The DevOps Roadmap for Security

How Security can take on a new role in the DevOps movement by embracing modern principles, practices, and tooling

By James Wickett

Signal Sciences

Signal Sciences

The DevOps Roadmap for Security

By James Wickett

Published by Signal Sciences, 8520 National Blvd, Suite A, Culver City, CA 90232.

VERSION INFORMATION

This is the all-new, 2nd Edition of the The DevOps Roadmap for Security. This document was last updated on August 15, 2018. The 1st Edition of the The DevOps Roadmap for Security was released on July 29th, 2016. Some changes in this updated second edition were made to chronicle changes to DevOps particularly in practices that get placed in the domain of DevSecOps. New data and metrics from recent reports, like the DevSecOps Community Report have been added as well as quotes from practitioners in our field. Also new and updated in this version are numerous graphics throughout.

SPECIAL THANKS

Signal Sciences, PagerDuty, Pivotal, BugCrowd, Threatstack

Table of Contents

"Security is joining forces with DevOps and this paper shows you how to get started with common principles and practices to effectively integrate security in your DevOps transition, written by some of the best in the game."

Gene Kim, co-author of *The Phoenix Project* and the *DevOps Handbook*

Introduction

Information security is in crisis. We see it in the murmurs of product teams, we see it in the countenance of other infosec professionals as they pass us in the hallways. Much worse, we see it in the headlines which serve as a nagging reminder that no matter what we do, we have an inability to deliver software without vulnerabilities. This isn't a crisis that has sprung up all of a sudden, but a long-standing, systemic outpouring of the practices and policies that security has built over decades of misalignment inside organizations large and small.

In the book *Agile Application Security*, the authors point out that "many security teams work with a world-view where their goal is to inhibit change as much as possible."[1] When was the last time you heard of a business touting that it inhibited change as a competitive advantage? Of course, never. Inhibiting change shows a disconnect between security and the reality of modern software delivery practices. Is it any wonder that security is often the most disliked group in organizations and is facing a crisis among its ranks?

This aversion to change would be very forgivable if it actually made software safer. If organizations with security teams that inhibited as much change as possible stood up and announced that, through their tough posture on change, they were delivering their software vulnerability-free, then this aversion to change would be more forgivable because it would result in safer software. The problem is that this just isn't the case.

In his latest book, *Thinking Security*, Steven Bellovin writes, "Companies are spending a great deal on security, but we still read of massive computer-related attacks. Clearly something is wrong." We see this truth in the daily media deluge of security breaches and the headline-making by the failings of major companies, some of which even specialize in security. Bellovin goes on to highlight the same point that the authors of the Agile Application Security share. He writes, "The root of the problem is twofold: we're protecting (and spending money on protecting) the wrong things, and we're hurting productivity in the process."[2]

Yet, all is not lost. It doesn't have to be this way. In fact, there are many organizations that are integrating security with business outcomes in mind. This is often done under the banner of DevOps or DevSecOps. The DevSecOps movement represents the joining of Security to DevOps, because it turns out the two have more in common than people think, and the organization as a whole benefits immensely from the outcomes of their collaboration.

Where DevOps Fits In

> "Security is not only failing to protect, it's also hindering the organization's productivity."
>
> **Steven Bellovin**

DevOps is a culture, movement, and practice that enables collaboration between development and operations teams throughout the entire software delivery lifecycle, from design and development to production support. It breaks down entrenched silos, allowing organizations to transition from functional area delivery to a more holistic approach. This results in robust processes, exponential improvements in deployment times, and ultimately, superior results for a company's bottom-line. Since DevOps was first coined in 2009,[3] it has been a massive movement among engineering focused organizations. In the 2015 State of DevOps report, it was found that high-performers were able to deliver thirty times more frequently—rapidly decreasing the time from concept to cash.[4] [5]

DevOps is transformational in four key transformation areas:

1. It creates feedback loops in the runtime environment to inform development and operations
2. It treats all systems and infrastructure as code
3. It shifts engineering culture towards total delivery and user experience.
4. It favors a faster delivery cadence and a reduction in changes per delivery.

In each of these areas, there's a common body of principles, practices, and tooling that's rapidly evolving. The DevOps Roadmap for Security will help you navigate these areas and suggest ways for security teams to get more involved with DevOps.

Why DevOps Matters to Security

In early 2015, Gartner, a leading market research firm, stated that: "By 2016, DevOps will evolve from a niche strategy employed by large cloud providers to a mainstream strategy employed by 25 percent of Global 2000 organizations."[6] It's safe to assume that if you aren't considering DevOps now, the market may soon decide for you. According to a survey and published report by CA,[7] there are five key benefits to DevOps adoption:

- **Provides new software or services that would otherwise not be available.** This arms organizations with a new playbook for cloud delivery, microservices and software as a service offerings.

