 September 2011, Pittsburgh, Pennsylvania, USA

16 Stoneburner Gary; Alice Goguen; Alexis Feringa; SP 800-30 *Risk Management Guide for Information Technology Systems*, National Institute of Standards and Technology (NIST), July 2012, p. 8, *http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf*

17 Faessler, Mike; "Improving Security Risk Management," Web Wire, June 2011, *www.webwire.com/ViewPressRel.asp?aId=138969*

18 Wai, Chan Tuck; "Conducting a Penetration Test on an Organization," SANS Institute, 2002, *www.sans.org/reading_room/whitepapers/auditing/conducting-penetration-test-organization_67*

19 Wilander, John; "Security People vs. Developers," February 2011, *http://appsandsecurity.blogspot.com/2011/02/security-people-vs-developers.html*