Automated Scaling of Security Requirements

A Whitepaper
Background

There is a strong bias in industry towards using automated testing solutions as a primary approach for application security. While other approaches to application security such as threat modeling and architectural analysis exist, most other mechanisms require application security expertise, and that expertise can be too expensive to scale. Unfortunately, a find-and-patch approach results in significant expense since it requires building and then refactoring code. One estimate places the cost of finding and fixing a single vulnerability in production at $5,000. Moreover, some well-known software risk analysis methodologies require heavy up-front investment in time, which makes them incompatible with agile development. One of the most promising techniques - training - is useful, but not sufficient to prevent vulnerabilities. The cognitive burden of retaining large amounts of training data makes it difficult for developers to recall all training in the context of day-to-day development.

One of the byproducts of relying primarily on testing for security is that development teams often imply rather than explicitly state security requirements. The net result is that when security assessments uncover vulnerabilities, development, security, and business representatives may not fix a security issue due to time-to-market pressures. If the security requirements were explicitly stated and planned for at the inception of a project, then stakeholders could agree on a level of security commensurate with the organization’s budget, timelines, and appetite for risk.

Fundamentally, secure applications begin with security requirements. If developers have secure requirements, they can incorporate the requirements into software design and code. They can also reason about which requirements are worth incorporating in a time-constrained environment. This is particularly important for iterative and agile environments.
Current Approaches to Non-Functional Requirements

A primary challenge is that many common development methodologies, such as Scrum, provide only minimal guidance on how to integrate security and other non-functional requirements (NFRs). One study found that software developers rarely have a formal process for integrating NFRs into their development work flow.

Security, as well as other NFR domains such as privacy, accessibility, internationalization, license compliance, other regulatory compliance, and adherence to corporate standards/policies are largely repeatable. They require distilling knowledge from a large body of knowledge based on a particular set of application properties. For example, providing users an ability to explicitly opt-out of tracking cookies is mandatory for any web-site that is subject to the laws of the European Union as per the European Data Privacy Directive.

Some security teams have attempted to call upon the repeatable property of security requirements by building secure coding guidelines. Unfortunately, time-to-market pressures mean developers rarely have time to read through comprehensive guidance documents in the midst of a development cycle.

Challenge with the Status Quo

Security and privacy requirements, along with other NFRs, are almost always implicit. Business analysts and product owners tend to focus on features that deliver tangible value to their customers. Developers generally learn about security issues, then write code, and then test or validate for security defects - often exclusively using automated tools. When the security test finds a defect, development teams make a choice of whether or not it’s worth fixing based on its potential for a breach. In practice, development and security teams often have differing opinions on whether or not a security issue is worth fixing.

Perhaps more importantly, automated scanning technologies alone - without customization - can only capture a fraction of all security requirements. We compared common software security weaknesses from the Common Weakness Enumeration and Security Compass testing results to the coverage of automated scanning tools. Our analysis found that the most comprehensive individual tools capture a maximum of 44% of security weaknesses in supported languages and frameworks. In other words, development teams...
could rely on a single automated tool to confirm the absence of about less than half of all security weaknesses. Security analysts could not rely on the tool at all for the remaining 56% without custom rules, manual code review and manual runtime testing.

Many of the high risk issues identified that either static or dynamic testing tools could not reliably test for include: certain authentication and password practices; authorization (which requires domain knowledge to assert); certain session management issues; many compliance issues such as treatment of confidential data for the Payment Card Data Security Standard (PCI DSS) and the Health Information Privacy and Accountability Act (HIPAA); storing and caching other kinds of confidential data; and privacy issues. Several of these issues have resulted in, or were contributors to, high profile breaches, or would result in substantial fines by regulatory bodies. Combining best-of-breed static and dynamic testing tools increases the limit of potential automated coverage to 56%. This still leaves application open to substantial risk. Properly configured static analysis and dynamic testing tools are still indispensable to application security. They are necessary, but not sufficient.

Automated Generation of Non-Functional Requirements

Development teams can effectively integrate security and privacy requirements, along with other repeatable NFR domains, into requirements gathering. We constructed SD Elements...
to automate this process and integrate scalable security requirements into development workflows.

The process flow works as follows:

1. Define basic properties of the software you are building through a survey.

2. Review all the requirements in a centralized database against the properties selected in Step 1 using Boolean logic rules.
3. Generate a set of relevant security requirements with links to corresponding test cases.

Figure 4: Generated set of contextually relevant requirements for security and privacy

By creating a centralized knowledge-base, practitioners can regularly test the applicability of requirements against real-world software and refine the rules to increase their accuracy. Reviewing one company’s data, SD Elements successfully would have predicted 97% of high or critical security vulnerabilities with prescriptive requirements across three years of penetration tests for web and mobile applications.

One of the common challenges with effectively deploying security and other NFRs is that they frequently appear in a different tool or repository than functional requirements. For example, security teams may ask developers to read a static programming guide or Excel checklist to uncover security requirements. With a tool-based solution, developers can consume repeatable non-functional requirements using the same Application Lifecycle Management (ALM) tools that they use for functional requirements tracking. For example, developers can automatically export SD Elements security & privacy requirements into Thoughtwork’s Mingle.
Validation

Validation is a necessary step for requirements management. While many organizations already employ some kind of security testing, these tests are rarely linked to requirements. Using the scalable approach to requirements, organizations can also provide test cases that link to the requirements - including education on how to perform the tests.
Moreover, if a development team already employs static and/or dynamic testing, they can concentrate on verifying those requirements that cannot be verified by their scanning tools.
Tracking non-functional requirements inside a dedicated tool also enables many kinds of reporting. For example, development teams can generate reports for compliance showing that developers built and testers tested for relevant requirements.

![Figure 9: Example of a test case with embedded instructional video](image)

**Conclusion**

Relying primarily on testing for application security and other NFR domains is error-prone and cost inefficient. Fortunately, a process and tool for scaling NFRs exists. Using this process leads to a shared understanding of acceptable security risk up-front with project team stakeholders. It also allows developers to consume NFRs in the same method as they consume their functional requirements. Tying these techniques with a combination of automated and manual validation techniques means greater confidence in building secure 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.

Reach out to us to hear more

web: www.sdelements.com
email: info@sdelements.com
Automated Scaling of Security Requirements


vii Leffingwell, Dean. Agile software requirements. Addison-Wesley Professional.


https://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf

http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/index.html

http://www.ynetnews.com/articles/0,7340,L-4279655,00.html

xv LeClaire, Jennifer. 6.5 million LinkedIn passwords may be in hands of hackers. Accessed November 27, 2012.

https://financial.ucsc.edu/Pages/Security_Penalties.aspx