Shifting Left for Application Security

Bridging the Divide Between DevOps and Security with the Right Security Tools

Peter Conrad

REPORT

Achieve Modern Application Security as Agile as Your DevOps Processes with F5 NGINX App Protect

Build Secure and Reliable Apps with Ease

Being agile is something every business strives for: adapting quickly to the latest trends, keeping up with competitors, and better serving your customers are all more essential than ever before. Modern apps are microservices that run in containers, communicate via APIs, and deploy via automated CI/CD pipelines. DevOps teams need to integrate security controls authorized by the SecOps team across distributed environments without slowing app release velocity or performance.

NGINX App Protect is a lightweight, high-performance, modern application security solution that integrates seamlessly into DevOps environments as a WAF or app-level DoS defense, helping your enterprise shift security left and deliver secure apps from code to customer.

- Lightweight, self-managed security solution for cloud, microservices, containers, and APIs
- Simple integration into the CI/CD tool chain, infrastructure agnostic
- Facilitates declarative policies for “security as code” enabling enterprises to shift security left
- Protect against Layer 7 Denial of Service (DoS) Attacks

Download a 30-day free trial today at:
nginx.com/free-trial-request/

NGINX
Part of F5

Shifting Left for Application Security

Bridging the Divide Between DevOps and Security with the Right Security Tools

Peter Conrad

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Shifting Left for Application Security
by Peter Conrad

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Jennifer Pollock
Development Editor: Gary O'Brien
Production Editor: Kate Galloway
Copyeditor: Liz Wheeler
Proofreader: Gregory Hyman
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Kate Dullea

April 2022: First Edition

Revision History for the First Edition
2022-04-06: First Release

This work is part of a collaboration between O'Reilly and NGINX. See our statement of editorial independence.

978-1-098-12732-9

[LSI]

Table of Contents

Preface

Security is important to everyone in application development and support, from design through deployment. Whether you're a developer, a security or operations engineer, or the CISO of a company, you're already thinking about security. To take on security holistically requires considering all available tools and their position in the software development pipeline. Shifting security left means bringing tools and processes to bear on security from the earliest phases of the pipeline, rather than bolting on a few security tests at the end.

This report will help you understand why shifting security left is important and how to do it. You'll learn some of the challenges facing organizations under pressure to deliver applications faster and more securely, how shifting left helps solve these problems, and the change in thinking that's necessary across the organization to make it all happen.

By the end of the report, you'll be able to make a few application security recommendations to your organization, and you'll have the tools to create strategies you and your team can apply to make security a priority for everyone. With the right relationships among teams and a shared set of security priorities, any organization can successfully strengthen its applications, services, and development processes by shifting security left.

Introduction

Shifting left is not a new idea. As the modern software pipeline has evolved to make everything continuous, shifting left has become the norm for various processes that were traditionally relegated to a testing phase between development and production. Shifting security left means implementing security policies, controls, and designs starting from the earliest stages of design and continuing through to production.

Despite broad agreement that shifting security left is a good idea, it can be hard to align on which tools and approaches are best suited to the task. A certain amount of the public discussion focuses on code- and container-scanning tools, automated patching, and other new security tools designed specifically for modern applications and infrastructure. Tools with a long history of protecting applications at runtime seem to have fallen out of fashion, labeled "legacy" tools that have no place in a modern software development environment.

These long-standing tools turn out to be important in today's enterprise. Tools such as web application firewalls (WAFs) provide runtime protection and valuable feedback on application performance that helps developers and security engineers refine both code and policies. WAFs and other tools have not stagnated, but rather have adapted to modern infrastructure and applications, becoming smarter as threat actors have developed more sophisticated attacks.

As the enterprise moves applications to the cloud, shifting left becomes ever more important. Modern applications, no longer monolithic, are composed of large numbers of services that present a complex, variegated attack surface that can't be defended with code scanning and good programming practices alone. Shifting left must

be about more than just security by design. The enterprise needs to execute an intentional cultural shift that makes security everyone's job, from design through production, driven by policies that are refined using feedback from runtime tools.

Traditionally, development and security teams have seemed to be at odds. Developers face pressure to deliver more features more quickly, while security is seen as a gatekeeper, sometimes halting development to investigate issues. The cultural change necessary to shift security left requires development and security to work together. This means new ways of working for everyone involved. Security must become an enabler of good practices, providing guardrails that keep applications and infrastructure protected without putting walls in the way of engineering. Developers must take on security as part of their role, from design through deployment.

Many factors can make such a significant cultural shift difficult, from entrenched processes, to budget, to internal politics. This report provides a survey of the current security landscape for cloud native application architecture, a few types of threats, and some strategies and tools for bringing a shift-left strategy to the continuous integration and continuous deployment (CI/CD) software pipeline.

CHAPTER 1
DevOps and DevSecOps

The traditional software development process separated designing, building, testing, and shipping into separate phases that focused on creating monolithic applications. It seems obvious in hindsight that this approach was neither nimble nor efficient. As a response to ever-increasing pressure to move faster, Agile methodology made the pipeline more flexible by breaking large tasks into smaller units. As the workforce became globally distributed, it was no longer practical to tie everyone to a single codebase. Application architecture moved from a monolithic model to microservices, which allowed dispersed teams to work on different parts without blocking each other. The challenge of managing the development process for increasingly complex collections of microservices has given rise to *continuous everything* in the software development pipeline. This is where DevOps comes in.

DevOps

DevOps is a philosophy and a set of practices that encompass several goals. As the name suggests, DevOps brings development and operational support together, blurring the lines of responsibility between the phases of developing, building, deploying, and maintaining applications. This breaks down boundaries between the teams responsible for these phases, enabling them to establish continuous integration and deployment of applications and features. Figure 1-1 shows the phases in a CI/CD approach to software development and deployment.

