VERACODE

The DevSecOps Playbook

Practical Steps for Producing Secure Software

Table of Contents

Introduction

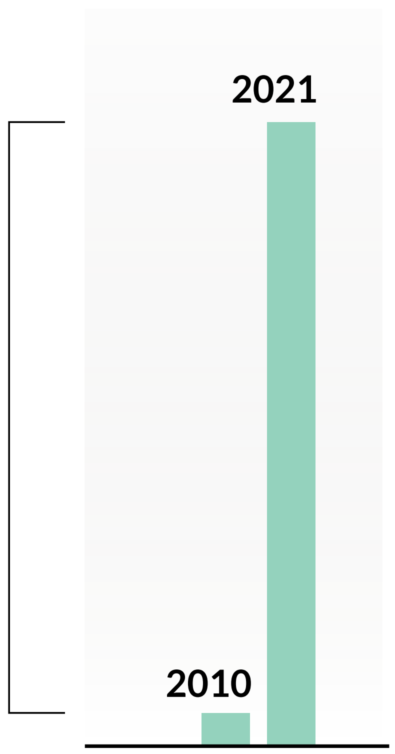
The traditional approach to software development (with security at the end) simply can't keep up with the speed of DevOps and modern software development. Research in the **State of Software Security v12** shows there has been a 20x increase in median scan cadence: from users averaging two or three scans per year in 2010 to 90 percent of applications being scanned at least once per week in 2021.

Continuous testing and integration, which includes security scanning in pipelines, is becoming the norm as a part of modern software development methodologies; that's where the DevSecOps philosophy enters the picture.

In this eBook, we'll first look at why incorporating security into DevOps to create DevSecOps is critical, and then we'll dive into what successful DevSecOps looks like and how it works in practice. By the end, you'll know how and why to build a modern software development workflow around security from the get-go.

20x

increase in median scan cadence from 2010 to 2021



DevSecOops – A Cautionary Tale

Before we dig into the elements of a successful DevSecOps implementation and what that might look like, let's cover an example of what happens when the "Sec" is missing from DevOps or is siloed and left to the end of the SDLC. The following is a fictional story, though it relays the all-too-real story of many companies you've heard of, buy from, or maybe even show up to work for every day.

Fail Fast Technologies was a company like any other attempting to develop modern software at a rapidly increasing rate using agile and DevOps processes.

Fail Fast had many applications to track and security flaws of varying degrees of severity in most of their applications. With a lean development team and aggressive delivery targets, developers weren't prioritizing secure coding or rigorous code testing in their environments. They performed basic SAST scans on applications on the critical path before adding their code to the larger repository. They were doing their best to prevent adding net new tech debt, but since **the average application grows at about forty percent per year** for the first five years regardless of its original and by the five-year mark 70 percent of applications contain at least one security flaw, they just continued watching that debt pile up.

Due to budget constraints, there was only one person functioning as the liaison between the security and development teams. New applications and APIs went through routine security tests at the end of each development cycle, vulnerabilities were triaged, and JIRA tickets were manually created by this SecOps practitioner for each developer – a task that took at least half a day. Developers generally had to delay prioritization of defects because vulnerability findings weren't in sync with their current sprint plan, and the lack of prioritization meant the slowing remediation of security flaws. Even so, the security effort struggled to keep pace with development. The SecOps practitioner routinely asked, "Why don't they catch this stuff earlier," and "How did they not see this?" It had been suggested that the SecOps practitioner lead a workshop to help the developers code more securely, but it never got organized. There just wasn't enough time. "Security" could barely test the new applications coming through and juggle patching urgent vulnerabilities that seemed to make the news before Fail Fast was aware of them. Even though the Fail Fast SecOps practitioner knew he needed to take a more comprehensive look at the entire application ecosystem and what needed an update, rework, or patch, he could only prioritize the applications that were on fire.

Then one day, the security team liaison had to make the call that anyone in his position would dread. He had to call the CISO and inform him that a bad actor had found a way to leverage a SQL injection vulnerability and cause a substantial breach – millions of records containing very sensitive personal identifying information of their customers had been stolen.

It was a regular day like any other until the CISO got that call...

What Went Wrong?

As they began to dig into the breach details and try to get to the root cause, the Fail Fast teams realized the vulnerability could have been surfaced through a static scan when the application was developed, but that scan wasn't run because the development team was under pressure to ship the app as quickly as possible to beat an analyst review deadline. The vulnerability could have also surfaced with a dynamic scan post-production, but because the application was now older and not as critical to the value chain, the security team had not prioritized it as a dynamic scan candidate.

They realized they didn't really have a standard for examining and prioritizing the results of scans. Often developers will try to ignore issues if those issues conflict with their ambition to release new functionality. Developers needed to be able to efficiently perform critical scans and handle security issues more self-sufficiently instead of relying so heavily on the security team. To do this they needed the process, tools, and methods that would integrate into their current workflow. The SecOps practitioner and the Ops team needed to feel confident that developers were taking every precaution not to introduce new flaws into the code base. They also needed a way to evaluate the risk on their attack surface and prioritize dynamic scanning, penetration testing, and critical patch candidates.