- **Reduces time spent fixing and maintaining applications.** DevOps practices have proven to require less break-fix work and decrease the mean time to recover (MTTR) from outages.

- **Improves cross departmental collaboration.** DevOps enables collaboration across functional silos. Organizations that adopt it are witnessing the benefits first hand.

- **Increases revenue.** In his book, *Leading the Transformation: Applying Agile and DevOps Principles at Scale*,[8] Gary Gruver directly tied DevOps transformations to bottom-line impact in HP's printer business by reducing costs by $45MM and freeing up 35% capacity for new innovation. This is a significant impact to the business.

- **Improves quality and performance of deployed applications.** One other tangible outcome of DevOps is a reduced failure rate. The 2016 State of DevOps report showed that high-performers had a 3X lower rate of failure.[9]

The benefits of DevOps are clear for organizations of all sizes, and the adoption rate suggests that even more evidence will be forthcoming.

Unifying the Tribes

The term "tribes" is used throughout this report. To add clarity, tribes are simply groups of people found in organizations that rally together around common concerns. DevOps is concerned with uniting two particular tribes: development and operations. These tribes have seemingly opposing concerns: developers value features while operations value stability. However, these contradictions are largely mitigated by DevOps. A strong argument could be made that the values of the security tribe—defensibility—could just as easily be brought into the fold, forming a triumvirate under the DevOps umbrella.

The security tribe's way forward is to find ways to unify with DevOps in its four key areas: (1) create feedback loops, (2) unite security and engineering culture, (3) enable delivery velocity, and (4) treat everything as code. Each of these areas will be covered in the following sections of this report.

TRANSFORMATION AREA 1

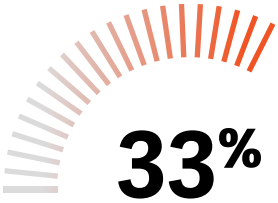
Create Feedback Loops

For security, the worst feedback loop is the breach feedback loop—the one where your company's name is in the headlines for the wrong reasons. Companies want to avoid the breach feedback loop and DevOps teams see this as part of their mission. The DevSecOps Community Survey 2018[10] found that mature DevOps teams are very concerned about high-profile breaches, with 73% of respondents indicating breaches as a driver.

What is interesting about many high-performing DevOps teams is where and how they're layering in defenses. The first place where many DevOps teams are focused on is defending the application layer. The DevSecOps Community Survey 2018 found that one-in-three companies experienced a web application security breach in the last 12 months. Across the board, the industry is seeing the attack surface move to the application.

Since the web application is the modern attack surface, it's often the best place to start for instrumenting feedback loops. The kind of feedback loops we want to create are the ones that connect application runtime in production to development. This way, when breaches occur—or begin to occur—automated defenses are triggered, or development staff is notified and ready to respond.

Feedback loops aren't a new idea—they're almost so inherent, so human, that it feels odd to specifically call them out at times. From human relationships to complex industrial systems to military strategy, feedback loops are foundational. In military strategy, there are Observe, Orient, Decide, Act (OODA) loops[11] and in the best selling book *The Lean Startup*, the feedback loop is identified as the Build-Measure-Learn cycle.[12] Yet somehow, software development struggles with defining feedback loops. It's still common, though increasingly less so, to have production software where users report outages before the development, operations, and security staff are even aware of it. If security is to be successful in the new, shorter DevOps cycles, feedback loops have to improve.



reported a web app breach or suspected breach within the past 12 months.

Once an organization has shifted thinking and processes to orient around a fast delivery cycle, the security team will need to quickly put feedback loops in place. Gaining insight into the rapidly-changing runtime environment gives security the ability to collaborate with development and operations to respond to an event before it becomes a threat.

A Defensive Thinking Approach

At the risk of over-simplifying security concepts, defense requires knowing answers to two first-order questions:

- Am I currently being attacked?
- What vector of attack is being attempted?

This is further complicated by second-order factors such as analyzing the likelihood of success and determining the potential cost of compromise. The security industry at large generally isn't equipped to address these questions due to a lack of first-order data—namely their limited insight into the frequency and types of attacks. Surprisingly, most organizations can't even approximate an answer to these questions.

THE SECURITY TRAINING FALLACY

It's common for application security teams to see application-level vulnerabilities like XSS or command execution and turn to developer training as a solution. While training is a good thing, it cannot replace a true feedback loop to developers. A feedback loop that works is one that instruments the application runtime and involves developers in security events as they're actively happening. This moves application security from a push model to a pull model which is considerably more effective for developers.

In a DevOps context, there are three areas where security provides direct value to the enterprise, utilizing value from integrations across the organization. Each of these areas can give insight into the first-order questions, and through instrumentation, can shift to a defensive thinking approach. When providing security defense in a web context, there are two key areas to evaluate: application security and usage feedback.

