March 12, 2020

Security Engineering

Version 1.0

https://carvesystems.com

Contents

Executive Summary: Why Security Engineering?

Why should teams care about security engineering?

Our premise for security engineering is simple, and is backed up by our experience with multiple teams releasing high quality software. Modern software teams often lack 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 attempt to define what DevSecOps is and why it is important, this white paper will outline key security engineering practices we use successfully with our customers, many of which 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 that has fewer defects will also, generally, have fewer security defects. Experienced product teams will inevitably be familiar with Boehm's[1] cost of change throughout the SDLC:

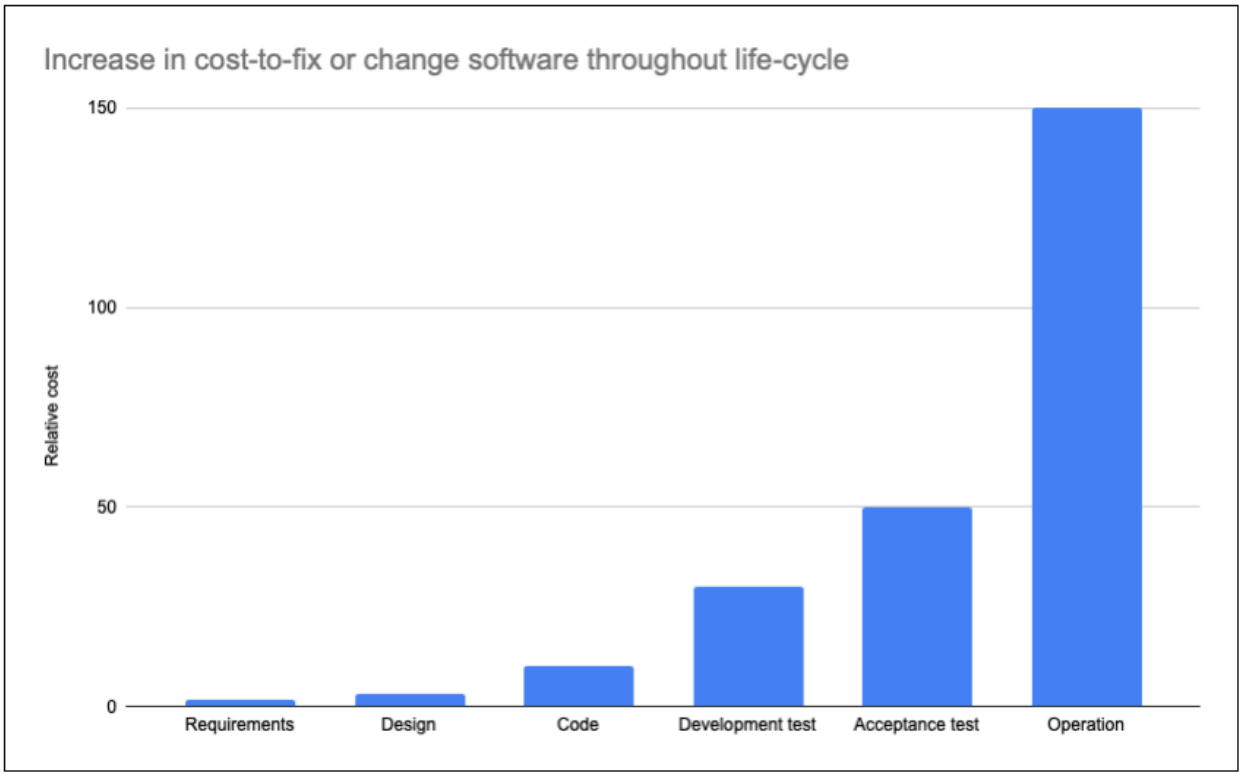

Figure 1.1: Boehm relative cost to fix or change software throughout life cycle

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

Later studies have generally backed up this graphic[2,3,4]. There are some refinements and key details to consider, but, in short, it is highly beneficial to get things right up front. This doesn't mean everything reverts back to Requirement and Design heavy Waterfall methodologies, but it does imply, strongly, that to have a high velocity software development lifecycle, we need to deploy software with as few defects and architectural flaws as possible while not falling down a rabbit hole of chasing perfection over progress. Our goal in this paper is to discuss how to improve quality, specifically around security defects, without sacrificing feature release velocity.

