ivision

July 14, 2023

Security Engineering

Version 1.3

https://ivision.com

Contents

Executive Summary: Why Security Engineering?

Why should teams care about security engineering?

Our premise for security engineering is simple, and it's backed up by our experience with multiple teams releasing high-quality software. Modern software teams often lack the quality, experience-driven guidance on how to implement security engineering for their software/product. DevOps organizations, in particular, aim to move towards a mature DevSecOps model. However, within the software industry, there is not a clear definition of what DevOps and DevSecOps even mean. Rather than attempting to define DevSecOps and justify its importance, this white paper outlines key security engineering practices we use successfully with our customers, many of whom follow a DevOps/DevSecOps model for managing their infrastructure and releasing their products.

Security defects in a product are really just a subset of software defects. Software with fewer defects will generally also have fewer security defects. Experienced product teams will inevitably be familiar with Boehm's[1] cost of change throughout the SDLC:

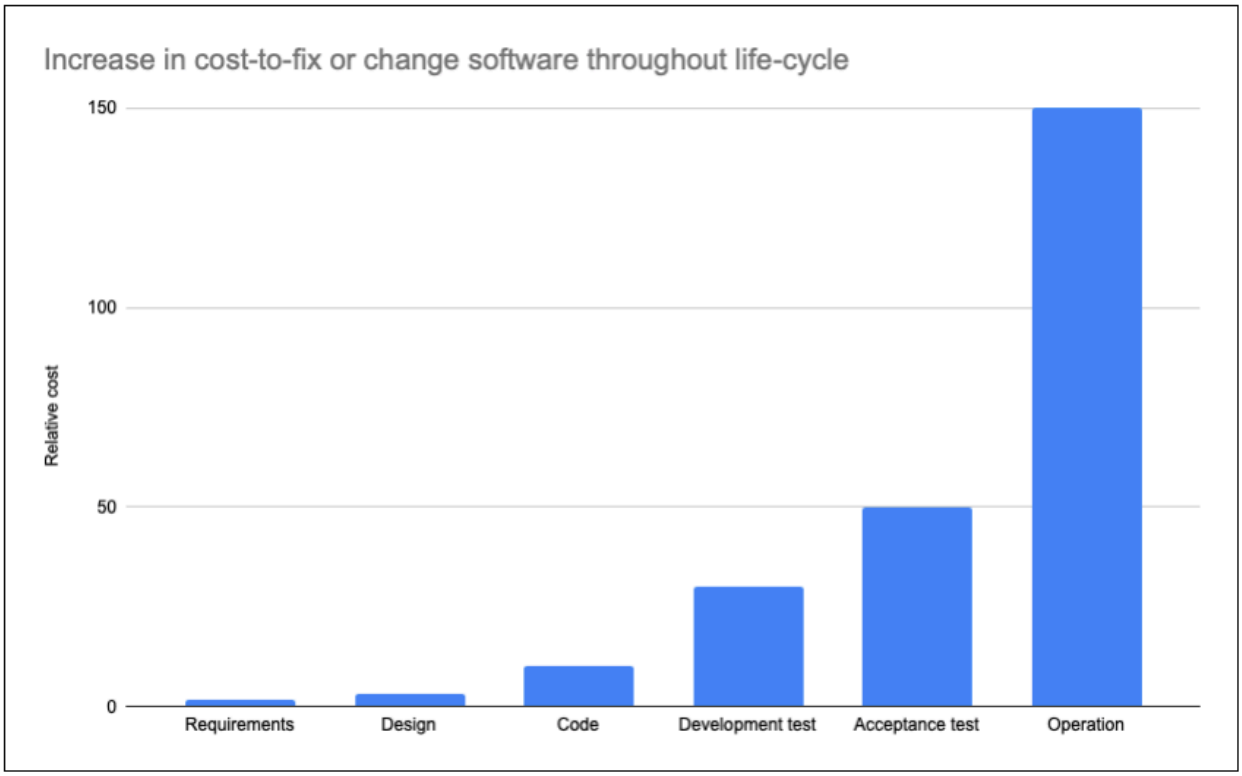

Figure # 1.1: Boehm relative cost to fix or change software throughout the lifecycle

[1]Boehm, B. (1981). Software Engineering Economics 1st. Prentice Hall PTR Upper Saddle River, NJ, USA ©1981.