Application Security Feedback

It's hard to think about modern approaches to delivering services without thinking about delivering them over the web. With the rise of microservices[13] and the decoupled architecture patterns therein, you find an even higher dependence on web-based REST APIs. Today, most systems are collections of loosely coupled applications delivered over the web. This hasn't been an abrupt transition but has been an ongoing shift over the last 20 years. But even with a long history of using the web, we have a dearth of mechanisms for detecting security problems in real-time.

Many organizations implemented web application firewalls (WAFs) a decade ago, however rarely has anyone operationalized them. Most WAFs were put in place for compliance adherence, namely PCI, and were generally put in "listening," or passive mode with no defensive posture. However, in the last ten years we have continued to see common web application security vectors get compromised, and the Open Web Application Security Project (OWASP) continues to issue guidance on the same threat vectors.[14] The problem hasn't been solved. We are clearly lacking feedback loops to improve on our application security stance in the face of changing underlying technology models.

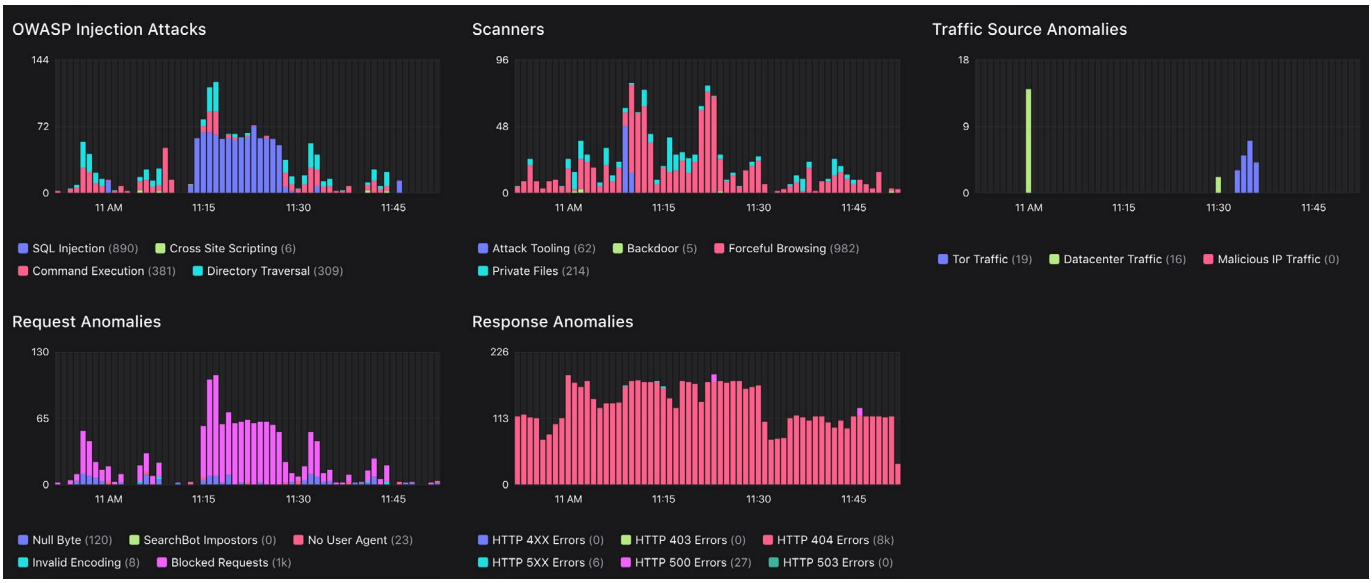
There are two main feedback loops to implement in application security: divergent patterns and known attacks. Divergent patterns, or signals, are seen in traffic that perhaps attempts to access resources that don't exist or spikes in traffic from uncommon sources. Known attacks are common OWASP Top Ten items like XSS or injection attacks. Feedback loops in both areas bring visibility to an otherwise neglected aspect of our systems.

Usage Feedback

Customer usage is an often overlooked, yet very important, feedback loop for security and DevOps. Are you experiencing a higher volume of logins? What about password changes? Have you seen more accounts created in the last hour than normal? These are all subjective questions that are specific to the current business state. More than likely, some of these metrics are already being tracked within the organization, but aren't visible throughout. Enterprise security teams should use these feedback loops to check for anomalous behaviors that are indicators of current or successful attack signals.

When combined with application security feedback, usage metrics become more powerful. Often these will give clues to how successful the attacks are. If there's a spike in XSS attacks, it's a more powerful metric when correlated with the number of password change requests happening in the application. Instrumenting the common flows for users in your system and tying them to application security feedback can bring tremendous value to all sides: development, operations, security, and most importantly, the business.