Following from the fact that security defects are still just basic software security defects, it becomes imperative to address these defects like any other engineering problem, which often means changing an organization's culture around engineering, and security engineering, specifically. The rest of this paper's goal is to provide the details of Carve's approach to security engineering and implementing lasting change at product organizations. The key goal is to spend time and resources efficiently and outperform competitors in security. Implementing security engineering moves product and software security from a fear and compliance driven activity to being a normal part 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 may be familiar with the phrase “Shift security left”. It is overused as a “catch-all” phrase for software security now, but the general concept is something savvy security practitioners have been advocating for well over a decade. Microsoft was one of the first large organizations to pioneer security in the development lifecycle. They published the results of their efforts within their own organization via books, articles, and white papers. Focusing on the development lifecycle is a large part of the battle, but it is difficult for organizations to go from zero security lifecycle and security engineering to a complete and robust Secure Development Lifecycle (SDL). Our approach bridges the gap by prioritizing activities and building a roadmap that optimizes for positive security impact while not reducing release velocity.

One of the key goals in the implementation of this process is to identify where various types of security vulnerabilities occur and eliminate them with security engineering and secure development practices. The goal should be to create a generative and positive security culture within an organization that eliminates many of the traditional barriers information security is perceived to create for product teams. Implementing security engineering should be empowering for development teams, not a time sink. Although the focus of this paper is on SDL and security engineering, it is important to keep the needs of the business omnipresent throughout the implementation of these practices. The aim is to create a cooperative security program that doesn’t pit a security team against the engineers in an adversarial structure. If the teams work in harmony and identify security vulnerabilities earlier in the life cycle, issues are easier to remediate, avoiding difficult choices such as delaying a release or releasing software with known vulnerabilities.

The rest of this paper explores how to achieve these outcomes using a methodical process.

Achieving Security Engineering Outcomes

Broadly, always begin with the desired outcomes in mind. The goal is to define security outcomes for your product or software releases. At a high level 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

Once outcomes are defined, we propose the following mapping of activities onto the SDLC:

Figure 3.1: SDLC Activities Mapping

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

3.1 Defining Security Outcomes

We borrow from two well respected frameworks to help measure and implement our security outcomes. The first goal of any software organization must be to understand what current best practices are and how well the organization is adhering to those best practices. BSIMM and the NIST Cybersecurity Framework are two excellent options for performing a basic risk assessment and gap analysis. Depending on your industry there may be other compliances regimes that come into play, such as PCI, GDPR and HIPAA.

BSIMM[1] (Building Security In Maturity Model) focuses more on the software security side of building software. It has many key activities and a relatively easy self assessment process, one of the key challenges with BSIMM is knowing

[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

which activities to prioritize after a self assessment to optimize the effort. The NIST Cybersecurity Framework[2] is a holistic cybersecurity framework that encompasses many activities found in BSIMM. We have found that combining these two frameworks into a blended risk assessment and gap analysis provide good visibility into an organizations overall security posture and help better prioritize activities.

The BSIMM Framework has four key focus 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	Cofiguration Mnaagement & Vulnerability Management

Figure 3.2: BSIMM Framework

The NIST Cybersecurity Framework breaks down into five key areas:

Figure 3.3: NIST Cybersecurity Framework

Carve Systems has created a blended cybersecurity assessment process that combines BSIMM and NIST while adding in elements such as an application inventory and data classification scheme. Once the high level assessment work has been conducted, most software organizations focus on BSIMM oriented activities and are responsible for the software security oriented portions of the roadmap. In the modern DevOps environment, NIST provides a more operational view of the security posture space and ensures that the operational focus is taken into consideration. The ultimate goal of this assessment process is to create a roadmap teams can follow that incorporates BSIMM and NIST Cybersecurity Framework activities and maturity goals.