Later studies generally substantiate this graphic[2][3][4]. Some refinements and key details should be considered, but, in short, it's highly beneficial to get things right up front. This doesn't mean that everything reverts to Requirement and Design-heavy Waterfall methodologies, but rather, to have a high-velocity software development lifecycle, we should aim to deploy software with as few defects and architectural flaws as possible, without falling into the trap of chasing perfection over progress. The goal of this paper is to discuss improving quality, specifically around security defects, without sacrificing feature release velocity.

Following from the fact that security defects are still essentially software defects, it becomes critical to treat these defects just like any other engineering problem. This often means altering an organization's culture around engineering, and security engineering, in particular. The rest of this paper provides details about ivision's approach to security engineering and implementing lasting change within product organizations. The key goal is to spend time and resources efficiently and outperform competitors in security. Implementing security engineering shifts product and software security from a fear and compliance-driven activity to being an integral part of software engineering.

[2]Card, D. N. (1987). A Software Technology Evaluation Program. Information And Software Technology, 29(6), 291–300.

[3]Jones, C. (1991). Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill.

[4]Boehm, B. W. (1987). Improving Software Productivity. IEEE Computer, 43–57.

Implementing Security Engineering Effectively

Security Engineering starts with the software development life cycle and how a software development organization approaches it. Many readers of this paper might be familiar with the phrase “Shift security left”. This phrase is often overused as a “catch-all” phrase for software security, but the general concept has been advocated by security practitioners for well over a decade. Microsoft was one of the first large organizations to prioritize security in the development lifecycle, and they documented their efforts and outcomes through various books, articles, and white papers. Focusing on the development lifecycle is a significant part of the process, but it’s often a challenge for organizations to transition from zero security lifecycle and security engineering towards a comprehensive and robust Secure Development Lifecycle (SDL). ivision’s approach helps bridge this gap by prioritizing activities and formulating a roadmap that brings forth positive security consequences without compromising release velocity.

A key aim in implementing this process is to spot where different types of security vulnerabilities arise and abolish them with security engineering and secure development methodologies. The goal is to instill a generative and constructive security culture within an organization, nullifying many of the traditional obstacles perceived by product teams concerning information security. The implementation of security engineering should work as an empowering tool for development teams rather than a drain on their time. Although this paper primarily focusses on SDL and security engineering, it’s crucial to keep business needs at the forefront throughout the application of these practices. The objective is to develop a collaborative security program that doesn’t set up an adversarial relationship between the security team and the engineers. If both teams work in sync and pinpoint security vulnerabilities earlier in the life cycle, issues become simpler to fix, mitigating the need to make tough decisions like delaying a release or disseminating software with known vulnerabilities.

The rest of this paper will delve into how to attain these outcomes by following a methodical process.

Achieving Security Engineering Outcomes

Start by defining the desired outcomes. The goal is to establish security outcomes for your product or software releases. Broadly, this maps to the following outcomes:

- Manage Security Risk
- Protect Personal Data from Breaches
- Detect Security Events
- Control and Minimize Impact (of security events)
- Recover from Security Incidents

After defining outcomes, our proposed mapping of activities onto the SDLC is as follows:

Figure # 3.1: SDLC Activities Mapping

We will review each of these activities in the following sections.

3.1 Defining Security Outcomes

We use two well-respected frameworks to measure and implement our security outcomes. The first goal for any software organization is understanding current best practices and how the organization adheres to those practices. BSIMM and the NIST Cybersecurity Framework are excellent options for conducting a basic risk assessment and gap analysis. Depending on your industry, other compliance regimes like PCI, GDPR and HIPAA may also be relevant.

BSIMM[1] (Building Security In Maturity Model) focuses on the software security aspect of building software. Although it has many key activities and a relatively simple self-assessment process, a key challenge with BSIMM is understanding which activities to prioritize post-assessment in order to efficiently use your resources. The NIST

[1]McGraw, G., Migues, S., & West, J. (2018). Building Security in Maturity Model (BSIMM) Version 9. Retrieved from https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf

Cybersecurity Framework[2] is a comprehensive cybersecurity framework including many activities found in BSIMM. By combining these two frameworks into a blended risk assessment and gap analysis, it provides good visibility into an organization's overall security posture and helps prioritize activities better.