> Real time attack and event data are required for application security success. Next-gen WAF and RASP technologies, when properly integrated with a DevOps toolchain, can provide the feedback necessary to make informed actionable security decisions.



TRANSFORMATION AREA 2

Unite Security and Engineering Culture

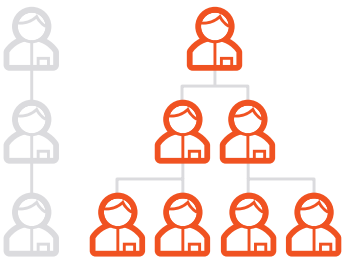
Culture is the foundation of any business function, and that's especially true for DevOps. In fact, many of DevOps' early adopters define it first and foremost as a cultural movement followed by operational and technology requirements. The adherence to a culture-first approach to DevOps was an outcropping of the organizational divide between development and operations. The cultural divide was often apparent just by examining an organizational chart. With staffing ratios of ten developers to one operations staff, coupled with different chain-of-command paths, it was easy to pinpoint the source of the problem. There were also competing priorities between stability (operations) and features (development). This tension created silos based on functional roles in many organizations.

> "You can't directly change culture, but you can change behavior, and behavior becomes culture."
>
> **Lloyd Taylor VP Infrastructure**, Ngmoco

Before DevOps had a name, it was originally referred to as Agile Infrastructure. Agile was successful at transforming development practices and behaviors. It seems obvious now that if the same Agile principles were applied to operations, the cultural divide could be resolved. Due to the close relationship between Agile and operations, behaviors did start to change. New practices arose like Scrums, Kanban boards, standups and planning poker sessions. These collaboration practices were evidence of the behaviors that would eventually influence cultural change.

Business Impact of DevOps Culture

As DevOps amasses a larger following, the industry has begun to quantify the cultural impact that DevOps makes. The 2016 State of DevOps Report released a significant finding on culture: "Employees in high-performing teams were 2.2 times more likely to recommend their organization as a great place to work." This statistic alone demonstrates the potential for culture change using DevOps concepts but even more interesting stats help solidify the fact that DevOps is a business enabling ideal.



Employees in high-performing teams were twice as likely to recommend their organization as a great place to work.
– 2016 State of DevOps Report

Employees working in high performing teams are twice as likely to recruit friends to work with them. Improved recruiting means that employees are not simply happy, but proud to associate their personal brand with the company. This is a direct reflection of the organization's culture and that teams are accomplishing their goals. Having personal pride in work is something that everyone strives for. It's one thing to work somewhere, but it's a completely different thing to ask your friends to join you.

What Culture Means to Security

If culture is the foundation of DevOps, and solving the cultural divide is important, shouldn't security take notice? It's easy for security to identify with the problems between developers and operations—security faces similar issues. An average staffing ratio of one hundred developers to one security engineer illustrates an even larger divide than that which exists between developers and operations (1:10 as mentioned above). Alongside this inequitable distribution of labor, there's the very real challenge of differing priorities: speed and features vs. defense and compliance.

100:1 | A normal staffing ratio of developers to security staff.[14]

As security makes the cultural transition to DevOps, security professionals must:

- **Recognize that if security blocks progress and speed, it will be ignored and marginalized.** Building or fostering a culture of gating functions surrounding security is not a sustainable or forward-thinking model. Security must get out of the way of progress in order to survive.

- **Collaborate across the organizational landscape and deputize security champions.** Enterprise security can't be solved by simply hiring additional resources as there isn't enough security talent available to fill the current needs, let alone future growth. Instead, the most effective security organizations are discovering ways to deputize security champions across their organization.[15]

Pragmatic Technical Changes

It's often said you don't fix cultural problems with technology. Generally, this statement is true, however, there are some technical changes that can influence cultural behavior. For example, the 2016 State of DevOps Report highlights one development pattern that actually led to higher performance and a better culture: the practice of trunk based development. This development pattern consists of having branches or forks in low quantities, with short lifetimes, and is an important aspect of continuous delivery. Lowering branch count was found to contribute to fostering a much higher performing team.

For security engineers there are two practices that impact culture: a Lean Security approach to eliminate waste and the democratization of security data.

LEAN SECURITY AND ELIMINATING WASTE

Currently, the entire software security industry is built on inspections at the end of development through processes like annual penetration testing or compliance assessments. This model runs counter to Lean Software Development practices.[16] Using annual cycles and end-of-cycle inspection is harmful because it creates waste, delays learning, and slows down overall delivery.