[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

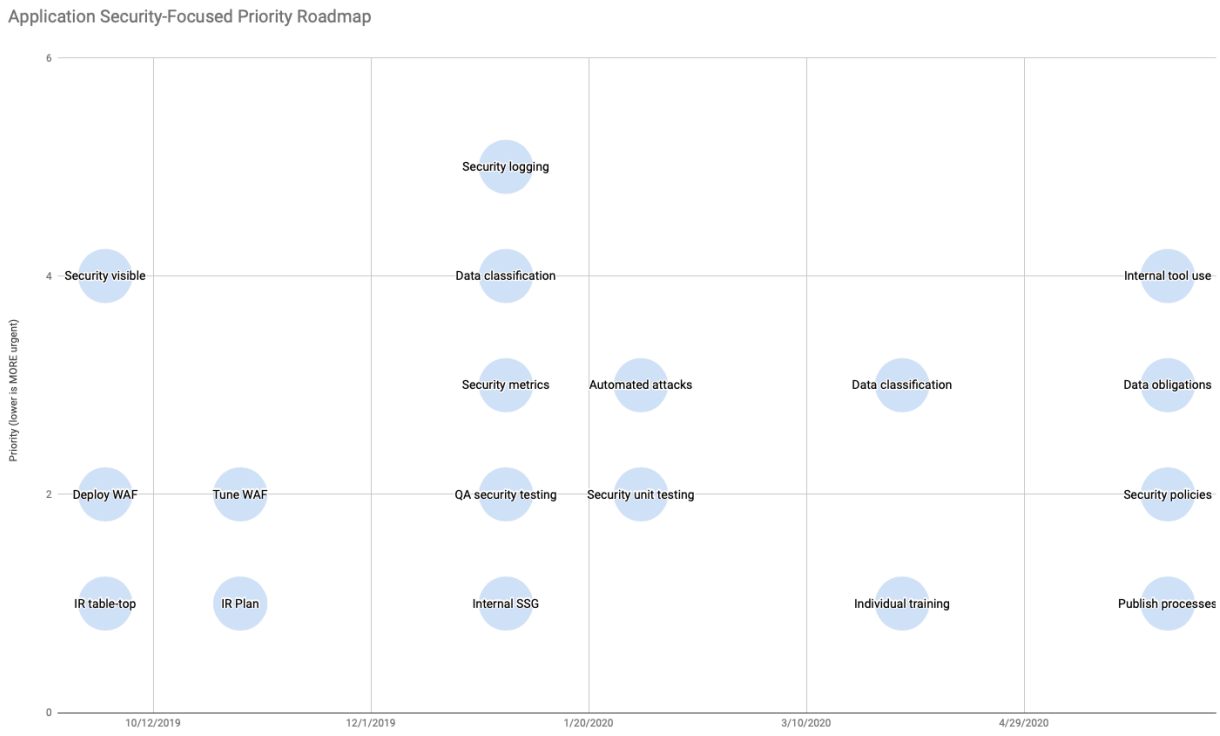

Figure 3.4: Example Application Security Roadmap

Security engineering outcomes all start with a good roadmap that incorporates an organization's priorities based on an actual assessment and analysis. This roadmap and the assessment process that creates it provides the skeleton from which fully defined security outcomes can be supported. From here, key activities must be examined and how they impact the overall security engineering process and security outcomes for an organization. These will ultimately be driven by senior goals that are easy for an executive management team to understand. "Our senior goal is to prevent a data breach" or "Our senior goal is to protect our brand and image" are completely reasonable senior goals driven by worst case scenarios for an organization. The assessment process is always performed with the senior goals in mind when prioritizing and roadmapping.

3.2 Threat Modeling

Threat modeling answers the question of "what can go wrong" in a structured process. The style of threat modeling that your organization uses must be tailored towards engineers so that they can enumerate threats in their system effectively without deep expertise on security issues or the process of threat modeling. Part of the premise of threat modeling is that engineers know their system best and with the right set of activities, e.g. threat modeling, the engineers can identify security flaws and implementation bugs. This is part of the process of creating a culture of security and training engineers to think like both an attacker and a defender.

The section outlines the threat modeling process. Carve considers threat modeling a foundational security engineering activity that helps rapidly shift security left within an organization. Security culture is covered in more detail in this paper, but **Security Champions** are trained in threat modeling early in the program. The Champions bring this knowledge of how to conduct a threat model to their teams and ensure that at every iteration and software release, the teams are also performing threat modeling on features, before they are implemented.