The BSIMM Framework focuses on four key areas:

BSIMM			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy & Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards & Requirements	Security Testing	Coniguration Mnaagement & Vulnerability Management

Figure # 3.2: BSIMM Framework

The NIST Cybersecurity Framework breaks down into five key areas:

Figure # 3.3: NIST Cybersecurity Framework

ivision has created a blended cybersecurity assessment process that merges BSIMM and NIST while including elements such as an application inventory and data classification scheme. After conducting a high level assessment, most software organizations concentrate on BSIMM-focused activities and take responsibility for software security-oriented parts of the roadmap. With an operational focus, in a modern DevOps environment, NIST provides a more comprehensive view of the security posture space. This assessment process aims to create a roadmap teams can follow incorporating BSIMM and NIST Cybersecurity Framework activities and maturity goals.

All security engineering outcomes start with a good roadmap that considers an organization's priorities based on actual assessment and analysis. This roadmap and the process that creates it gives a foundation from which fully

[2]National Institute of Standards and Technology. (2018, April 16). Framework for Improving Critical Infrastructure Cybersecurity. Retrieved from https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

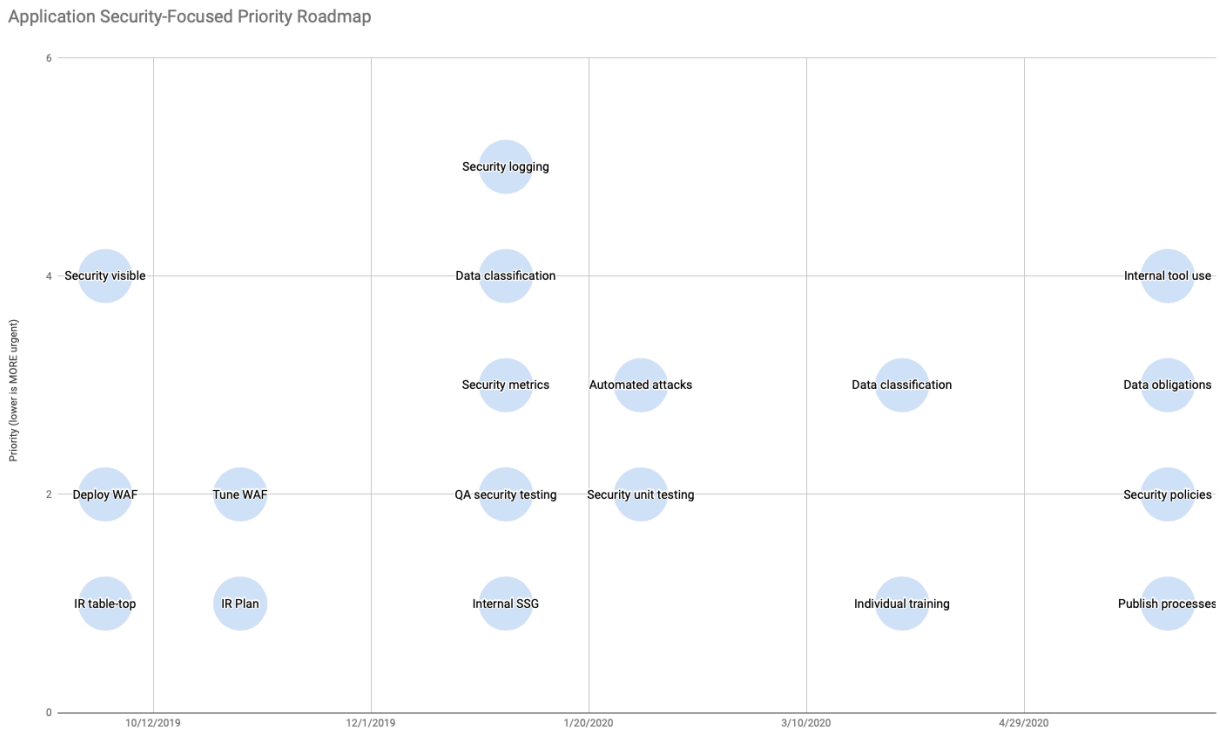

Figure # 3.4: Example Application Security Roadmap

defined security outcomes can be supported. From here, examine key activities and their impact on the overall security engineering processes and security outcomes for the organization. These will ultimately be driven by senior goals that the executive management team can easily understand. "Our senior goal is to prevent a data breach" or "Our senior goal is to protect our brand and image" are reasonable senior goals driven by worst-case scenarios for the organization. Keep senior goals in mind when prioritizing and planning during the assessment process.