> "Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place."
>
> **W. Edwards Deming**[17]

One of the first considerations of a Lean Methodology is identifying waste and eliminating it from your production. At RSAC 2016, Ernest Mueller and James Wickett presented on Lean Security[18] and how to identify waste with security practitioners in mind. Security professionals should focus their process improvement energy on lowering or otherwise improving:

- **Excess Inventory:** Caused by handing off a thousand page PDF of vulnerabilities to an already busy team,excess inventory can be solved by prioritization and limiting the Work In Progress (WIP) queue. Focus on attainable goals that don't overwhelm your staff.

- **Overproduction:** Security controls stemming from fear, uncertainty, and doubt (FUD)—the lingua franca of security—cause a misalignment with actual business need. Instead, choose to align with actual needs and eliminate ideas that can't be solved and only lead to confusion and FUD.

- **Extra Processing:** Relying on compliance testing cycles as opposed to designing processes to eliminate problems from inception is a major issue. Security practitioners must decide to be involved in the earlier stages of software creation rather than post-development testing.

- **Handoffs:** Handing problems to others to solve instead of collaborating and being a part of the solution limits collaboration and long term effectiveness. Solve problems together—don't pass the buck.

- **Waiting:** Lag time waiting for approvals or analysis for security fixes impedes the goals of the business. Create self-service flows by automating security tooling, thus lowering the impact on development and operations.

- **Task Switching:** Rapid "break-fix" work or hot patching should be avoided. Security should adapt to use the current "work intake" processes that the development team prefers. Whenever security can operate within the confines of the current operational model, they should do so.

LEAN SOFTWARE DEVELOPMENT PRACTICES

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

- **Inaccurate Defects:** Both false positives and false negatives are unimportant findings that often get reported, resulting in zero-value rework items and a waste of development and operational resources. Validate findings before reporting them to the team, and make sure they're legitimate to streamline software security improvement.

The process of finding waste and eliminating it in your system will increase productivity and boost culture. Shifting security engineering efforts earlier in development is not just about the removal of waste—it has plenty of cultural benefits as well. Making gains both on your culture and your productivity will give your teams a one-two punch of security improvement.

DEMOCRATIZATION OF SECURITY DATA

The rise of DevOps has spurred other sub-movements, one being ChatOps. ChatOps is the practice of integrating your monitoring, logging, and other operational tasks into the team communication medium. Many organizations achieve ChatOps goals via Internet Relay Chat (IRC) or other chat client systems like Slack or Atlassian HipChat. Almost every piece of tooling you use integrates in some way—from code deploys to monitoring to new customer signups—with the leading technologies in the ChatOps space.

From a security vantage point, many teams have benefited from integrating their security tooling into their development and operational ChatOps efforts. Security and ChatOps integration takes the siloed knowledge of where attacks are happening and distributes it across the organization, opening up major lines of communication. As you might have guessed, security plus ChatOps has changed the way security is perceived at many enterprise organizations by quickly turning the ChatOps concept into a security-centered cultural win.

> At Signal Sciences, we distribute security events to the entire team through methods that encourage collaboration. Many of our customers integrate with Slack and Atlassian Stride, or alerting products like OpsGenie, VictorOps, and PagerDuty.

TRANSFORMATION AREA 3

Delivery Cadence

Continuous Integration and Continuous Delivery (often referred to collectively as CI/CD) are not wholly new concepts but growth and adoption are on the rise. The focus on delivering software rapidly is influenced by the rapid growth of DevOps, and in many ways the delivery cadence of an organization is an indicator of how successful your organization has been at adopting DevOps. This doesn't mean that faster is always better. Success is better measured by reducing Mean Time Between Delivery (MTBD) for your organization. Moving from monthly to weekly to daily delivery is a journey—and it's a journey worth making.

HIGH-PERFORMING IT ORGANIZATIONS:

experience
60x
fewer failures

recover from failure
168x
faster

deploy
30x
more frequently

have
200x
shorter lead times

– 2015 State of DevOps Report

Why is Delivery Cadence Important?

In the 90s and throughout the early 2000s, most of IT followed a waterfall model for delivering software. Software spent the majority of its time in architecture and design, and only towards the end of development did it actually come together to function. The window for design and development could easily have been six to twelve months or longer, with the last month being the integration phase where it was all connected, ran together to be tested as one final unit. In many cases, this would be the first time the software would come together to function as a whole.

The theory behind waterfall development is that if all the requirements were gathered first to specify all the development tasks upfront, then at the end it would produce the correct result—within budget and on time. Effectively, all changes are batched together into a release in the latter stages of the software engineering effort. Since delivering a batch of changes is quite an undertaking, it's untenable to do it that often. This causes releases to happen as slow as twice a year or, worse, once every twelve months in many organizations. Since releases are so infrequent, waterfall also encourages stuffing as many changes as you could fit into each individual release so you don't have to wait for the next release that may be months away.

Many believed that waterfall would result in less rework due to upfront specificity, increased stability, and security since all changes would be made in large batches. Today, the industry has realized this type of thinking is incorrect.

