Introducing Secure Application Lifecycle Management

A Whitepaper
Proactive Approach

Secure Application Lifecycle Management enables building in security from the start.
The Current State of Defensive Controls

The application security product marketplace offers automated mechanisms to detect and block software security flaws, such as static analysis, binary analysis, runtime testing, and web application firewalls\(^1\). For example, dynamic testing tools can easily discover if a standard Session cookie has the “secure” flag set to true\(^2\). Similarly, a static analysis tool can verify whether or not the session cookie has been configured to secure based on the framework in question\(^3\).

Dynamic Tools and Logic Flaws

Automated dynamic and static analysis both break down when developers choose to implement a proprietary session management system. Perhaps more importantly, these systems cannot report on the absence of a session management mechanism altogether. In practice, this sort of logic flaw is caught by manual source code review or manual penetration testing, human-guided or exploited in the wild. These techniques suffer from scalability. Few organizations can afford to perform the level of manual testing required for their entire application portfolio. The problem is further exasperated by domain-specific flaws or bugs such as insufficient authorization, which require not only human expertise but an understanding of the software domain to uncover\(^iv\). In one famous example, security researchers were able to access “privacy-protected” photographs on Facebook without sufficient permission.\(^v\) 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 suffer from being inefficient versus preventative techniques. In software development, researchers have long studied the sheer cost-effectiveness of planning for and preventing defects up-front rather than finding and fixing them\(^vi\).

![Figure 1: Cost of fixing defects in different phases of software development](image-url)

*Source: NIST*
Threat Modeling
Practitioners sometimes turn to design or architecture review techniques, such as threat modeling\textsuperscript{vii}, for preventing software security defects. Some industry practitioners consider threat modeling to be a cost efficient method of security defect prevention\textsuperscript{viii}. This approach suffers even more from the issue of scalability. Practitioners report that threat modeling without the involvement of qualified security experts can be less effective\textsuperscript{ix}, even though there is a shortage of such expertise\textsuperscript{x}.

Developer Education
Another preventative technique, developer education, seeks to empower developers with the knowledge to write secure code. Research shows a positive correlation between education and application security quality\textsuperscript{xi}. A single training class is a point-in-time activity; the value of the education will diminish over time unless the developers are continuously in touch with material and are updated on new / emerging techniques. Moreover, given the pressures of building software under strict deadlines, software developers can forget about specific security defect due to cognitive burden\textsuperscript{xii}. Thus, developer education is important but not sufficient for preventing application security defects.

Secure Requirements
Industry experts assert that building security in requirements is one of the best mechanisms for preventing security issues\textsuperscript{xiii \textsuperscript{xiv}}. Few resources offer comprehensive requirements on how to deal with the range of issues defined in the Common Weakness Enumeration. Moreover, empirical data suggests that requirement gathering techniques vary wildly between and even within organizations. Some organizations have built secure programming guideline documents to address preventative software security\textsuperscript{xv}. In the experience of Security Compass consultants, large static documents prove ineffective for use in day-to-day development under time pressure.

Introducing Secure Application Lifecycle Management
Secure Application Lifecycle Management (SALM) systems seek to close the gaps in the current detection-focused software security product market. SALM systems are the security extension of Application Lifecycle Management products: tools designed to help manage the process of building software\textsuperscript{xvi}. SALM systems defines specific application security defects and their corresponding preventative 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 “SQL Injection”\textsuperscript{xvii} and its corresponding defensive control “bind variables in SQL Statements” are relevant to any application that interacts with a database using SQL.
Researchers at SALM vendors 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 series of checklists of preventive controls in various phases of the Software Development Life Cycle (SDLC).

Each ‘Task’ specifies the underlying security weakness ('Problem') and a succinct discussion of the control 'Solution'. By providing contextually relevant Tasks, SALM systems re-enforce one of the most effective of 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. Where possible, the SALM system provides examples of known-good source code in the developer’s programming language. Armed with actual solution code, junior developers do not have to resort to source code examples posted on the Internet from unverified sources. Moreover, by providing instructional videos on how to test for defects, the SALM system provides contextual training of how an attacker may exploit an underlying weakness.