A deeper examination points out that some of the manual processes, like assigning remediation tickets really slowed down the interaction and remediation activity. However, these were just symptoms of a larger problem. Adding sporadic automation could help, but the root cause was inconsistent utilization of security to development and operations. Security needed to be integrated into Fail Fast's software development lifecycle across the board. The teams realized that throwing a bunch of disparate tools at the problem could potentially overcomplicate things, because tools alone do not magically solve the problem or change culture. With so many different security tools to choose from, attempting to onboard and integrate them was difficult. They needed a better way to implement policy, track progress against tech debt, and burn down any new flaws introduced into the code base. They needed a common language and a unified way of viewing their security posture as a team. The Fail Fast team needed a DevSecOps makeover.

The Role of Security in Modern Software Development

In order to discuss how the Fail Fast team achieves successful DevSecOps, let's examine the role of security in modern software development methods. Many companies, like our fictional Fail Fast Technologies, are currently developing modern software using DevOps processes or looking to adopt it, so let's get clear on what DevOps is.

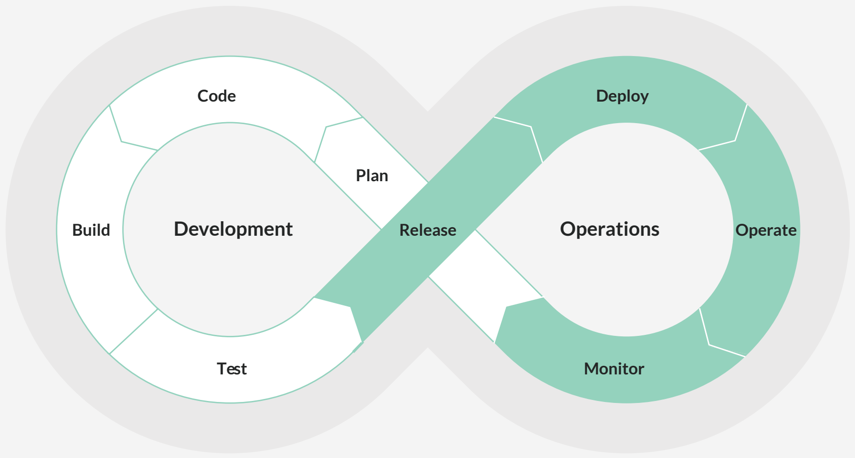
What is DevOps?

DevOps is a culture; it's a way of viewing the software development lifecycle like an ongoing conversation between teams. DevOps only works if teams can break down silos and collaborate – while still maintaining the integrity of their specific function.

DevOps is a philosophy that uses a set of combined practices to integrate software development and IT operations. DevOps complements agile software development by fostering a continuous feedback loop and chunking up work and the associated risk. One of the best aspects of DevOps is that you typically automate the entire building of environments from the ground up. You build the application, you build the host it will sit on, and you deploy it consistently to the host.

Now, let's investigate the specific functions that allow for automation. The integrated nature of DevOps is best viewed as a continuous loop. Every activity within the loop maps to a corresponding principle. Each step or fundamental function blends into the others drawing in practitioners across teams, from coding and testing to release and deployment, and finally to operating and monitoring, with the cycle repeating infinitely.

Real - Time Communication



Simplified Fundamentals of DevOps

Continuous Integration

Starting at the top left of the DevOps loop is Continuous Integration as a part of the "Code" and partially the "Build" function. Here we are taking the minimally complex pieces of the system and making sure they all still build and pass other tests to ensure the entire collection functions to specification. Many developers may be writing code for different applications or APIs in the same system, and all that code must "integrate" preserving the basic integrity of the entire collection of components. If any component fails a test, we know that the system is likely to fail, and we should not deploy it.

Continuous Deployment

Continuous Deployment (sometimes called Continuous Delivery) takes the output of the Continuous Integration step and gives it a place to live (these two steps are often grouped and referred to as CI/CD). This step deploys consistent environments which provide a live execution environment for the functional system. The environment can be referenced as the code that was written to deploy it and debugged as a system.

Continuous Testing/Validation

This step of the DevOps practice involves testing code to determine how it will behave in a production environment. Here tests can be measured against a set of objective metrics determined ahead of time by all teams involved in the DevOps effort. Those metrics qualify that the application is ready for end users. Load testing, functional testing, regression testing, and security testing are all examples of tests that validate an application as ready for production. This is the last place to address issues with performance or security.

Continuous Monitoring

The activities associated with Monitoring are generally led by the "Ops" side of the DevOps collaborative. Network, systems, applications, threats to any part of an organization's attack surface, and vulnerabilities that surface over time are monitored across applications and APIs in production. Monitoring is meant to provide feedback that can be actionable and result in development prioritization of said feedback – integrating the feedback brings us from the right side of the loop into planning, coding, and building.