The 2016 State of DevOps Report found that high performing organizations actually deploy 200 times more frequently than their peers resulting in faster recovery times, less rework, and faster cycles from concept to cash. In addition, security gets better from faster delivery, not worse. The report continues, "We found that high performers were spending 50 percent less time remediating security issues than low-performing organizations. In other words, because they were building security into their daily work, as opposed to retrofitting security at the end, they spent significantly less time addressing security issues."

Two key tools that need to be in place to influence enhanced delivery cadence:

- **CI/CD System.** Using a CI system like Jenkins or a service like TravisCI or CircleCI that runs tests and creates artifacts that can move on to the next stage in the delivery pipeline.
- A **deployment system** with minimal gates that handles orchestration.

CI/CD systems are available as a service with two great options as the market leaders: **TravisCI** and **CircleCI**.

Three Common Practices with Security Implications

There exists an excellent book titled *Continuous Delivery* by Jez Humble and David Farley that is the comprehensive and definitive work on the subject. This book defines three specific practices that improve security:

1. SMALLER CHANGES ARE EASIER TO RATIONALIZE

One of the benefits of having a higher frequency of deploys is that you will also have smaller number of changes going out each time. This makes each deploy simpler and easier to rationalize as well as giving security the ability to isolate changes made to the more sensitive portions of the code base.

2. AUTOMATED TESTING

Continuous Delivery pipelines hinge on automated testing. Each commit, no matter how small, goes through the same testing before getting released to a pre-production environment and ultimately to production. This is a good thing—and security can take advantage of this playing field by adding in static and dynamic security tooling (SAST and DAST respectively) to the pipeline.

3. ASSURANCE AND CONFIDENCE IN CHANGES

One of the core tenets of Continuous Delivery is that the artifacts (the outcome of a build) are only built once, and as much as possible, are immutable. Continuous Delivery tracks the artifact to a repository, through completion of testing, to production deployment. This practice increases confidence and assures the security team that there's an audit chain for changes.

TRANSFORMATION AREA 4

Treat Everything as Code

The last area to explore where security fits into DevOps is treating everything as code (also called "Infrastructure as Code"). It might seem odd to you that this was left until the end. This was intentional because it's often the first thing thought of when joining DevOps and Security, but by putting the other items first, the hope is to draw attention to the more neglected areas of DevOps.

Infrastructure as Code is the complete codification of the system from networking and routing to system configuration to all the acceptance and smoke tests. Everything that's needed to create, run, test, change, monitor, secure, and destroy infrastructure, and the system as a whole, is expressed in code. During the early days of DevOps, this was the force du jour of the movement. Operations engineers moved from storing configs and scripts in shared drives and wikis to actually using version control and building complete automation of their systems.

As DevOps grew, so did our understanding of Infrastructure as Code. The broader goals of Infrastructure as Code have security implications:

- **Version controlled artifacts that describe the system and all its components.** This keeps configuration out of wikis and documents and in a versionable, referenceable state.
- **Configuration management of the system in running state.** Configuration and runtime state tracking replaces CMDB.
- **Testing as a first order priority with test-driven development (TDD) and integration testing as common practices.** Tests are written for infrastructure code as well as application code while under development. Writing tests while creating your infrastructure both asserts desired state as well as provides a test suite for CI/CD efforts.
- **Facilitating distributed computing and scaling.** Without treating infrastructure as code, scaling is difficult and distributed computing (cloud) becomes almost untenable. Seeing distributed computing and scaling as desired outcomes guides the development practices.

- **Understanding your software supply chain.** Software is not merely the hundreds or thousands of lines of code that are written by developers. It's composed of much more, from dependencies to the OS to the virtualization framework. Infrastructure as Code encourages software supply chain management by introducing specificity and an auditable log for the actual runtime of the system.

Some practical artifacts of adopting Infrastructure as Code include Dockerfiles, Terraform Plans, or Chef cookbooks. The use of such artifacts will change based on the underlying infrastructure shifts from bare metal to virtual machines, to public cloud services, and now to containers and serverless patterns. No matter what types of infrastructure you're using, whether AWS EC2, Kubernetes, or Azure Functions, we see each of these broader goals of Infrastructure as Code in practice. We will take each of these goals in turn and evaluate where security fits in.

Version Controlled Artifacts

Having version controlled artifacts is one of the first steps to doing Infrastructure as Code. These artifacts bring in the core functions of auditability and change control to the operational process. Version Controlled Artifacts include version controlling all of the configuration management code but also creation scripts and image packaging code. There are often other pieces of the infrastructure that need to be added that can't operate as readable artifacts. These are components like ssl wild card certs, license files, or passwords and often will be version controlled but only in encrypted binary form.