3.2 Threat Modeling

Threat modeling answers the question, "what can go wrong" in a structured process. The style of threat modeling your organization uses must be custom-built for engineers to effectively enumerate threats in their system without requiring deep expertise on security issues or the threat modeling process itself. Threat modeling is based on the idea that engineers know their system best and can identify security flaws and implementation bugs using the right set of activities. This helps create a culture of security and trains engineers to think like both an attacker and a defender.

This section outlines the threat modeling process. For ivision, threat modeling is a foundational security engineering activity that greatly shifts security left within an organization. Security culture is discussed more in this paper, but**Security Champions** are trained in threat modeling early in the program. The Champions share this knowledge on how to conduct a threat model with their teams, and ensure that every iteration and software release is also performing threat modeling on features before they are implemented.

This section briefly explains threat modeling and its workings. It is not intended to be a comprehensive guide to threat modeling but seeks to portray the key components of the threat modeling process and their value. Threat modeling often begins with a good diagram; for developers and teams relatively new to threat modeling, we recommend using a simple and strict type of diagram known as a Data Flow Diagram (DFD). DFD identifies the system's components and allows a per-component analysis. For each component, the threat model aims to understand "what can go wrong?". For this, we recommend teams use the threat enumeration technique known as "STRIDE".

Threat	Desired Property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiability
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Application of stride is done per component in your system, represented on a DFD.

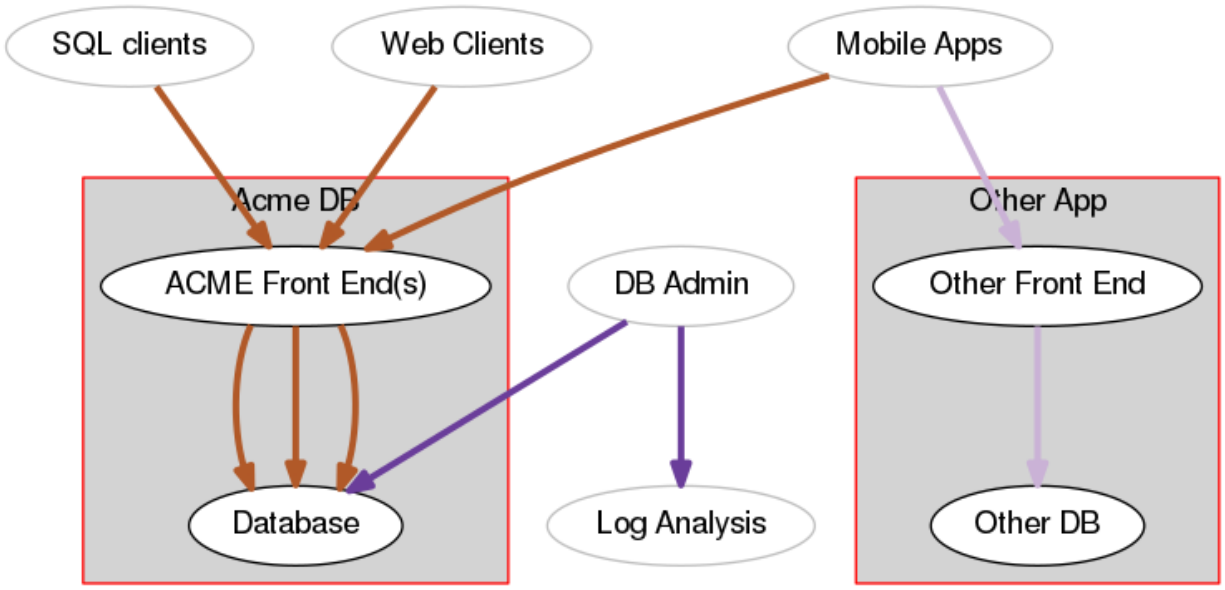

Figure # 3.5: Dataflow Diagram Example

Once you have listed potential threats, they can be triaged managed. A Threat Model is a living activity evolving with the software. Once identified, a threat is likely to remain a threat for the entire lifespan of the system. A vulnerability is a realized threat in a software system. Even if you address that vulnerability, the threat remains. For instance, a system that relies on a SQL based relational database will always have a risk for "SQL Injection" as long as the system uses a SQL-based RDBMS.