What is DevSecOps?

The cornerstone of a successful DevOps practice is automation; this is why adopting DevSecOps (security automated within workflows) makes so much sense. DevSecOps is surrounding each step of the DevOps process and practice with security. By adding security into each step of the software development lifecycle (SDLC) from coding and building to operating and monitoring compliance to policy – code bases, applications and APIs are designed, built, and deployed with security in mind. Integrating security into each DevOps function effectively creates DevSecOps – an overarching layer covering all activity along the SDLC.

The Benefits of DevSecOps

The benefits of DevSecOps include overlaying security at each step, so it does not become a siloed afterthought and just that last “hurdle” to overcome before an application gets deployed. Practically speaking, this means actively applying security at every step in the SDLC – teaching developers to spot code flaws early, helping developers choose third-party libraries that do not introduce vulnerabilities, defining the meaning of “done,” and continuing to shift right as well as left when monitoring attack surfaces.

Why integrate security testing so tightly into the DevOps process? It allows teams to catch potential attack targets early where they are far less costly and exhausting to fix. For example, **research**[1] shows that users of hands-on **developer security training** who had completed at least one lesson took 110 days to remediate 50% of flaws – while those who had no such training took 170 days. That’s a difference of two months!

The thing is, successfully implementing DevSecOps is one of the most difficult problems to solve due to a variety of factors. Cultural adoption is one of the biggest blockers to shifting left and optimizing the “right” shifted evaluation and testing of the attack surface. Now we will return to our story of Fail Fast Technologies and see what steps they took to successfully incorporate security into DevOps to create DevSecOps.

1 info.veracode.com/soss-v12-ungated

Practical Steps for Adopting DevSecOps

While Fail Fast was already implementing DevOps, they needed to holistically automate security in their process to successfully adopt DevSecOps and avoid running into another crisis like the SQL injection.

Establish a Common Vernacular

The Fail Fast team started by establishing a common vernacular: agreeing on definitions of terms to mean the same to everyone on both teams. This eliminated confusion and helped each team member better understand what their requirements were before they moved their code to another part of the software development lifecycle.

CWE vs CVE

Even though the teams had heard these terms before, a few of them were not 100% clear on what they meant and in some cases were conflating them. It was important for the teams to understand the difference because it would help set them up for success by knowing how to treat each one differently within the context of their applications.

CWE. "Common Weakness Enumeration (CWE™) is a community-developed list of common software and hardware weakness types that have security ramifications," **states their website**. Not only do these weaknesses apply to software, but they also apply to design and architecture. There are thousands of CWEs, but the most recognizable compilations of security flaws are the **SANS** top 25, which represents the most common 25 CWEs that specifically affect software. There is also the Open Worldwide Application Security Project® **(OWASP)** Top 10 which applies more directly to web applications. An example of a CWE is CWE 89, or SQL injection, which can lead to direct access/manipulation of the underlying SQL database that was not intended through the original application's designed purpose – a weakness with which the Fail Fast team unfortunately was all too familiar. CWEs give teams a common language around security flaws and what those flaws are to help them prioritize where developers need more training and guidance as well as what to fix.

CVE. CVE stands for Common Vulnerabilities and Exposure. Much like CWEs, CVEs are organized in a way that allows teams to speak the same language around exploit capabilities or potential exploit capabilities. The CVE numbering system started back in 1999 with the National Institute of Standards and Technology's **(NIST)** effort to maintain a database of publicly disclosed security issues. AppSec vendors like Veracode began building their own proprietary databases to augment the NIST database. A CVE is an ID number that begins with the year the vulnerability was discovered followed by a series of numbers that uniquely identify it. CVEs are then scored low to high (0-10) for severity. A CVE indicates a specific issue associated with a particular library and version number. CVEs can only be issued by a certified authority, and there is a time-consuming reporting and proof of concept process that must be followed for a vulnerability to be "official" enough to warrant a CVE number. CVEs are real, demonstrable vulnerabilities.

Vulnerability vs Security Flaw

Security flaws and vulnerabilities are perhaps the easiest two security terms to mix up, leading many development and security teams to wonder what the difference is between the two.

To put it simply, a security flaw is an implementation defect that can lead to a vulnerability, and a vulnerability is an exploitable condition within the code that allows a bad actor to attack.

So, just because a flaw is not a vulnerability now, that does not mean it will not become one in the future as environments and architectures change or get updated. Any updates to the architecture or changes in the function of an application can expose that application to attacks.