In Gene Kim's book on the topic of change control, Visible Ops Security, Kim demonstrates a direct cause and effect relationship between the ability to detect change in security components and the success of security initiatives. With operations moving into version control, just like in development, the security team now has a foothold and view into the entire system. This visibility encourages change control with alerting of changes to critical components and auditability that was never available previously.

Configuration Management

Configuration management expresses the configuration of the running system in code. Convergence and idempotency are the two core concepts behind configuration management.

- **Convergence** assures that the infrastructure will reach its desired state through the configuration management system.
- **Idempotency** guarantees that a command can be run over and over with the same results. Because configuration management has both of these attributes, there can be better reasoning around the system.

From a security team's perspective, there are two key benefits to configuration management. First, configuration becomes accessible in an easy-to-read, federated format which simplifies auditing and gives security insight into how the systems are built, complete with logs. The second benefit is compliance and adherence to policy. Policy enforcement is a fundamental function of a successful security team. Configuration management allows security teams to reach their goals in an automated fashion.

The majority of configuration management systems have built-in functionality to run in validation mode rather than to attempt convergence. Running configuration management in validation mode allows verification of the system on a daily (or more frequent) basis, ensuring deviations from compliance standards are kept in check. The popular configuration management system, Chef, provides this exact benefit via Inspec, an open-source testing framework for specifying compliance and policy requirements.[19] Inspec provides a huge advantage by creating daily reports of runtime drift out of compliance or even more importantly, identifying new exposure areas.

Testing

There are two main types of testing relevant to infrastructure as code: test-driven development and integration testing. Test-driven development means that the developer writes tests alongside the development of the application or, in the case of Infrastructure as Code, the infrastructure. There are numerous benefits, but one of the key improvements is the creation of a functional test suite that can be used with continuous integration / continuous deployment efforts.

The second type is Integration Testing. This is an outside-in approach of asserting that the infrastructure and system meet the requirements set forth at the time of design. Tools like Serverspec, KitchenCI, or Robot Framework are often used to do this layer of testing. It's tempting to think that integration testing is done at a later stage in the software development lifecycle, but there's a growing trend of shifting this testing "left," or earlier in the development and delivery pipeline.[20]

The O'Reilly book, *Agile Application Security*, states, "The goal should be to come up with a set of automated tests that probe and check security configurations and runtime system behavior for security features that will execute every time the system is built and every time it is deployed." This means that security testing is not treated differently from the other types of testing.

In fact, the industry is continuing to move security testing further left in the pipeline using tooling like Gauntlt.[21] With these security-centric testing frameworks, you gain the ability to specify the security standards all software should meet. For example, "our website should not fail a scan for XSS," or, "when not logged in you should not be able access certain resources." Once the definition of the requirement is set, you can implement automated integration testing and test driven development to ensure success.

Cloud and Distributed Computing

Cloud computing changes our threat landscape. In today's world, we often find ourselves running our systems on third-party providers like Amazon Web Services (AWS), Microsoft Azure, or Google App Engine. Running in these cloud providers changes how we think about security incidents and lateral movement. Cloud computing changes our threat landscape. Attackers are less likely to gain a foothold by pivoting across your systems through network segments, but instead will attack your cloud provider's configuration and seek to open holes in that environment.

One key weakness that most enterprise security departments have is that they're not prepared to deal with the cloud landscape. In fact, industry experts have dubbed it a "black hole" due to the disconnect that they often feel. [22]

To deal with this, cloud providers like AWS provide a complete audit log called CloudTrail which logs all changes to every single configuration in your cloud architecture. Meanwhile, auditing monitors all system commands run on the hosts. Combining these two vectors of logging and auditing provides a clearer picture to changes happening throughout the environment.

One company taking this approach is ThreatStack. They actively look for changes with security implications both at the host and the cloud provider configuration layer.

Software Supply Chain

Software is not merely the hundreds or thousands of lines of code that are written by developers. In reality, software is composed of much more, from individual dependencies, to the operating system, to the virtualization framework. Unfortunately, the software we build inherits vulnerabilities from the entire codebase—*including code we didn't write.*

Infrastructure as Code encourages software supply chain management by introducing specificity and an auditable log for the actual runtime of the system. Software supply chain is a difficult problem to solve due to the nature of code reuse, and knowing what code is shipping through a software bill of materials (Software BOM) is an important task for both engineering and security. Knowing what is in your current runtime down to the specific version is critical, including all code libraries as well as sub libraries that your inclusions use.

A solution like **Sonatype** can be immensely helpful to understand your Software BOM.

Following the Roadmap

The DevOps Roadmap for Security was written to help provide guidance to security practitioners preparing for, or currently experiencing the transition to, DevOps and DevSecOps in their organizations. This is no small task—uniting the DevOps and security tribes radically changes an company's culture. Now that you have the roadmap and understand the four areas to focus on, it's time to follow the roadmap.