This section will briefly outline threat modeling and how it works, but is not meant to be a comprehensive guide to threat modeling, but is meant to illustrate the key components of the threat modeling process and why they

are valuable. Threat modeling often starts with a good diagram, and we recommend using a relatively strict and simplistic type of diagram called a Data Flow Diagram (DFD) for developers and teams relatively new to threat modeling. The data flow diagram identifies the components in a system, which then allows for a per component analysis. For each component the threat model then aims to understand "what can go wrong?". We recommend teams execute 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

Stride is performed for each component in your system, represented on in a DFD.

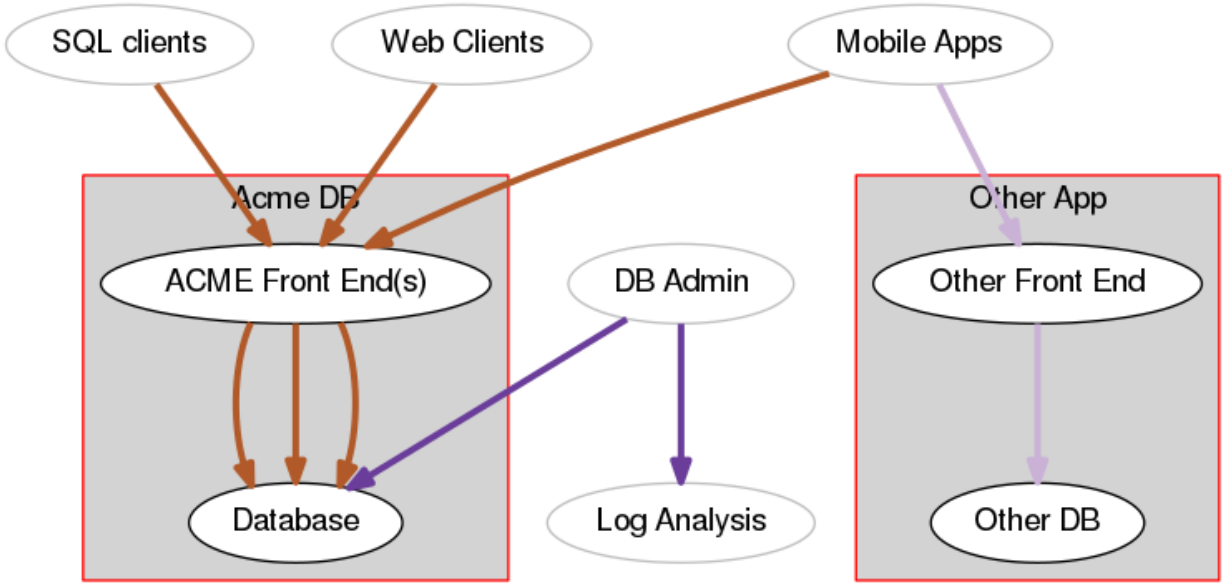

Figure 3.5: Dataflow Diagram Example

Once an enumeration of threats has been performed they can be triaged managed. It is important to understand that a Threat Model is a living activity that evolves with the software. Once a threat has been identified it is likely to remain a threat for the entire lifetime of the system. A vulnerability is a threat realized in a software system. Even if that vulnerability is mitigated, the threat remains. For example, a system that relies on a SQL based relational database will always have a threat for "SQL Injection" until that system no longer relies on a SQL based RDBMS.

We can use an analogy of building a house to illustrate some of the early concepts in designing a security program. The engineers are the builders. One particular threat may be the threat of a break-in and theft of your belongings. That threat will never disappear, although we can mitigate the threat by locking doors, closing the garage door, and putting bars on windows. In that case, the threat of a break-in is mitigated to an appropriate degree. However, if a vulnerability is introduced (garage is left open at night), then the threat could be realized by a burglar (i.e. break-in and theft of your belongings).