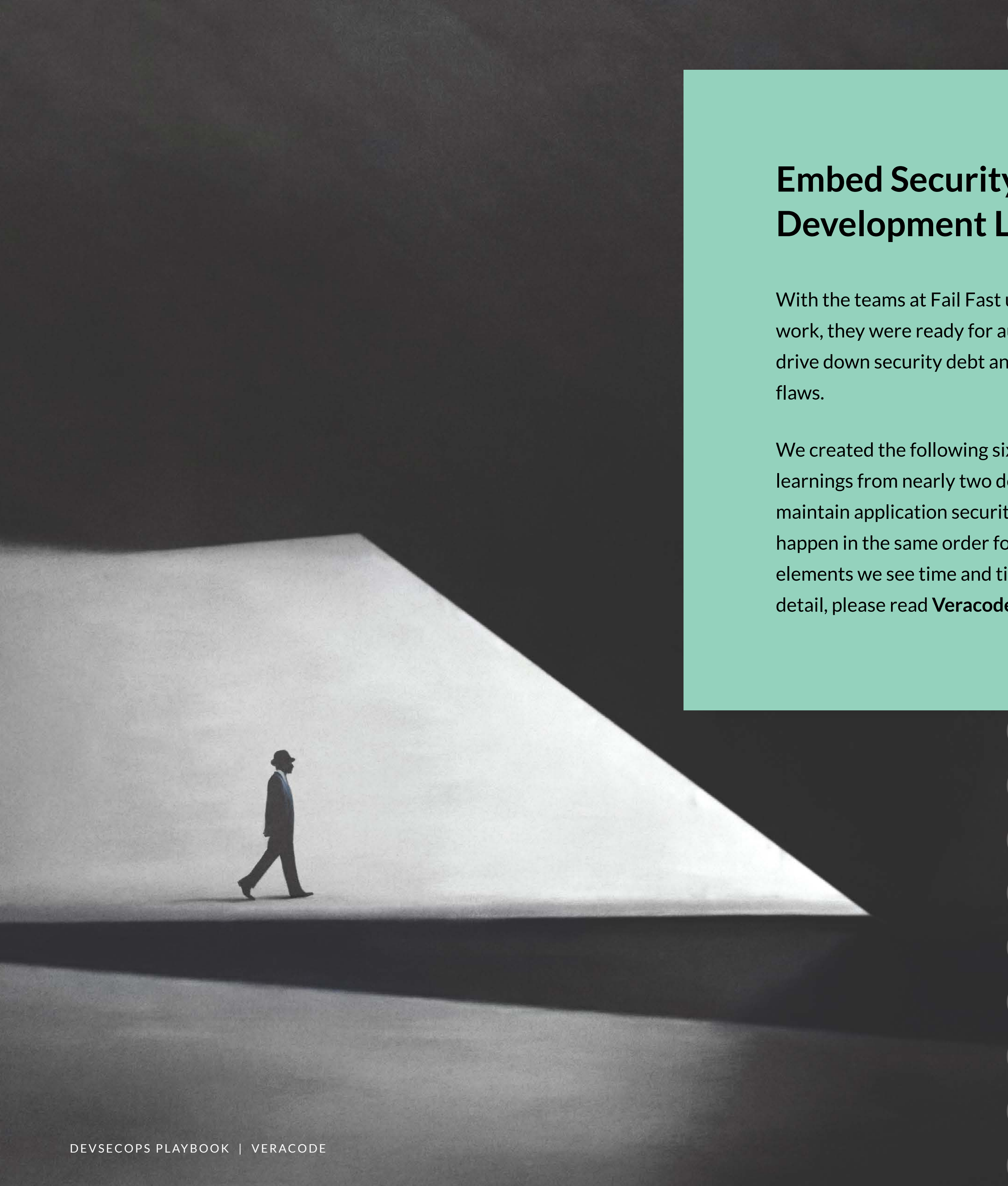
Once there is a known way to attack – or exploit – a flaw, the flaw becomes a vulnerability. The difference is probably best summed up this way: all vulnerabilities are flaws, but not all flaws are vulnerabilities. Plus, all flaws have the potential to become vulnerabilities. In the case of Fail Fast, the flaw in their code base became a vulnerability once an attacker figured out how to leverage it to exfiltrate sensitive data.

False Positive

According to the **(NIST)**, two ways to define a false positive are: "an alert that incorrectly indicates that a vulnerability is present" or "an alert that incorrectly indicates that malicious activity is occurring". It appears this would be a definition that's straightforward enough to satisfy both developer and SecOps practitioner's question: What needs to be fixed?

However, the debate between development and security teams in many organizations is that the security scan or evaluation of an application identifies flaws that, in the context of the larger application portfolio, don't necessarily represent an exploitable flaw. This debate goes to the heart of the miscommunication between developers and SecOps.

Flaws can be more or less likely to be exploitable. If a flaw falls into the "less likely" category, the developers will call it a false positive and resist or become suspicious about other flaws that are identified. Some vendors will allow labelling of a "flaw that isn't a vulnerability now" as a false-positive. Veracode requires these be mitigated, so that we do not lose track of it in the event that it does become a vulnerability in the future.



Embed Security into the Software Development Lifecycle

With the teams at Fail Fast using the same words to describe their work, they were ready for automating the tasks that would ultimately drive down security debt and reduce the likelihood of adding new flaws.