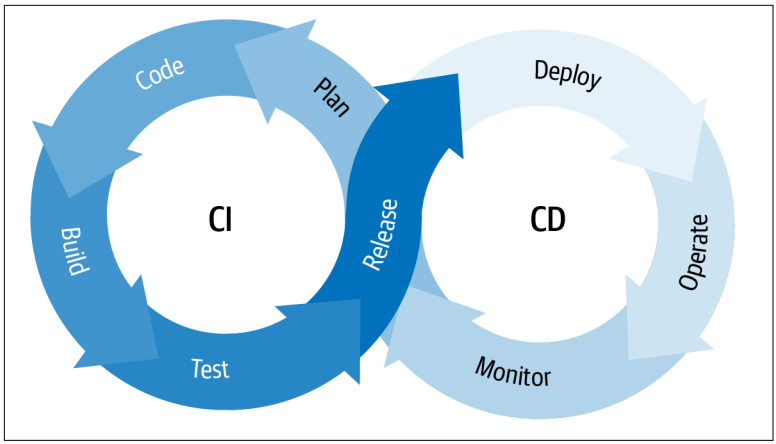

Figure 1-1. Software development phases in the DevOps CI/CD pipeline

Continuous integration means merging code changes to a shared main codebase as quickly as they're developed, keeping the codebase up-to-date with the latest of everyone's work so it can be tested and released as soon as possible. *Continuous deployment* automates the release process, including tests that ensure the final product is ready to deploy. By merging and testing changes constantly, the DevOps pipeline reduces the risk of integration conflicts from distributed teams working independently.

At its heart, DevOps is about breaking down barriers wherever they prevent continuous motion in the direction of quality. Testing and feedback loops become faster, partly through a shifting left of these steps to earlier phases in the development pipeline, meaning that developers begin to take on more responsibility for testing their own code. DevOps relies heavily on automation to reduce risk at every step while improving the workflow of coding, testing, and deploying changes.

DevOps brings faster development and deployment with higher quality, fewer failures, and shorter recovery times. One concern that DevOps doesn't explicitly address is security. The pressure to move ever faster has made it challenging for organizations to prioritize security, which is seen as a drag on the development process.

In the traditional waterfall development model, security was typically tacked on at the end. Figure 1-2 shows the waterfall approach, in which security often happens during the verification and maintenance steps.

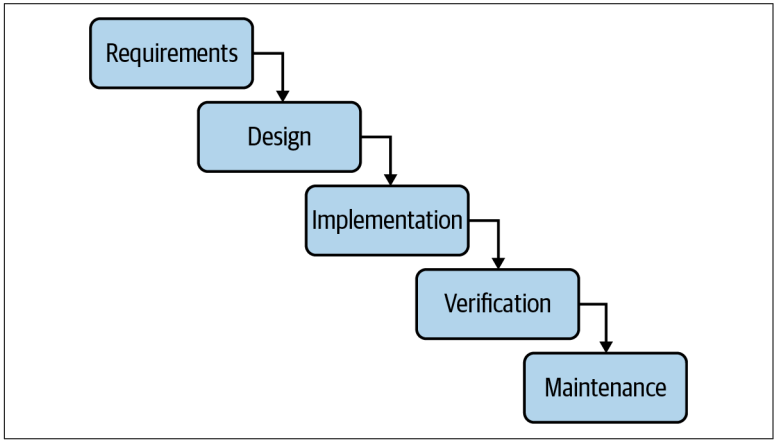

Figure 1-2. The traditional waterfall development model for software development

This approach to security wasn't effective in traditional monolithic software development, and it's even worse for modern microservices-based applications. A new way of thinking emerged, to integrate security throughout the development process: the evolution of DevOps into DevSecOps.

DevSecOps

When development and security are separate, friction is a natural consequence. Developers might see themselves as the drivers of change, and security as a constant impediment. Security teams are known to bring development to a halt to perform audits or investigate incidents. At the same time, security teams see developers creating or ignoring the same problems time and time again, unable or unwilling to adopt clear solutions.

DevSecOps integrates security as a priority within DevOps, placing it at the center of application development and building a security-first culture among everyone in the software pipeline. In the DevSecOps model, security is everyone's job. This is a first step in reducing friction between developers and security engineers.

Like DevOps, DevSecOps focuses on breaking down barriers between teams, making transparent communication easier. The goal is to build security into every product from the beginning, sacrificing as little speed and agility as possible during the development process. Just as DevOps gave developers more responsibility for testing their own code, DevSecOps makes building secure and compliant code every developer's top priority.

DevSecOps builds security practices into every phase of the application development lifecycle, providing feedback at each step. Security often starts even before the design phase, in the form of training to help developers learn secure coding practices. Security teams work together with developers, helping educate them, documenting security policies and best practices, and coaching everyone to adopt a security mindset. As an organization's DevSecOps practices mature, different teams begin to view themselves as part of a single culture with security as a central goal.

How DevOps and DevSecOps Are Changing the Organization

Whereas security was once an add-on feature at the end of development, DevSecOps has brought awareness that every component in an application can be vulnerable and must be secure by design. By bringing flexibility and transparency to the software development pipeline, DevSecOps has aided the move to cloud-first software, deployable in any environment. As more and more companies recognize the need to bring security into the core of development, DevSecOps is becoming mainstream. DevSecOps has proven to be a natural evolution of DevOps, just as DevOps naturally grew from Agile.