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

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

3.2.1 Security Architecture Review

Security architecture review is a close relative of threat modeling, but it differs in that it is not meant to be a comprehensive review of a given system. Rather, the goal of security architecture review is to make important security design decisions about the architecture. Many times these decisions can be informed 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, placing the burden on each microservice to deal with authentication and authorization. Security architecture review is here to take a critical look, early in the design process, to understand the implications of such decisions. In the given example, this creates an "insecure by default" situation for each microservice and the developers must actively write code or configure their service to perform authentication and authorization. Security architecture review systematically addresses this by understanding the potential architecture flaws and bugs holistically and continuously.

3.2.2 Learning Threat Modeling and Implementation Ideas

Threat modeling is something that practitioners can and do spend a significant amount of time gaining expertise in. But the 80% solution for development teams is often learning just enough technique and mindset to be effective at identifying key issues. One of our favorite methods of training teams up on 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." This concept seems great on the surface, but it is a skill that many practitioners develop over time. Tools, such as games, can make threat modeling much more relevant and collaborative, while reducing the friction of the learning process. In our experience the response to the EoP game is very positive. It has been around for almost 10 years now and it continues to work reasonably well as an introduction to threat modeling concepts.

To continue the analogy from above, it is more expensive to put bars on the windows and install a garage door opener with a rotating code after the house is built than it is to build the windows a little higher up and install a secure garage door opener when the house is being built. Engineers should be trained in threat modeling concepts so that it can be done in-band, reducing the risk of re-work later on because of a design flaw.

3.3 Security Culture and Active Support

Creating lasting change within an organization often requires changing fundamental behavior within the organization. In this case, the focus is on the culture of security and creating a "generative" mindset in everyone writing code. Although BSIMM identifies numerous activities that bring security into a software system, we have found that security culture in particular is a huge component of any successful security engineering within an organization.

Security culture is about maturing how the organization approaches security, and the way it builds software in general. To that end, creating a software security group that collects key stakeholders and engineers into one group is a powerful concept. This group is a resource to the organization and has "office hours" and a formalized role within the software development process. The Software Security Group (SSG) according to BSIMM[7] is:

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

[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

This is a first, and foundational, step in any software security initiative. The first members of a SSG are often key stakeholders, such as architects, and security champions. Security champions are generally developers assigned to own security for their team. The next section covers security champions and the pivotal role they play in creating a culture of security at organizations.

A SSG should hold regular "office hours" to support engineers and perform simple internal marketing. For example, handing out unique shirts for engineers who engage the SSG, or even offering snacks at the office hours can make an impact on engagement within the organization.

3.3.1 Security Champions

Security champions become a critical part of creating security culture within an organization. The following is a summary of the key features of a security champion:

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

Security leadership includes activities such as leading threat models, and owning security specific code and features in a code base. Security champions are also a part of the Software Security Group (SSG). Having visible security champions and a SSG make security a visible and more approachable part of an organization's culture. It also reduces the separation between security and engineering. Performing threat models means security issues will often be fixed at design time and not later in the life cycle when it can be more painful to remediate security issues, especially if it requires deeper architectural support to remediate a given issue.

Expect for security champions to spend approximately 10-30% of their time learning, implementing light weight threat modeling, identifying security relevant features, and identifying security issues during planning, where they can be addressed less expensively. We recommend starting with threat modeling and light OWASP Top 10 training (if applicable). This gives Champions a baseline and common vocabulary for discussing security-relevant issues.

Ideally, security champions are volunteers from the engineering team with an interest in security. It is also critical to have management buy-in for this activity, because it will initially take a portion of the engineer's time. We argue that this time gets re-paid later on, when issues can be identified earlier in the SDLC, less time is spent on un-planned security work, and generative security teams build guides and code snippets that can be re-used across teams to save time.

3.4 DevSecOps and Security Automation

Security in a DevOps pipeline is still largely aspirational. One of the key outcomes from this activity is providing visibility and security metrics. Automated tooling produces a high number of false positives if it is not adequately tuned, which takes valuable time and resources. In most cases, the tooling for pipeline integration is designed for consumption by security teams and is not helpful to even security-focused engineers. Forcing an engineering team to triage and respond to dozens 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		

Our criteria for prioritizing DevSecOps activities is based on optimizing velocity and technology that produces minimal false positives while having a positive impact on security.