To illustrate some of the early concepts in designing a security program, we can use an analogy of building a house. The engineers are the builders. A specific threat could be a break-in and theft of your belongings. That threat will never disappear, although we can protect against the threat by locking doors, closing the garage door, and installing bars on windows. Here, the break-in threat is mitigated to an acceptable degree. However, if a vulnerability is introduced (like leaving the garage open at night), then a burglar could take advantage of this situation (break-in and steal your belongings).

For more details about threat modeling, refer to Threat Modeling[3]: Designing for Security (Shostack 2014)

3.2.1 Security Architecture Review

Security architecture review is a close relative of threat modeling, but it differs as it is not meant to a comprehensive review of a given system, but instead focus on making significant security design decisions about the architecture. These decisions can be influenced by the threat model itself.

For example, a common pattern observed when an organization is building microservices for the first time is to decentralize authentication and authorization and put the burden on each microservice to deal with it. The pur-

[3]Shostack, A. (2014). Threat Modeling: Designing for Security. Wiley.

pose of security architecture review is to critically examine these kinds of decisions early in the design process and understand the implications. In the provided example, this creates an "insecure by default" situation for each microservice, and developers must actively write code or configure their service to perform authentication and authorization. Security architecture review systematically handles this by understanding potential architectural flaws and bugs holistically and continually.

3.2.2 Learning Threat Modeling and Implementation Ideas

Threat modeling is something that practitioners often spend a significant amount of time gaining expertise in. But the 80% solution for development teams often involves learning just enough technique and mindset to effectively identify crucial issues. One of our favourite methods of training teams in threat modeling is the card game Elevation of Privilege[4]. Adam Shostack gave an excellent presentation at BlackHat in 2010 on how to play the EoP game[5][6].

Security practitioners often use the phrase "Think like an attacker." Although it seems like a good concept on the surface, it is actually a skill that many practitioners develop over time. Tools, such as games, can make threat modeling more relevant and collaborative whilst reducing the friction of the learning process. In our experience, the EoP game is well-received. It has been around for almost 10 years now and continues to serve as a good introduction to threat modeling concepts.

Continuing the analogy from above, it costs more to install bars on windows and a secure garage door opener after the house is built than building the windows a little higher and installing a secure garage door opener during the building process. Train engineers in threat modeling concepts so that it can be done in-band to reduce the risk of re-work later on due to a design flaw.

3.3 Security Culture and Active Support

Creating lasting change within an organization often involves changing fundamental behaviour within the organization. In this case, it's about establishing a culture of security and creating a 'generative' mindset in everyone who writes code. Even though BSIMM identifies many activities that can bring security into a software system, we have found that security culture in particular is a major part of any successful security engineering within an organization.

Security culture aims to mature how the organization approaches security and software development in general. To this end, creating a software security group that consists of key stakeholders and engineers is a powerful concept. This group is a resource to the organization with 'office hours' and a formalized role within the software development process. According to BSIMM[7], The Software Security Group (SSG) is:

"The internal group charged with carrying out and facilitating software security."

This is the first, and foundational, step in any software security initiative. Often, key stakeholders like architects and security champions are the first members of an SSG. Security champions are usually developers assigned to manage security for their team. The next section examines security champions and their pivotal role in cultivating a culture of security at organizations.

An SSG should hold regular 'office hours' to support engineers and conduct simple internal marketing. For instance, giving unique shirts to engineers who interact with the SSG or offering snacks at office hours can have a positive impact on engagement within the organization.