The cultural shift that DevSecOps represents is significant. Security and DevOps teams work with a common purpose: to bring high-quality products to market quickly and securely. Developers no longer feel stymied at every turn by security procedures that stop their workflow. Security teams no longer find themselves fixing the same problems repeatedly. This makes it possible for the organization to maintain a strong security posture, catching and preventing vulnerabilities, misconfigurations, and violations of compliance or policy as they occur. Developers, operations, and security teams work together on threat modeling, sharing knowledge to anticipate

and close potential weaknesses in both the product and the processes involved in its development.

Beyond a change in attitude, DevSecOps provides tangible benefits. By catching problems earlier, organizations deliver better, more secure products and services to their customers. The end-user experience is better when there are fewer urgent patches or unexpected breaches. DevSecOps finds vulnerabilities earlier and fixes them before deployment. This results in less downtime for customers, making it a more cost-effective for the enterprise.

The Challenges of Adopting DevSecOps

Adopting DevSecOps practices has clear advantages, but that doesn't mean it's easy at every step. From the day-to-day mechanics of securing distributed applications to the heavy lift of cultural change, bringing DevSecOps into an organization presents challenges.

Traditionally, security focused on a well-understood application perimeter, usually surrounding a single data center. As the enterprise adopts DevOps and cloud native, microservices-based architecture, applications—and security challenges—decentralize. These applications are composed of microservices running in multiple environments, communicating across many networks, working with data from devices and users all over the globe. This makes the attack surface both difficult to define and very large. It's nearly impossible to inventory all the interactions among services, or all the data transmitted across public and private networks.

At the same time, when application ownership shifts from teams to wider departments, it's easy for security to fall by the wayside. Individual engineering teams don't invest in security if they see it as a problem for a dedicated security team. This tends to push security rightward, toward the later stages of the software development pipeline, where security becomes more difficult and less effective.

Shifting left only works when developers understand security well. Not only do the developers need a good working knowledge of secure development practices, but they also need enough education to understand the issues they are tasked with fixing. This means training, which takes time and money.

The key to solving these problems is to create a culture of collaboration that supports rapid, continuous iteration with a security focus. That means many teams, once siloed, learning to work together: development, IT operations, and security. Rather than a continual interruption, security must become a practice incorporated in all aspects of work throughout the software development pipeline. Security teams must provide guidance rather than interruptions, becoming *continuous* just like the development and operations parts of the pipeline.

The good news is that the benefits outweigh the costs. When security becomes everyone's job, problems can be resolved earlier, with less expense, providing a better experience for everyone involved in the development pipeline—and for the customers.

CHAPTER 2
Shifting Security Left

As modern software architecture and the software supply chain have become more sophisticated, security threats have also continued to evolve. High-profile attacks have shown that vulnerabilities can appear at any point in the software development lifecycle (SDLC). In 2020, threat actors gained access to the build system at a software company called SolarWinds, maliciously modifying software updates that were then distributed to customers. The same year, attackers compromised the upload script of a code-scanning tool called Codecov, giving them access to environment variables on customer machines. Attacks like these, which use a company's supply chain itself to distribute malware, give threat actors access to infrastructure that belongs to the company's customers. Detecting attacks after the fact is no longer a practical solution. Only by integrating security throughout the SDLC can organizations protect themselves and their customers.

DevSecOps: Shifting Security Left

Traditionally, security and other tests were relegated to the end of the software development process, after the design and build phases. Only at the end of the process was security tested, the results determining whether software was fit for shipment. Shifting security left means embedding security by design throughout the entire SDLC.

In traditional waterfall development, software follows a linear path from design to deployment. By contrast, the continuous integration and deployment in a DevOps software environment mean that

everything is happening all the time. To understand shifting security left in the modern SDLC, we must temporarily unfold the infinity symbol of the CI/CD pipeline, as illustrated in Figure 2-1.

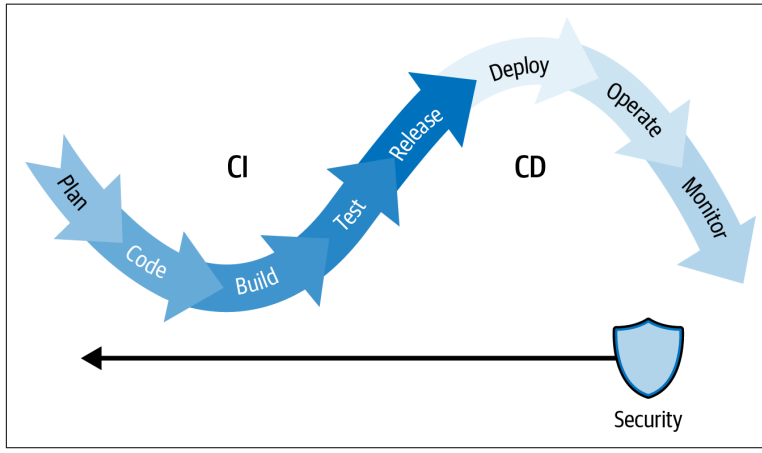

Figure 2-1. Shifting security left by embedding security into every phase of the CI/CD pipeline

Shifting security left involves strong security measures throughout the software supply chain, from planning all the way to the operation and monitoring of deployed software. Treating security as an integral part of the application design helps teams detect and fix potential issues earlier. According to Accenture, shifting security left can reduce build costs by 70% and halve the time to go-live. Earlier identification and remediation of issues strengthens application quality and security, prevents the need to design and apply workarounds for flaws after the fact, and eases the operation team's application maintenance burden.