We created the following six steps to securing the SDLC based on learnings from nearly two decades of helping teams achieve and maintain application security programs. Note that the steps may not happen in the same order for everyone, but they form the six essential elements we see time and time again. To learn about them in greater detail, please read **Veracode's 6 Steps to Secure the SDLC eBook**.

1

Discover sources of risk and what to prioritize

The first key element to securing your SDLC is discovering and inventorying your application portfolio to understand sources of risk and what you want to prioritize. How many applications do you have? Where do they reside? Who owns them? Are they still around? What are your open-source dependencies?

Fail Fast realized they could not secure what they could not see, and that a comprehensive inventory of their attack surface was critical. Their first step was understanding what applications they had, what was in those applications, and what systems their teams used. They were struggling to identify all the applications, dependencies, and systems, and they felt as though they were boiling the ocean. They decided to focus first on just the applications they had already inventoried – knowing they could always iterate from there.

2

Onboard applications with an initial scan to establish a baseline and gain visibility

After Fail Fast had a good enough understanding and mapping of the applications, libraries, and artifacts that made up their software development portfolio and process, they onboarded applications with an initial scan to establish a baseline and visibility.

They were able to quickly scan thousands of applications using Veracode's cloud platform. They got an initial snapshot of potential issues in the code they wrote in-house and in their third-party components. They understood what was in the applications they had and decided how to automate the use of scanning engines moving forward.

One of the issues Fail Fast ran into was having one security person running tests while also figuring out how to triage and address the results of those tests. Everyone has a part to play in DevSecOps. They identified lead developers, scrum masters, and DevOps leads and let them know exactly what tests to run, when to run them, and how to automate the testing.

Developers

- Static Application Security Testing (SAST) - Scanning source or binary code in a-pre-production or "static" state to find security flaws or CWEs. Also called "white box" testing.

- Software Composition Analysis (SCA): Analyzing the open-source libraries and third-party components in an application for CVEs. Used in the creation of Software Bill of Materials (SBOM) and supports one or both standard formats: CycloneDX and SPDX.

Security Teams, Vulnerability Managers, or Operations

- Dynamic Application Security Testing (DAST): Simulating attacks in a production, runtime, or "dynamic" state to expose configuration and end-point weaknesses also known as "black box" testing.

Note: SAST finds security flaws while DAST finds vulnerabilities.

3

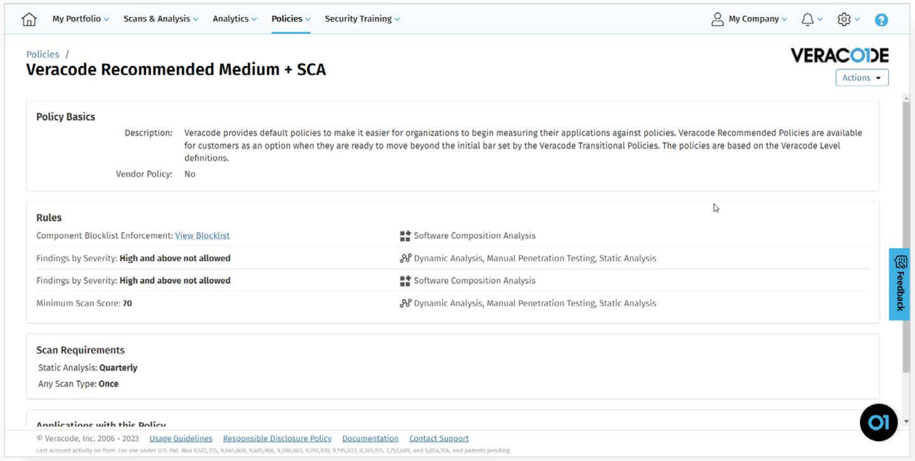
Define and assign security policies for applications

Policy sets clear expectations and rules with which applications must comply. At Fail Fast, the policy was not clear so compliance with it was not either. They needed to bring security and development teams, including board level, together around a common policy based on risk tolerance, DevOps processes, maturity, team capacity, and more.

Qualifying risk tolerance and setting policy is massively complex. Fail Fast did not have anyone on their team with quite the skill set to do this, so they relied on Application Security Consultants from Veracode to easily resolve this problem without having to wait for the hiring process to run its course.

Fail Fast started with Veracode's built-in security policies recommendations for applications based on business criticality; they planned to tailor them later and restrict findings by severity, CWE category, CWE ID, license risk, Common Vulnerability Scoring System (CVSS) score, or a common standard such as OWASP, OWASP Mobile, CWE Top 25, or Precast/Prestressed Concrete Institute Security Standards Council (PCI SSC).

Here are two examples. This is a recommended policy for most applications.



Policies /
Veracode Recommended Medium + SCA

Policy Basics

Description: Veracode provides default policies to make it easier for organizations to begin measuring their applications against policies. Veracode Recommended Policies are available for customers as an option when they are ready to move beyond the initial bar set by the Veracode Transitional Policies. The policies are based on the Veracode Level definitions.