3.3.1 Security Champions

Security champions are a crucial part of creating a security culture within an organization. Below is a summary of the key features of a security champion:

[4]Microsoft. (n.d.). Elevation of Privilege Download. Retrieved from https://www.microsoft.com/en-us/download/details.aspx?id=20303

[5]Shostack, A. (2010). Elevation of Privilege Game. Retrieved from https://adam.shostack.org/Elevation-of-Privilege-BlackHat2010ShostackFinal.pptx

[6]Shostack, A. (2010). Black Hat USA 2010: Elevation of Privilege: The Easy way to Threat Model. Retrieved from https://www.youtube.com/watch?v=gZh5acJuNVg

[7]McGraw, G., Migues, S., & West, J. (2018). Building Security in Maturity Model (BSIMM) Version 9. Retrieved from https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf

- Deeper security knowledge
- Ownership of security issues
  - Advocacy for ensuring security issues are tracked and fixed
- Security leadership
- Visibility of Security
- Promoting a generative security mindset

Security leadership includes activities like leading threat models and managing security-specific code and features in a code base. Security champions are also part of the Software Security Group (SSG). Having visible security champions and an SSG helps make security a more visible and approachable part of an organization's culture. It also reduces the separation between security and engineering. By performing threat models, security issues are often fixed at design time and not later in the lifecycle when it could be more painful to remedy security issues.

Security champions are expected to spend around 10-30% of their time learning, implementing light-weight threat modeling, identifying security-relevant features, and noting security issues during planning so they can be addressed more cheaply. Initial training for champions should include threat modeling and light OWASP Top 10 training (if applicable), providing Champions with a basic and common vocabulary for discussing security-relevant issues.

Ideally, security champions are volunteer engineers with an interest in security. However, management buy-in is vital, as it will require a portion of the engineer's time initially. We argue that this time is repaid later when issues can be identified earlier in the SDLC, less time is spent on unplanned security work, and generative security teams build guides and code snippets that can be reused across teams, saving time.

3.4 DevSecOps and Security Automation

Incorporating security into a DevOps pipeline is still largely aspirational. A key outcome from this activity is providing visibility and security metrics. Without proper tuning, automated tools can produce a high number of false positives, consuming valuable time and resources. Often, tooling for pipeline integration is designed for use by security teams and is not helpful to even security-focused engineers. Constant pressure on an engineering team to triage and respond to a myriad of false positives or minor issues quickly erodes trust and goodwill.

Table 3.2: DevSecOps Prioritization

High	Medium	Low
RASP	DAST	SAST
Penetration Testing	KPI Visibility	
Threat Modeling	Open Source Security Management	
Security Focused Unit Tests		

The criteria for DevSecOps activity prioritization includes optimizing velocity and tech that yields minimal false positives while positively impacting security.

3.4.1 External Validation

External validation uses a third party to confirm the security of existing code. Penetration testing is a temporary activity that can identify critical vulnerabilities and serves as a "badness-ometer"[8], a term coined by Gary McGraw. Alternatively, bug bounties and full red teaming can achieve similar goals, but bug bounties success greatly depends on what a team implements and the structure of the bug bounty. External validation is a crucial component of any security engineering process. It provides valuable KPI metrics and catches critical security issues before a security breach occurs.

Activity	Cost	Scope	Skill	Result
Penetration Testing	Varies	Customer sets scope	Varies with $	Vulnerabilities
Attack Simulation	High	Fully (phishing, old infrastructure)	Intentionally varied	Attack paths
Bug Bounty	Low	Attacker chooses from limited scope	Unknown	Vulnerabilities

3.4.2 Monitoring and Analysis

A DevSecOps strategy should include real-time monitoring, analysis, and protection. Web Application Firewalls (WAFs) are beneficial when used correctly. Historically, WAFs haven't fulfilled their promise, but they offer valuable capabilities to any API or web application. New wave WAFs encompass not just monitoring and analysis but also protection; falling into a new category: Runtime Application Self Protection (RASP). Compared to traditional WAFs, RASP offers more advanced capabilities and extraordinarily low rates of false positives.

Understanding ongoing attacks on your application is another significant aspect of monitoring. This important lagging indicator of how to adjust WAF/RASP solutions allows for a collaborative "attack-driven defense" when working to resolve security vulnerabilities.

3.4.3 KPIs

Developing clear, measurable security metrics that provide value can be challenging and may need to be tailored to the implementing organization. This section details a few key metrics that work well for most teams.

CVSS/CWSS Scoring of Security Issues

CVSS, or Common Vulnerability Scoring System, is the most popular vulnerability scoring system. It is generally simpler and more straightforward to implement, but it can miss the nuances and struggles to tailor the score around the business impact. CVSS only applies to existing vulnerabilities.