Shifting left is not just about catching vulnerabilities earlier. DevSecOps is a security-first cultural shift that aims to prevent issues by design, while also assuming that vulnerabilities will always exist—what the Accelerate State of DevOps 2021 report, produced by the DevOps Research and Assessment (DORA) team at Google Cloud, calls a *diagnostic* approach. To quickly and automatically assess security, DevSecOps uses a number of tools as part of the CI/CD pipeline, including the following:

Static application security testing (SAST)
: Scans source code for weaknesses, providing early feedback to developers

Container image–scanning tools
: Automatically scan containers to identify vulnerabilities

Dynamic application security testing (DAST)
: Scans applications at runtime to detect issues that aren't apparent by scanning the code

Runtime application self-protection (RASP)
: Analyzes actions and blocks suspicious activity by applications

Web application firewalls (WAFs)
: Monitor application-level network traffic to detect and block attacks

In particular, WAFs can provide protection that is specific to an application. Historically, WAFs were used by security and network operations teams to protect applications running in production in a centralized data center. In a modern, cloud native infrastructure environment, this approach is not fast or efficient enough. It no longer makes sense to wait for an app to go into production, flag issues the WAF identifies, and fix them in the next version of the code. The modern development approach is to include the WAF earlier in the build and test phases, addressing issues as they come up. This means that the WAF itself must be capable of being shifted left.

WAFs can be configured with knowledge of the application's endpoints, APIs, and data, making it easier to detect anomalies while leaving legitimate traffic alone. This provides finer-grained security than policies that only address traffic at the protocol level. WAFs can also be aware of the users and devices using the app, and which privileges they require, making it possible to automate policies such as providing the lowest necessary privilege to each client. In this way, WAFs make it possible to shift security further left, both efficiently and precisely, providing protection tailored to the applications being developed and deployed. In turn, this enables the development of stronger apps that can be deployed in a broader array of environments. With some WAFs, you can encapsulate security policy as code, making it possible to build stronger policies that can be developed and evolved using source code management.

The Challenges of Shifting Security Left

Shifting security left involves both technical and cultural challenges in any organization. Different teams have different roles. Developers strive to move forward quickly, adding innovation to their projects, and security teams continually work to reduce risk, even if it means slowing things down. This can lead to friction or conflict. To alleviate these challenges, the CI/CD pipeline will need to adopt new processes and tools, and teams must learn to use existing tools in new ways. All teams involved must strive for the consistency, visibility, and transparency necessary to bring security to new parts of the SDLC.

In some organizations, different teams have a great deal of leeway in how they organize themselves and their work. While this provides flexibility, it can make it difficult for teams to shift security left consistently. It's difficult to bring a holistic view to applications and services that might not be developed uniformly across the organization. Visibility is key to security, especially in the realms of vulnerability detection and compliance. Without consistent visibility along the entire CI/CD pipeline, it is very difficult to shift security left in a meaningful way. As teams mature, they tend to align with organization-wide best practices, but when shifting left you can't wait for this to happen. It's impossible to implement good security in a project that doesn't already follow best practices. Every team must have good test coverage and high visibility into other teams' processes, so that developers can innovate securely together.

If the company culture is too formal about accountability, it might be hard for people outside the security team to take on additional responsibility for security testing early in the SDLC. A fear of blame when something goes wrong can lead people to duck responsibilities that they feel might result in undesirable consequences. This problem can be mitigated somewhat by automated tools, but it's still important for everyone involved with application development and operations to proactively take on security responsibility.

One pitfall when shifting left is a failure to engage early and strongly with the security team. The security team must be prepared to interact with a much broader group, many of whom might not have much expertise in security. Developers are not always accustomed to thinking of security as part of the development process. Without guidance from the security team, developers don't always have a

strong awareness of potential security risks. This often necessitates training and coaching, an additional burden of responsibility for the security team, and a potential point of friction between security and others in the organization.

If security is not given the budget to grow, the team might rightly feel they've been asked to do more with less. Worse, if development teams are inadequately prepared for these engagements, the security team might encounter resistance as they try to help educate others about how to build in security by design. The security team must be sufficiently staffed, prepared, and ready to engage with development and operations teams from day one. Agile methodology and DevOps have increased the speed of software development and deployment, requiring companies to move faster without sacrificing reliability and security. This has placed a heavy burden on security teams, and made security a bottleneck that throttles fast, iterative development cycles. Security is responsible not only for protecting the applications under development but the development tools as well.

Some tools familiar to security teams remain useful as security shifts left. Other legacy tools, which are not designed for automation and integration into modern software development infrastructure, are difficult to use in a shift-left strategy. Solving these problems takes time. This can perpetuate a perception that the security team is an impediment to a fast, efficient development pipeline, undermining the relationship that security is attempting to build with development and operations teams.

Why Shifting Left Is Important

Every interaction customers have with a service or application represents the company, building or destroying customer trust. The importance of steady, trusted interactions remains constant as threat actors get smarter and faster. In large organizations, there is a potentially vast landscape of vulnerabilities, any one of which can represent an existential threat to the organization.

While the applications and services under development are the most salient security target, software development relies increasingly on third-party, open source libraries and packages that can themselves contain vulnerabilities. Left undetected, these weaknesses can propagate throughout the software pipeline all the way to the

build and deployment phases, leaving the application or the development environment itself open to attack. Relegating security to a test process at the end of development would leave vulnerabilities open throughout the pipeline. Furthermore, the tools themselves can present attack surfaces. In large organizations, the number of installed tools can be sizable, and can include abandoned or orphaned tools whose only remaining function is to wait until their vulnerabilities can be exploited.

Shifting security left helps security keep up with the DevOps software development model while managing emerging vulnerabilities and risks. Shifting security left transforms it from a bothersome bolt-on to a design constraint, reducing the cost and complexity of fixing and preventing vulnerabilities.