The Power of Checklists

SALM systems can also benefit security-savvy developers. By providing a reliable rules engine for tasks, SALM-produced checklists serve as essential reminders for tasks that knowledgeable developers may forget under the pressure of day-to-day development work. In his
seminal book *The Checklist Manifesto*, Dr. Atul Gawande studies fields such as piloting to discover why some professions have low defect rates with respect to human safety\textsuperscript{xix}. He discovers that checklists are a critical component, and armed with this knowledge his team of researchers pilot the World Health Organization (WHO) safety checklist which results in a 47% drop in deaths arising from surgical complications\textsuperscript{x}. In both surgery and piloting, practitioners initially resisted the idea of using checklists because checklists seemed too simple to be effective in helping complex domains where practitioners often pride themselves on mastery\textsuperscript{xx}. Nonetheless, a wealth of data continues to support the effectiveness of simple checklists in reducing preventable defects. The challenge in the software security domain is that checklists are often too long and complex because they cover a large range of known flaws such as the entire Common Weakness Enumeration (CWE), or too short and not context-relevant because they cover a small subset of defects such as Open Web Application Security Project (OWASP) Top 10. Some IT practitioners have grown weary of a “checklist approach” to security based compliance-oriented, inflexible policy audited through checklists\textsuperscript{xxii}. SALM systems solve this by using rules and profiling to deliver context-relevant defects and controls, and by providing a mechanism to triage the defects through priority scores. Thus, developers who are pressed for time can ensure they address the highest priority software security issues relevant to their application. Empirical data suggests that as with other domains like medicine, seasoned practitioners may be skeptical about the benefits of SALM checklists until they experience modeling an application and examine the resultant checklists. One of the core developers of SD Elements, the market leader in SALM, was himself initially skeptical of SALM until he saw the results:

"When I first saw it, I couldn’t see myself using the product. After I modeled a real application I was working on, I immediately recognized that it caught a number of security issues I was either unaware of had completely forgotten about."

An architect at a system integration company and early adopter of SALM technology articulated his experience:

“If our experience every dollar spent on identifying robust security requirements early on in the development life cycle has translated into significant cost savings as well as increased productivity in the testing and deployment of our projects.”

— Architect of System Integration Company. Early adopter of SALM.

Industry is just starting to collect relevant data from SALM systems. One health insurance company used SALM requirements up-front for a new internally-developed application and received a 99% security quality score from a popular binary analysis solution upon first submission, where only 17% of internally developed applications even receive an “acceptable” score upon first submission\textsuperscript{xxiii}.
Security Visibility

SALM solutions also provide visibility into application security posture. Currently, many organizations assess application security posture by manual risk assessments\textsuperscript{xxiv} or 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 security risks really can’t be calculated"\textsuperscript{xxv}. Approaches such as penetration testing also have limitations\textsuperscript{xxvi}, including cost. SALM solutions detail lists of controls for known software security weaknesses and track completion status. Thus, security auditors can quickly ascertain if an in-house, outsourced or Commercial Off The Shelf (COTS) application has appropriately handled security controls by profiling the application in a SALM solution and tracking which security controls are completed.

Early testing indicates that a comprehensive review of an application using a SALM approach lasts approximately four hours for a security analyst, including application profiling, interviewing developers/architects, and following-up on outstanding unknown areas. Using this technique, security analysts can maintain an enterprise-wide view of application security posture.

![Figure 6: Progress of security tasks](Image)

![Figure 7: Enterprise dashboard](Image)

Higher-risk applications can then undergo more extensive, testing-based analysis such as penetration tests. Security analysts can also use this global view of risk to determine which applications need more help from security experts.

Copyright 2012. SD Elements. 8
Knowing security controls up-front 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. Up-front discussion and risk acceptance have the benefit of side-stepping arguments later in a development cycle and avoiding a culture of “development vs. security people”\textsuperscript{xvii}.

Conclusions

SALM solutions offer unprecedented ability to achieve scalable, prevention-based application security. Although detection based controls such as static & dynamic analysis are still critical components of an overall secure SDLC, early evidence shows a clear business case for adopting a SALM solution. Data from other domains also indicate that effective, check-list based preventative controls generated by SALM tools can have a dramatic effect on defect reduction. Combined 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 for in-house developed and purchased software.
About SD Elements

SD Elements is the market leader in Secure Application Lifecycle Management. Some of the world’s most security sensitive organizations in software development, financial services, healthcare, and public sector use SD Elements to bring secure applications to market faster.

Learn More.

Reach out to us to hear more.

Christopher Faciana
Director of Sales, Western Region
cfaciana@sdelements.com
602-501-5249

Kevin Wirvin
Director of Sales, Eastern Region
kwirvin@sdelements.com
732-284-8635


Gawande, A. The Checklist Manifesto, Metropolitan Books. Pg. 154

Gawande, A. The Checklist Manifesto, Metropolitan Books. Pg. 35, 157


http://www.sans.org/reading_room/whitepapers/auditing/conducting-penetration-test-organization_67