CWSS[9] stands for Common Weakness Scoring System. The CWSS system has more parameters and can apply to vulnerabilities that don't even exist. This aligns more with our goals of shifting security to earlier stages in the development lifecycle. Teams should, at minimum, apply a CVSS score to all their issues in their tracking system. As the organization's security processes mature, more focus can be placed on CWSS. CVSS and CWSS can both mislead, as bugs may score with a lower impact, but result in catastrophic consequences for an organization (e.g.,

[8]McGraw, G. (2019, March). Badness-meters are good. Do you own one? Retrieved from Badness-meters are good. Do you own one? website: https://www.synopsys.com/blogs/software-security/badness-ometers/

[9]Martin, B., & Coley, S. (2014, September 5). Common Weakness Scoring System (CWSS). Retrieved from Common Weakness Scoring System (CWSS) website: https://cwe.mitre.org/cwss/cwss_v1.0.1.html

information disclosure may be a critical vulnerability in some environments). Despite these caveats, scoring security issues is simple and serves as a rough yardstick. Over time, organizations can refine and adjust their scoring systems to ensure that low scoring issues in vanilla CVSS/CWSS can be appropriately tuned.

Planned and Unplanned Security Issues

Planned and unplanned security issues are key metrics that any security engineering effort should include. A planned security issue or feature is one that is designed into the system and implemented. Simply tag such stories/issues as they're implemented. Ideally, planned security issues occur in the normal flow of development and they don't resemble or have the cost of unplanned bug fixes. Unplanned security issues are those identified after software release. These issues are often more expensive to fix and may require significant architectural changes that would not be necessary if the system were properly planned for. Over time, organizations aim to reduce unplanned security issues using the techniques discussed in this whitepaper.

Build time delay

This simple metric can be tricky to iron out. The tools that run in the pipeline or on a developers machine should be streamlined to avoid increasing pipeline build time by more than 5 minutes. Security is only a part of any product and should not provide significant contributions to the deployment or build times.

Tools that provide value to the security and engineering teams but increase the build time by an unacceptable amount can run out of band regularly.

Time spent resolving security issues

Tracking the time spent resolving security issues can necessitate relatively sophisticated time tracking and may not be feasible for many organizations. However, the goal is clear: to track and ultimately minimize the time spent on unplanned security issues. Tracking the implementation effort for planned security issues can also provide insight into the relative cost of security in the development process and help manage it accordingly.

Number of builds that fail due to security

This metric depends on two key things:

1. Security tooling that is useful and capable of failing a build based on automated analysis or other ad-hoc checks.
2. Working to reduce failed builds to an acceptable level according to organization standards.

The key idea here is to have tooling, even if it's simple scripting doing sanity checks on the environment. For example, in a C/C++ environment, it is possible to use unsafe string functions. Also, rudimentary static analysis provided by the compiler can easily fail build if these functions are used. Track these build failures and minimize them over time through education and developers receiving immediate feedback in the build process.

Conclusion: Security can be simple, but not easy

This paper provides an overview of one approach to achieving security engineering outcomes within an organization that builds software. Security engineering is a relatively new field. There's no precise agreement on what security engineering means to various security practitioners. In this paper, we define security engineering as a systematic approach to achieving meaningful security outcomes for an organization. These outcomes will differ to some degree for each organization, hence tools such as the NIST Cybersecurity Framework and BSIMM are recommended. They establish a baseline against which organizations can measure their security activities and practices. Alongside the baseline, maturity goals and high-impact security practices and activities are identified. A roadmap is created based on a prioritization of difficulty of implementation and impact. The roadmap, along with the specific implementation plan, is built with reasonable targets in mind. The team or teams then work towards implementing the roadmap. Measurement and KPIs are put in place to track success and continue the process of improving an organization's security engineering practices within ivision.

References

Boehm, Barry W. 1987. “Improving Software Productivity.” *IEEE Computer*, September, 43–57.

Boehm, Barry W.and Philip N.Papaccio. 1988. “Understanding and Controlling Software Costs,” *IEEE Transactions on Software Engineering* 14 (10): 1462–77.

Card, David N. 1987. “A Software Technology Evaluation Program,” *Information And Software Technology* 29 (6): 291–300.

Gilb, Tom. 1988. *Principles of Software Engineering Management*. Wokingham, England: Addison-Wesley.

Gilb, Tom, and Dorothy Graham. 1993. *Software Inspection*. Wokingham, England: Addison-Wesley.