Strategies for Change

The benefits of shifting security left are clear: catching problems earlier is more effective and far less costly than tacking security testing onto the end of a waterfall-style development process. To make it work, however, all teams involved must change the way they think about security, how they interact with each other, and how they respond to problems as they emerge. This cultural change is not a prerequisite for shifting security left, nor a result of it, but a corequisite; the cultural change drives the leftward shift, and vice versa. Here are a few steps you can take to begin the journey.

Define Your Strategy

The first step is to understand what shifting security left means to your organization. You must have a clear idea of what success looks like so that your teams know what to aim for. This means defining roles and ownership, milestones along the way, a way to measure progress, and a clear vision. To make the cultural changes required for success, everyone involved needs to align on common goals. For example, this means understanding the current threat landscape and how shifting security left can help.

Only by making everyone a stakeholder can your organization accomplish the most important aspect of the cultural change: to make security everyone's responsibility. The security team can help other teams build continuous security throughout the CI/CD pipeline from the beginning of the planning phase, before a single line

of code is written. Programmers must have security in the front of their minds as they implement and test every component. Good security practices must become second nature to everyone.

Understand the Supply Chain

Understanding the software supply chain at an organization can be more difficult than it appears. Often, different business units use very different software development processes and tools. Unless the organization is already moving in lockstep, it can be challenging to understand what dependencies each team incorporates in their builds, which tools are important and which have been abandoned, and how code flows from individual developers' laptops to the CI/CD pipeline. To shift security left means baking security into every aspect of the software supply chain.

Developers must become more aware of security risks at every stage of development. Selecting packages and libraries as dependencies, planning application architecture, deploying environments, writing functions, and running tests are all processes with security implications and needs. Developers must take responsibility for assessing and managing these risks, and for continuing to accumulate knowledge that helps them make their work more secure.

Provide Guardrails

It's not only developers who must adjust to new ways of thinking about security. The security team must shift from a gatekeeping mindset, in which they stop everything to troubleshoot or review a problem, to establishing guardrails that help development continue with fewer problems. By building new policies, tools, and recommendations, the security team can help developers keep moving safely. With automation and self-service, security can help developers help themselves. Security engineers can use their extensive knowledge about what attacks are possible to help developers build in security more effectively.

Automation and feedback are very important, because they enable developers to adapt and keep moving. Test-automation tools, code- and container-scanning tools, and automated integration tools take some of the burden off developers, while giving them the information they need to fix problems early in the development process and avoid introducing vulnerabilities into the final product. It's

important to have tools that provide sufficient automated feedback to enable security, developers, and operations teams to continually adapt.

Make Training Continuous

When security is a new responsibility for developers, there is a steep learning curve as they become aware of practices that they must adopt to create secure code. The first step is to assess the security knowledge of the development team, and to help educate everyone about secure coding practices. The security journey is never-ending. There is no "done" when it comes to learning how to mitigate threats and vulnerabilities. Developers must continue to learn as a part of their jobs.

As the security team teaches other teams about building security into the product from the start, they are learning too, discovering new kinds of threats and new ways of mitigating them. As the security team learns, they must share information with developers so that everyone's security practices continuously evolve.

CHAPTER 3

Application Security

Application security has evolved over the last several years as software architecture has changed. The main goal of application security remains the same: to mitigate vulnerabilities and other threats. Today's SDLC is faster, more iterative, and more distributed. With teams dispersed geographically and application components separated into microservices on cloud infrastructure, the network has become an attractive attack surface. As the complexity of protecting applications continues to grow, the pressure to move more quickly is unrelenting. These new realities of software development require new approaches to application security.

How the Security Landscape Is Changing

In the days of monolithic software development, security was mainly at the edge. Once a secure perimeter was established, defending it was straightforward. Planning, coding, and testing an application were sequential phases, with security often applied only at the end.

In a modern software development pipeline, everything moves much more quickly. The development phases are iterative, concurrent, and continuous. Infrastructure itself, which used to mean physical hardware that had to be planned, ordered, delivered, and installed, has become virtual and fungible. Provisioning new hardware is essentially instant, meaning that organizations can scale up and down quickly as demand changes. Monolithic architecture has given way to collections of microservices, developed in pipelines

that release code many times per day. All these factors add up to rapid, unceasing change at all levels.

The cloud infrastructure presents a broader, more complex attack surface. Only a few years ago, it was reasonable to assume that infrastructure and network boundaries were stable and well defined, presenting a defensible perimeter. In cloud or hybrid environments, this is no longer true. These boundaries have become fuzzy, with many points of access and ingress that can change at any time. Nearly any resource can be made public or private with a few configuration changes, and sensitive data is transmitted routinely across public networks. Containerization turns monolithic applications into large collections of services available on the network, each with its own perimeter.

DevOps is dragging security into a new model, focusing on transparency, immediacy, agility, and continuous integration. Shifting security left entails building and using security tools and controls throughout the pipeline for applications, containers, microservices, and APIs, relying on automation to maintain consistency and keep up with the rapid pace of development. This *security-as-code* approach, like infrastructure-as-code, must use declarative policies to maintain the desired security state, getting out of the way of innovation.

Threats

Because applications and services rely so heavily on distributed infrastructure, the network itself has become the critical attack point. Layer 7, the *application layer*, is a target of particular interest. The complex communications among interconnected services and systems on layer 7 make it difficult to secure. Ironically, secure transport protocols such as HTTPS and Transport Layer Security (TLS) can actually help attackers by hiding payloads, making certain kinds of attacks more difficult to detect.

Denial of Service Attacks

The new frontier for attacking the application layer is an old kind of threat: the denial of service (DoS) attack. Layer 7 attacks are relatively easy and inexpensive, because all that's needed is the ability to make HTTPS requests or API calls to the application. Traditional defenses leave the application layer open to these attacks by design,

since they are the mechanism by which a legitimate client uses the service or application.