Vendor Policy: No

Rules

Component Blocklist Enforcement: View Blocklist	Software Composition Analysis
Findings by Severity: **High and above not allowed**	Dynamic Analysis, Manual Penetration Testing, Static Analysis
Findings by Severity: **High and above not allowed**	Software Composition Analysis
Minimum Scan Score: **70**	Dynamic Analysis, Manual Penetration Testing, Static Analysis

Scan Requirements

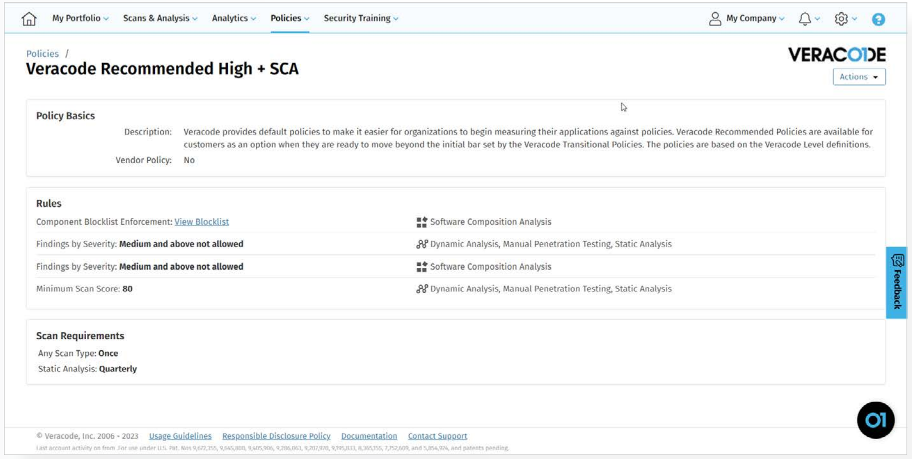
Static Analysis: **Quarterly**
Any Scan Type: **Once**

Applications with this Policy

© Veracode, Inc. 2006 - 2023 Usage Guidelines Responsible Disclosure Policy Documentation Contact Support

Last account activity on from . For use under U.S. Pat. Nos 9,672,355, 9,645,800, 9,405,906, 9,286,063, 9,207,970, 9,195,833, 8,365,155, 7,752,609, and 5,854,924, and patents pending.

This is a recommended policy for microservices.



Policies /
Veracode Recommended High + SCA

Policy Basics

Description: Veracode provides default policies to make it easier for organizations to begin measuring their applications against policies. Veracode Recommended Policies are available for customers as an option when they are ready to move beyond the initial bar set by the Veracode Transitional Policies. The policies are based on the Veracode Level definitions.

Vendor Policy: No

Rules

Component Blocklist Enforcement: View Blocklist	Software Composition Analysis
Findings by Severity: **Medium and above not allowed**	Dynamic Analysis, Manual Penetration Testing, Static Analysis
Findings by Severity: **Medium and above not allowed**	Software Composition Analysis
Minimum Scan Score: **80**	Dynamic Analysis, Manual Penetration Testing, Static Analysis

Scan Requirements

Any Scan Type: **Once**
Static Analysis: **Quarterly**

© Veracode, Inc. 2006 - 2023 Usage Guidelines Responsible Disclosure Policy Documentation Contact Support

Last account activity on from . For use under U.S. Pat. Nos 9,672,355, 9,645,800, 9,405,906, 9,286,063, 9,207,970, 9,195,833, 8,365,155, 7,752,609, and 5,854,924, and patents pending.

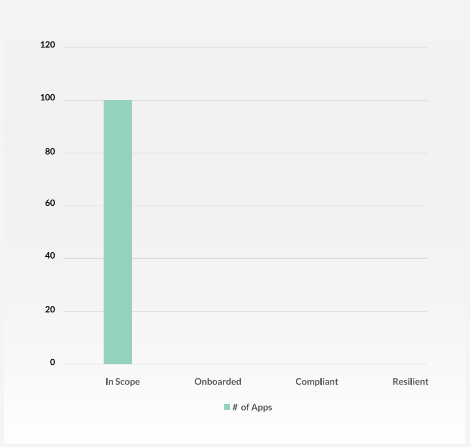
4

Triage and address findings in developer workflows

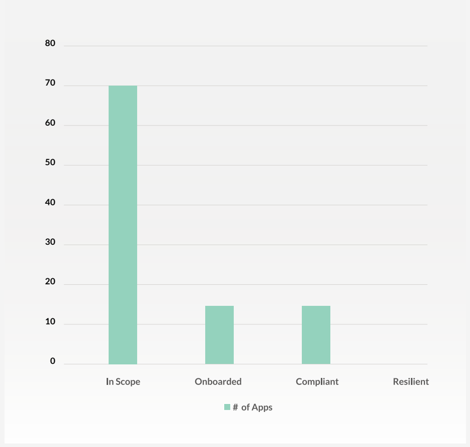
When Fail Fast onboarded applications, they found thousands of flaws that had accumulated in their codebase as security debt. They needed a plan for how to get all their onboarded apps to pass their new policy. They needed to prioritize developer time on high-impact efforts to bring legacy apps into compliance and successfully deploy new applications securely into production. Upon finding a policy-violating flaw, teams would process and resolve that finding through remediation or mitigation.