3.4.1 External Validation

External validation refers to using a third party to validate the security of existing code. Penetration testing is a point in time activity that can identify critical vulnerabilities and serves as a "badness-ometer"[8], a term coined by Gary McGraw. Separately, bug bounties and full red teaming can achieve similar goals, however bug bounties are often hit and miss and greatly depend on what a team is implementing and how the bug bounty is structured. External validation is an important component of any security engineering process. It provides valuable KPI metrics and catches critical security issues before they become a breach.

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 part of a DevSecOps strategy has to be real time monitoring, analysis, and protection. Web Application Firewalls (WAFs) are a proven technology. Historically, WAFs have not lived up to their promise, but when used correctly they provide valuable capabilities to any API or web application. New breeds of WAFs provide not just monitoring and analysis, but also protection and fall into a new category: Runtime Application Self Protection (RASP) and provide more advanced capabilities than traditional WAFs. RASPs provide extremely low rates of false positives.

Another key aspect of monitoring is visibility into ongoing attacks. Knowing how your application is actually being attacked is an important lagging indicator of how to tune WAF/RASP solutions and allows for collaborative "attack-driven defense" when working to fix security vulnerabilities.

3.4.3 KPIs

Clear, measurable security metrics that provide value can be very challenging and often need to be tailored to the organization implementing them. This section details a few key metrics we think work well for most teams.

CVSS/CWSS Scoring of Security Issues

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

CWSS[9] stands for Common Weakness Scoring System. The CWSS system has more parameters, but can be applied to

[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

vulnerabilities that don't even exist and is more in line with our goals of moving security to earlier in the development life cycle. Initially, teams should at least apply a CVSS score to all of their issues in their issue tracking system. As the organization's security processes mature, more focus can be paid to CWSS. CVSS and CWSS can both be misleading as bugs can score with a lower impact, but result in catastrophic consequences for an organization (e.g. in some environments information disclosure is a critical vulnerability). Caveats aside, it is simple to score security issues and it serves as a rough measuring stick. Organizations can refine and tune their scoring system over time to ensure that issues in vanilla CVSS/CWSS that might score low can be tuned appropriately.

Planned and Unplanned Security Issues

Planned and unplanned security issues are one of the key metrics any security engineering effort should include. A planned security issue or feature is one that is designed into the system and implemented. A simple tag is then applied to any such stories / issues as they are implemented. In theory, planned security issues are happening in the normal flow of development and don't look like or have the expense of unplanned bug fixes. Unplanned security issues are those that are discovered after the software has been released. These issues will tend to be more costly to fix and in some cases can require significant architectural band aids that would not be required if the system is designed and planned for. Over time an organization should aim to reduce unplanned security issues using the techniques discussed in this whitepaper.

Build time delay

This is a simple metric, but it is worth sorting out. Whatever tools are running in the pipeline or on a developers machine should be streamlined to not increase pipeline build time by more than 5 minutes. Security is only a slice of any product and it should not be contributing greatly to the deployment or build times.

Tools that provide value to the security and engineering teams, but increase build time by an unacceptable amount, may run out of band on a regular basis.

Time spent resolving security issues

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

Number of builds that fail due to security

This metric relies on two key things:

1. Security tooling that does something useful and can fail a build based on automated analysis or other ad-hoc checks.
2. Ensuring that failed builds are reduced to an acceptable level, which will vary per organization.

The key concept here is to have tooling, even if it is simple scripting that does sanity checks on the environment. For example, in a C/C++ environment it is possible to use unsafe string functions. It is also simple to fail a build if those functions are used via rudimentary static analysis provided by the compiler. These build failures can be tracked and then minimized via education and developers learning over time via 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 new field and there is 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. What these outcomes are will vary for each organization to a small or large extent, which is tools such as the NIST Cybersecurity Framework and BSIMM are recommended to establish a baseline that organization's can measure their security activities and practices by. Along with 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 and the specific implementation plan is built with reasonable targets in mind. The team or teams then works to incorporate and implement the roadmap. Measurement and KPIs are put in place to track success and continue the process of improving an organization's security engineering practices.

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.