Modern DoS attacks usually take one of the following forms:

`GET` *and* `POST` *flood*
: The attacker overwhelms the service with a large number of requests

Slow and low
: The attacker slows the server down by consuming resources sluggishly

TLS attack
: The attacker ties up the TLS server with invalid messages

In `GET` and `POST` flood attacks, the attacker sends so many requests that the service can't respond to real users, often using a network of hijacked servers or internet-connected devices to do so. In slow and low attacks, the attacker sends legitimate requests so that the service keeps the connection open, but then slows down, tying up server resources. In a Slowloris attack, for example, the attacker sends partial request headers slowly, saturating the service's pool of available connections while the service waits for the remainder of each header. In a TLS attack, the attacker bottlenecks the entire network by targeting the network traffic security mechanism itself. In all these cases, the goal is the same: to make the application or service unavailable to real users.

DoS attacks can be difficult to detect. Attackers can use encryption or network address translation (NAT) to disguise the origin of their attacks, making it more difficult to detect or filter malicious traffic. Because payloads are encrypted, it's difficult to differentiate attacks from legitimate traffic except at the endpoints. One tactic to defeat such attacks has been *rate-limiting*, setting an upper limit to the number of requests allowed through. However, attacks on layer 7 can be damaging with a lower volume of requests than DoS attacks on layers 3 and 4, which deal with networking and transport respectively. This means that rate-limiting alone is not enough to mitigate them. Security tools must be able to recognize the difference between legitimate traffic and attacks reliably enough to know quickly when an attack has started.

Attackers have become more sophisticated, using AI and machine learning to make their attacks more difficult to detect. Security based on static rules can't keep up with changes in attack strategies and the changes in application architecture that make new attacks possible. Detecting attacks against an application requires insight into the behavior of the application itself, establishing baselines to help determine which traffic is legitimate. This means that every application, service, and endpoint must be protected individually with tailored tools that are aware of the application or service API.

Other Threats

Although DoS attacks represent an easy exploit of the application layer attack surface, subtler threats are possible as well. In many cases, these attacks take advantage of failures to protect data or code. Authentication and access control attacks, for example, often work by modifying session identifiers or other parts of a request, by exploiting weak passwords or password recovery procedures, or, in the case of brute force attacks, by guessing valid credentials after some period of time. Injection attacks slip past data validation deficiencies to insert malicious SQL commands, scripting, or data into an application or service.

In some cases, threats come from within by accident. Misconfiguration of security controls, use of outdated or vulnerable packages as dependencies in the build process, and insecure design can open vulnerabilities in the application without anyone being aware of them. For this reason, it's important to employ automated tools at all access points on the network, providing redundant security checks.

The DevSecOps Approach

The challenge DevSecOps must address is to shift left with a built-in security solution that can keep up with the velocity and complexity of modern cloud native application architecture. Tools and processes are emerging to integrate security earlier in the SDLC without slowing the DevOps team down. In a way, the move to DevSecOps is not much different from the Agile, integrated testing, and continuous integration shifts that came before.

Shifting security left helps protect applications by making security an integral part of every phase in the workflow so that engineers can catch vulnerabilities before they become problems. In this model,

the developer's first step is to look at the proposed architecture of an application, service, or feature with a security lens. From the start of design, it's important to look at what ports and components are available, what information is to be transmitted or exposed, and how this data is to be protected both in transit and at rest. These issues and others must remain at the forefront throughout the development pipeline. Fortunately, there are tools and techniques to help make the process easier. These include automated analysis tools, policy management strategies, defense in depth, and the proper use of legacy security tools.

Automated Analysis Tools

There's no replacement for human expertise, but automatic code-scanning and security-testing tools bring consistency, speed, and relentlessness to the search for security issues. Along with code audits and reviews, automated analysis tools help prevent and detect vulnerabilities in the codebase, the dependencies, and the deployed application or service.

SAST and DAST tools can automatically detect some security problems in applications and services under development. SAST examines the application from the inside out, scanning the source code for weaknesses such as potential injections, authentication vulnerabilities, and other exploitable flaws. SAST happens early in the CI/CD pipeline, before the application is built and deployed to staging or test environments. SAST works by analyzing how well the source code complies with a set of coding rules designed to provide strong security protections. DAST works with the running application, testing it from the outside in with no knowledge of the source code or frameworks on which the code is built. The intention of DAST is to take the attacker's approach, looking for a way in through a security hole. DAST runs as part of the application testing, and can often surface coding defects that SAST doesn't find.

Although SAST and DAST help secure the application codebase, these tools don't directly address potential vulnerabilities in third-party libraries and packages. These upstream dependencies can either be specified directly by the build process, or indirectly by other dependencies, making it difficult to keep an inventory of all the packages included and the vulnerabilities they might contain. Software code analysis helps track dependencies, detecting which

components contain known vulnerabilities and providing information to help developers mitigate them.

Policy Management

Security policies help protect the environment where applications and services are deployed. The goal of a security policy is to allow valid traffic, usage, and requests while blocking threats, abuse, and malicious misuse of a service or application. It makes sense to model security policy on the intended legitimate use of an API, for example, so that policy management tooling can properly enforce the goals of each policy.

By managing policies declaratively, taking them directly from API schemas, it becomes possible to represent the API contract precisely in security policy. When the schema changes, the policy changes accordingly. In this way, policy management becomes automated, versionable along with the API schema, and effectively managed in source control; because the policy is derived from the API itself, it becomes effectively policy-as-code. This makes it easier to define an individual policy for each service, using the control plane to manage them.