In order to tackle the security flaws in their old apps (while automated scanning and triaging happened concurrently in new builds), Fail Fast created simplified metrics over 30, 60, and 180-day increments through which they tracked applications in scope or discovered, onboarded, compliant, and resilient (apps built with security proactively built in from the start).

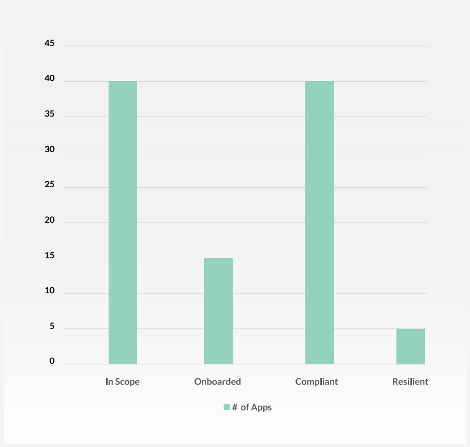
Simplified Metrics
Day 1



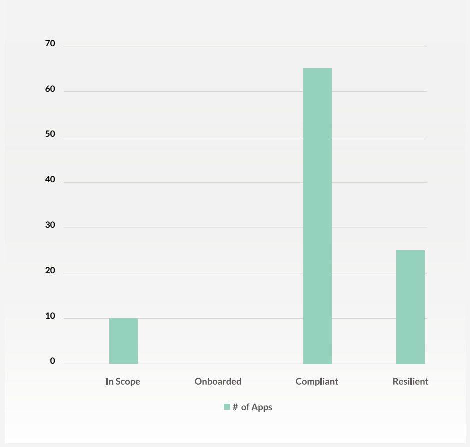
Simplified Metrics
Day 30



Simplified Metrics
Day 60



Simplified Metrics
Day 180 / 6 months



5

Leverage reporting to measure, manage, and improve outcomes

The simplified metrics used in Step 4 were a great starting place for measuring success, but the board wanted more information. Fail Fast knew that asking the right questions resulted in measuring the right things. They started with questions such as: How many flaws do we have? How quickly are we fixing them? Who is shipping the safest/riskiest code? How do we compare against similar organizations in the industry?

Fail Fast needed reports available on-demand so leaders and teams could see the real-time status and health of the application security program. That way they could establish a baseline, identify areas for improvement, set quantitative goals, and track progress against those goals. Beyond measuring, managing, and monitoring the health of their application security program, their reporting also needed to satisfy the demands of their board and regulators, provide security assurance to customers, and enable go-to-market teams to demonstrate and leverage security as a competitive advantage in-market.

Using Veracode's robust analytics – including patented peer benchmarking and the State of Software Security report with insight into macro trends across Veracode customers – they identified strengths and weaknesses, analyzed flaw and remediation performance, and prioritized high-value activities targeting specific application security goals.

6

Prevent security flaws via education and technical controls in the CI/CD process

With much of the automation underway, Fail Fast began working on ways to prevent security flaws from entering code in the first place. They brought security and development teams together to agree on mechanisms and controls that prevent security flaws in first-party code, shared libraries, third-party components, and open-source dependencies from reaching production. This was the beginning of fostering a secure coding culture.

Via technical controls in the CI/CD process, they established a continuous feedback loop that monitors compliance with policies and ensures that if code drifts from policy – or if any new flaws are created – teams are alerted, and the violations are addressed.