Howard, Michael, and Steve Lipner. 2006. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press.

Jones, Capers. 1991. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill.

Jones, CapersTutorial, ed. 1986. *Programming Productivity: Issues for the Eighties*. 2nd Ed. Los Angeles: IEEE Computer Society Press.

Kitson, David H.and Stephen Masters. 1993. “An Analysis of SEI Software Process Assessment Results, 1987-1991.” *Proceedings of the Fifteenth International Conference on Software Engineering*, 68–77.

Martin, Bob, and Steve Coley. 2014. “Common Weakness Scoring System (CWSS).” *Common Weakness Scoring System (CWSS)*. https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

McGraw, Gary. 2019. “Badness-Meters Are Good. Do You Own One?” *Badness-Meters Are Good. Do You Own One?* https://www.synopsys.com/blogs/software-security/badness-ometers/.

McGraw, Gary, Sammy Migues, and Jacob West. 2018. “Building Security in Maturity Model (BSIMM) Version 9.” https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf.

Microsoft. n.d. “Elevation of Privilege Download.” https://www.microsoft.com/en-us/download/details.aspx?id=20303.

Russell, Glen W. 1991. “Experience with Inspection in Ultralarge-Scale Developments,” *IEEE Software* 8 (1): 25–31.

Shostack, Adam. 2010a. “Elevation of Privilege Game.” https://adam.shostack.org/Elevation-of-Privilege-BlackHat2010ShostackFinal.pptx.

———. 2010b. “Black Hat USA 2010: Elevation of Privilege: The Easy Way to Threat Model.” https://www.youtube.com/watch?v=gZh5acJuNVg.

———. 2014. *Threat Modeling: Designing for Security*. Wiley.

Standards and Technology, National Institute of. 2018. “Framework for Improving Critical Infrastructure Cybersecurity.” https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.

References

Boehm, Barry W. 1987. “Improving Software Productivity.” *IEEE Computer*, September, 43–57.

Boehm, Barry W.and Philip N.Papaccio. 1988. “Understanding and Controlling Software Costs,” *IEEE Transactions on Software Engineering* 14 (10): 1462–77.

Card, David N. 1987. “A Software Technology Evaluation Program,” *Information And Software Technology* 29 (6): 291–300.

Gilb, Tom. 1988. *Principles of Software Engineering Management*. Wokingham, England: Addison-Wesley.

Gilb, Tom, and Dorothy Graham. 1993. *Software Inspection*. Wokingham, England: Addison-Wesley.

Howard, Michael, and Steve Lipner. 2006. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press.

Jones, Capers. 1991. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill.

Jones, CapersTutorial, ed. 1986. *Programming Productivity: Issues for the Eighties*. 2nd Ed. Los Angeles: IEEE Computer Society Press.

Kitson, David H.and Stephen Masters. 1993. “An Analysis of SEI Software Process Assessment Results, 1987-1991.” *Proceedings of the Fifteenth International Conference on Software Engineering*, 68–77.

Martin, Bob, and Steve Coley. 2014. “Common Weakness Scoring System (CWSS).” *Common Weakness Scoring System (CWSS)*. https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

McGraw, Gary. 2019. “Badness-Meters Are Good. Do You Own One?” *Badness-Meters Are Good. Do You Own One?* https://www.synopsys.com/blogs/software-security/badness-ometers/.

McGraw, Gary, Sammy Migues, and Jacob West. 2018. “Building Security in Maturity Model (BSIMM) Version 9.” https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf.

Microsoft. n.d. “Elevation of Privilege Download.” https://www.microsoft.com/en-us/download/details.aspx?id=20303.

Russell, Glen W. 1991. “Experience with Inspection in Ultralarge-Scale Developments,” *IEEE Software* 8 (1): 25–31.

Shostack, Adam. 2010a. “Elevation of Privilege Game.” https://adam.shostack.org/Elevation-of-Privilege-BlackHat2010ShostackFinal.pptx.

———. 2010b. “Black Hat USA 2010: Elevation of Privilege: The Easy Way to Threat Model.” https://www.youtube.com/watch?v=gZh5acJuNVg.

———. 2014. *Threat Modeling: Designing for Security*. Wiley.

Standards and Technology, National Institute of. 2018. “Framework for Improving Critical Infrastructure Cybersecurity.” https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.