Defense in Depth

Any single layer of protection can fail, letting attacks through. For example, during a DoS attack, the first layer of defense is to block requests from IP addresses known to be controlled by the attacker. As we've seen, attackers can use encryption or NAT to hide their true origin, or a network of hijacked devices to attack from many IP addresses at once. A second layer of defense can take an additional step such as blocking requests that appear to share characteristics with the attack. This strategy, layering protections on top of each other so that a failure in one layer is covered by another layer, is called *defense in depth*. Defense in depth helps prevent attackers from breaching a system in the first place, but also provides layers of defense within the system, reducing how far a successful attacker can explore once they get inside.

Shifting Traditional Tools Left

Modern security tools designed for cloud native architecture and infrastructure are proactive and intelligent, leading many to ignore tools that were originally designed to protect a simple perimeter around a monolithic application. So-called "legacy" tools are evolving and adapting to the new ecosystem, used in new ways to protect clusters, containers, and microservices by enforcing runtime security policies.

For example, a WAF is a powerful tool in protecting against application layer attacks. After all, if a WAF can protect a large perimeter, many WAFs can protect many small perimeters. To be effective, the WAF itself must evolve and shift left. The WAF must be lightweight, easy to deploy, and adaptable for tailored deployment to protect different components in the application, the cluster, and the software development pipeline itself. For a cloud native application, each WAF must be aware of the API of the service it protects, preferably by consuming the schema from source control directly. To fit into a modern CI/CD pipeline, the WAF must make it possible to automate security policies declaratively so that developers and security teams can treat security as a provisionable resource rather than something that must be manually configured.

CHAPTER 4
Scenarios: Making the Cultural Shift

Moving toward cloud native containerized architectures takes time. Many enterprises still rely on legacy applications, infrastructure, and traditional waterfall development. This often means a "bookend" security model: defining security requirements at the beginning of development, performing penetration testing in production, and not much in between. Leadership may understand that this approach doesn't scale, but moving to DevSecOps is complex, requiring changes in how different teams relate to each other.

Often, the enterprise starts by building static code-analysis and composition-analysis tools into the development process. This checks the box of building security into the pipeline but falls short of a holistic solution. The responsibility for protecting the code still falls squarely on the security team. They may provide feedback to the developers, but the relationship is not as collaborative as a true DevSecOps approach requires. Only when a true cultural shift makes security everyone's responsibility is end-to-end accountability possible. The result is the ability to deploy a mix of tools, including traditional security tools, throughout the pipeline from design to runtime.

This last point is key: no matter how good your vulnerability detection is during the development process, you still need strong runtime security. Even with security built into the design, it's impossible to catch a DoS attack early in the pipeline. Tools like WAFs, which

can be inserted throughout the process, provide a last line of defense and a strong source of feedback to help teams collaborate better on security.

While the cultural shift requires cooperation by all teams, it's often the security team that must make the first move. By becoming more transparent, security teams can overcome their reputation as gatekeepers. Managing threat modeling and security policies as code helps make their initiatives and tactics accessible to engineers, promoting collaboration. The engineers know how the application has been designed, but security has spent time observing how it performs in production. Automated tests, WAFs, and other tools can help validate that the performance of the application meets the design requirements, which in turn makes it easier to ensure that security is built in rather than bolted on. One key to a successful shift is to cultivate a few champions who can show others the importance of shifting security left.

Example: A Large Bank in Europe

Faced with nimble competition, a large bank in Europe needed to build new infrastructure quickly to move their many data centers and employees to a modern application architecture while maintaining compliance with a host of financial and other regulations. Though they had firewalls and other security measures in place, their security and development teams were only loosely connected. Testing new applications usually only meant that the software was installed on a virtual machine (VM) and an email sent to the security team asking them to test it. Teams within the company had their own budgets, didn't talk to each other much, and didn't get along.

Adopting automation was a necessity, but the question of how to do it created a great deal of internal resistance. Security didn't have the budget for the tools. The application developers didn't want to take on the burden of a CI/CD pipeline. The network team didn't want the responsibility of owning central automation that affected all the different teams. No one wanted to take on the job of managing tools for everyone else. At the same time, no one wanted to lose people, power, or budget. The result was an internal cold war.

The executive team understood the value of end-to-end security and gave more power to the application developers, asking them to work side by side with security on building a modern application. The

shift involved moving from VM-based architecture to orchestrated containers. As they built a true DevSecOps pipeline, they selected WAFs and other tools that they could control and operate with representational state transfer (REST) APIs, helping them automate security and infrastructure. The overall result was greater collaboration and transparency across the organization.

Example: A Bank in Australia

Around the world, banking is heavily regulated and highly competitive. Customers sometimes choose a bank based on how quickly it can respond to a loan application. This fast pace is driving a strong push to cloud native architecture to serve banks' many customer-facing channels and applications. The biggest internal challenge is getting applications to production quickly. The days of architecting an application for months and testing only at the end of the process are long gone.

To reduce friction, teams are moving to the cloud, working to manage infrastructure and security as code so that releases can be automated. Facing such a shift, the security team at one prominent Australian bank realized quickly that a gatekeeping posture doesn't work in the cloud. Unlike on-premises infrastructure where security has a great deal of control, the cloud makes it possible for teams to spin up whatever they need without the security team's knowledge or involvement. In order to continue providing value to the organization, the security team understood that they needed to collaborate closely with other teams.

Catalyzed by the move to the cloud, the security team took an active role in managing WAFs and other tools in the application runtime environment, building a feedback pathway back into the pipeline to inform changes to the design and build processes. The security team helps manage WAF configurations and other policies as code, making releases easier. The result is better cooperation among teams, helping the bank compete more effectively in a fast-moving marketplace.