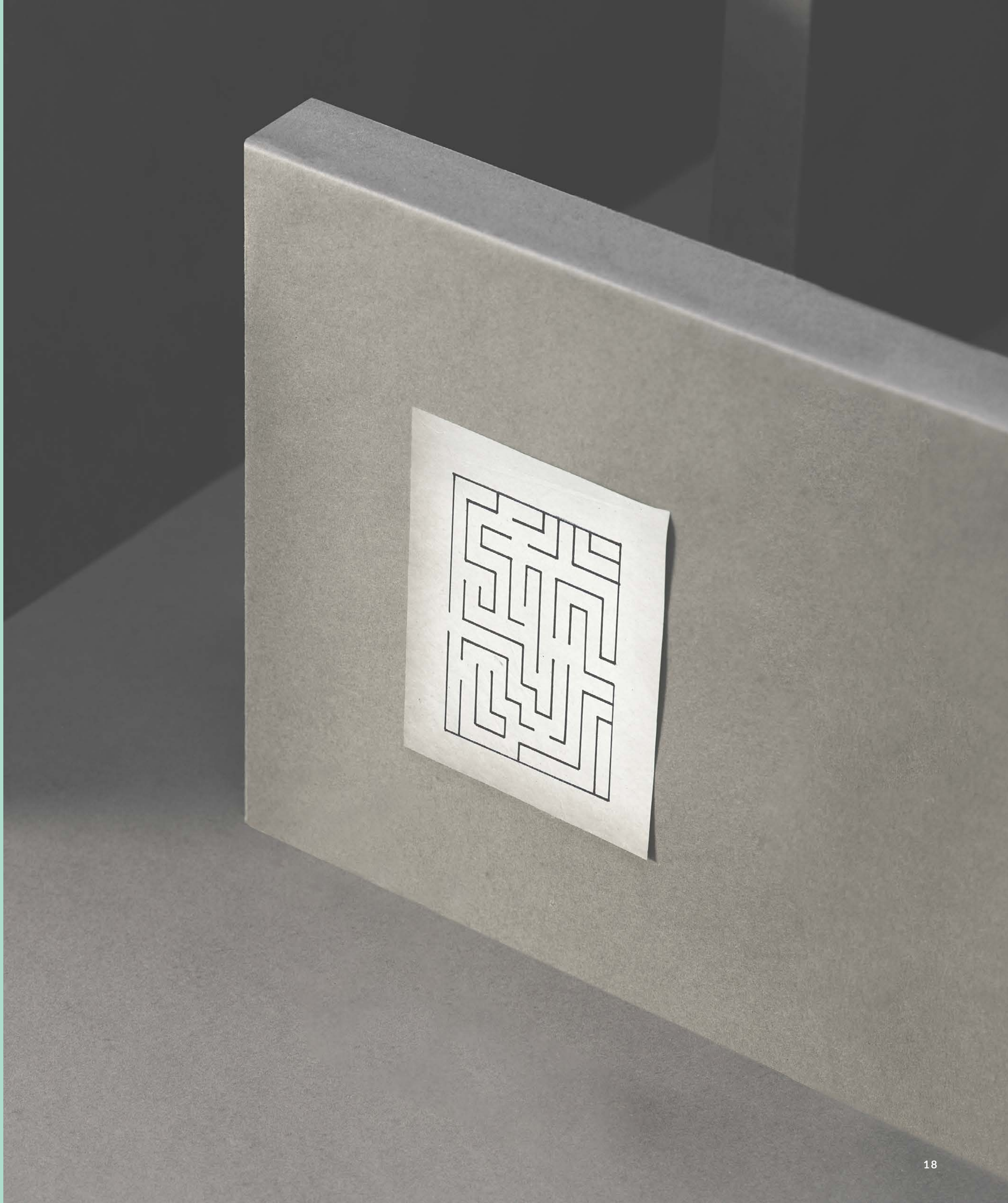
In addition to establishing preventive controls in the pipeline, they leveraged reporting to track the nature of issues being introduced and identify skill gaps. This information was used to design security programs and training curricula to target areas of concern.

They also provided developers with contextual – and intelligent – remediation tools to help them develop software that is secure from the start. With all the competing demands and concerns developers face, intelligent security solutions that let developers implement a code change or update a vulnerable library with a pull request liberates them from much of the time and effort of manually fixing flaws.

For example, they noticed a certain CWE causing many applications to violate policy, and they led a CWE burn down effort to focus on those findings. In another instance, one team was struggling to remediate a certain category of flaw within the grace period, so they designed a security training curriculum that provided the team with hands-on experience remediating code in sample applications.

In Summary

While the story of Fail Fast Technologies may relate to many organizations, all journeys to application security maturity are different. Different organizations have different needs and priorities. Some need to tackle seemingly insurmountable security debt in their legacy code base. Others prioritize rapid and secure development of cloud-native applications. Regardless of the journey, the path to maturity converges around core objectives: full visibility into the application portfolio with regular automated scans; all applications in compliance with a well-defined security policy; and the prevention of new flaws from reaching production.



Conclusion

DevSecOps is not a destination; it is a journey. Not a state to attain, but a process to enact and maintain. You must integrate and re-integrate security and continuously adapt to stay pervasive (but not invasive) as the development needs and tooling change.

No matter your journey, Veracode supports you every step of the way. The Veracode Continuous Software Security Platform enables you to establish comprehensive security around your legacy and cloud-native applications. We support you in seamlessly integrating security into your entire SDLC, bringing security and development together, and providing a vehicle to define and implement a set of security policies that align to the business criticality and operating environment of software in production. With Veracode solutions, support, and services, you can avoid and overcome the challenges of securing your software from start to finish.

"We chose Veracode because it was the easiest and best solution when it comes to integrating into our existing processes."
- Andrew McCall, VP of Engineering, Azalea Health

Click here to schedule a demo of Veracode today and let us show you how easy we can make DevSecOps for your organization.

Veracode is a leading AppSec partner for creating secure software, reducing the risk of security breach, and increasing security and development teams' productivity. As a result, companies using Veracode can move their business, and the world, forward. With its combination of process automation, integrations, speed, and responsiveness, Veracode helps companies get accurate and reliable results to focus their efforts on fixing, not just finding, potential vulnerabilities.

Learn more at www.veracode.com, **on the Veracode** blog **and on** Twitter.