Example: An Energy Company

In many enterprises, the rift between application developers and other teams is the main obstacle to cultural change. A large energy company in Europe faced an additional schism. Owned in a public/private partnership, the company's goals were often split. Those on the private side, holding the purse strings, were usually the decision makers. The public side, accountable to the citizenry, was shackled in regulations and bureaucracy. This made a move to DevSecOps especially challenging, because shifting security left involves such tight cooperation among different teams. Adding to the friction was the fact that the decision makers often weren't technical, which sometimes made it difficult for them to see the value in technology options. In fact, at the beginning of the process, the company had not adopted VMs, running instead on bare metal servers.

The main impediment to adopting cloud native DevSecOps was inertia. Many people wanted to keep things as they were rather than changing things they felt were already working. The DevSecOps story was hard to sell inside the company to people who didn't want to lose their power or budgets, didn't understand the technology, and didn't care about features such as REST APIs and automation. The company knew it had to adapt to the times, but bureaucracy and inertia kept them where they were, without a roadmap for change. Eventually, the company hit the limits of the existing tools and was forced to look at new solutions.

The company began to develop a list of requirements, including telemetry, customization, and programmability. Champions within the company had to work hard to help the decision makers understand the capabilities and limits of the technologies. Ultimately, what made the difference was having conversations with other companies who had adopted DevSecOps tools, including shifting WAFs and other tools left in the pipeline. In the end, the move to DevSecOps was not about getting security and other teams to work together. The solution was educating the decision makers about why end-to-end security is important and how to implement it.

Example: A Telecommunications Company

The telecommunications world is always moving. As bandwidth has become a commodity, telecommunications companies must stay competitive by branching out into add-on products without losing focus on supporting network infrastructure, service delivery, and other IT goals. The transition to 5G is pushing telecommunications companies to adapt and move faster, and driving their adoption of cloud native architecture.

The cultural change in a telecommunications company can sometimes happen faster than in a bank or an energy company, because the security team is smaller. A telecommunications company might have a security team of 30 people, whereas a bank's security might be 20 times that size. With a small security team, the challenge shifts to educating the application developers, who might see security as an afterthought. A knowledgeable champion can help engineers understand the problems that result from releasing code without the right level of controls in place.

Building security into the automation pipeline as the company transitions its architecture to cloud native helps reduce friction. By adding runtime controls and commit-time code scans, the company can begin its journey to a mature DevSecOps pipeline. Continual education can help the application developers understand that end-to-end security is everyone's responsibility.

Conclusion

Shifting security left for modern cloud native applications is not just about a specific technology or practice. Although it's important to build security in by design, a holistic approach requires much more: thinking like a threat actor, considering all available tools, and making a decisive cultural shift to build bridges among security, operations, and engineering teams.

Thinking like a threat actor means acknowledging that no matter how much security is built in, vulnerabilities will always exist. Throughout the design process, threat modeling helps everyone understand, predict, and mitigate vulnerabilities, but it's important to protect and monitor applications—and the tools used to build them—at runtime as well. This is especially important at the application layer of the network. The complexity of communication among interconnected systems and services makes the application layer both difficult to secure and attractive to threat actors. Threats such as DoS attacks can't be designed out of an application, and can require a great deal of diligence to detect and mitigate. When a DoS attack is thwarted, the attacker can pivot slightly to evade security controls, then continue the attack. As attackers become more sophisticated, DevSecOps efforts must become smarter as well.

SAST and DAST tools are good ways to catch known security issues during testing. Container-scanning tools can identify additional vulnerabilities. In production, tools such as WAFs can be tailored to protect individual applications, services, or APIs. In this way, WAFs can detect anomalies and attacks without disturbing legitimate traffic, while providing feedback about application performance and threats. This helps inform both stronger policies and refinement

of the code itself, enabling the development of more secure applications that can be safely deployed in a broad array of environments.

Central to shifting security left is a cultural change that makes everyone a stakeholder in security throughout the development process and beyond. Teams across the organization must take ownership of security, changing how they respond to issues and the way they interact with other teams. It can be challenging to get all teams aligned on a security vision, but defining success is a crucial first step. Everyone needs to understand the threat landscape, the state of the application in development and production, and everyone's role in meeting common security goals. Developers, operations teams, and security engineers must become more aware of one another's practices, processes, and concerns.

The security team must be willing to shift from a gatekeeping mindset to providing guardrails that help everyone avoid problems. Policies, recommendations, and tools—including the effective use of feedback from WAFs—can help developers move forward safely. In turn, developers must take on their part of the security journey, understanding the risks along the way. From the design phase, through selecting dependencies, through building and testing, security must be at the top of their minds. As developers learn how to build applications more securely, and as threats evolve, ongoing training is a must. Security never stops, and learning must be as continuous as other processes in the CI/CD pipeline.

Changing the culture in an organization is never easy, and it is different from one company to another. In some cases, the challenge is educating leadership about why security is important. In other cases, the first step is to heal a rift between security and developers. No team wants to lose their power, their budget, or control over their own roadmap. Security can feel like an imposition, or even like an attack. Security champions can help build understanding, locating sources of friction and smoothing them. Ultimately, a true leftward shift in security can only happen when everyone works together.

About the Author

Peter Conrad is a technical writer with diverse content development experience, ranging from consumer electronics and telecommunications to IoT and enterprise software, in both hardware and software environments. He has cultivated strong interpersonal skills from interviewing and collaborating with diverse subject matter experts; from showcasing technology effectively for different audiences; as a liaison obtaining reviews and approvals of documentation; and as a user advocate.