The first edition of the *Roadmap* put feedback loops as the last area, but in this edition it was moved to the first area to explore. This was in part a reaction to the security industry's belief that DevSecOps is all about shifting left—moving security testing closer to development. This is a worthy pursuit; however, no matter how much testing is put in place, there needs to be a focus on instrumenting the runtime environment and creating feedback loops. When are you under attack? Are the attackers finding success? These are questions that no amount of "shifting left" could ever answer.

We suggest that you follow all of the *Roadmap*; but if you're just getting started, beginning by creating security feedback loops is usually the best place. This puts security instrumentation in your production applications and creates feedback to developers, operations, and security. Adding this level of instrumentation with Signal Sciences can support faster development cycles, and serves to change the perception of security in an organization from the "inhibitor to innovation" to an accelerator of innovation. Using the DevSecOps practices discussed in the *Roadmap*, there's a real chance for security to add value to the organizations they're protecting.

APPENDIX

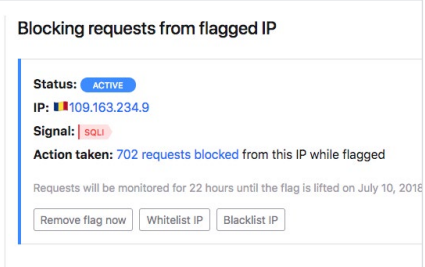
1 Agile Application Security, Laura Bell, Jim Bird, Rich Smith, Michael Brunton-Spall, O'Reilly Media, Inc., September 2017. Emphasis added.
2 Thinking Security, Steven Bellovin
3 theagileadmin.com/what-is-devops/
4 puppet.com/resources/white-paper/2015-state-devops-report
5 DevOps borrows heavily from Lean and moving from Concept to Cash was popularized in *Lean Software Development: From Concept to Cash* by Mary Poppendieck and Tom Poppendieck
6 gartner.com/newsroom/id/2999017
7 CA Research Report: DevOps: The Worst-Kept Secret to Winning in the Application Economy
8 amazon.com/Leading-Transformation-Applying-DevOps-Principles/dp/1942788010
9 puppet.com/resources/white-paper/2016-state-of-devops-report
10 info.signalsciences.com/devsecops-community-survey-2018
11 medium.com/@aneel/theory-in-practice-ooda-mapping-antifragility-df7f03a36a9c
12 theleanstartup.com/principles
13 Sam Newman rocked the industry with his recently released book on Microservices. samnewman.io/books/building_microservices/
14 The OWASP Top Ten is largely unchanged since 2004 even though the list has been refreshed and updated with new data every three years. owasp.org/index.php/Category:OWASP_Top_Ten_Project
15 labs.signalsciences.com/more-silo-smashing-ideas-bringing-infosec-and-devops-together-c338eb3b36ad
16 Lean Software Development practices were created by Mary and Tom Poppendieck in their book Lean Software Development: An Agile Toolkit (2003)
17 deming.org/explore/fourteen-points
18 slideshare.net/mxyzplk/lean-security-rsa-2016
19 chef.io/inspec/
20 en.wikipedia.org/wiki/Shift_left_testing
21 gauntlt.org
22 computerweekly.com/news/450300208/DevOps-a-black-hole-for-security

ABOUT THE AUTHOR

James Wickett (@wickett) is the Head of Research at Signal Sciences, a web protection platform that high performing DevOps teams love. He is the author of the most popular courses on DevOps topics in the Lynda.com and LinkedIn Learning platforms. James lives in Austin, Texas and has helped run DevOps Days Austin for the last six years. In his spare time he is trying to make a perfect BBQ brisket.

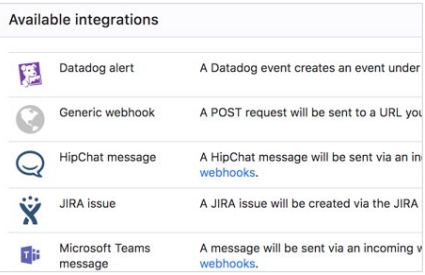
About Signal Sciences

We make web applications more secure. Simple as that. We provide unparalleled web protection that security, operations, and engineering teams actually want to use.



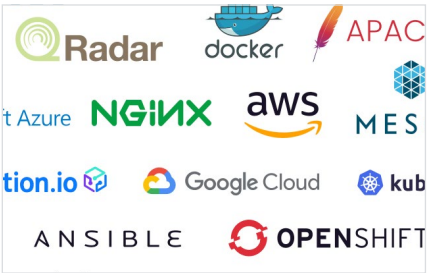
Reliable, automated blocking

- Runs directly in your web servers or application code
- Fail-open architecture keeps your site running fast
- Proprietary SmartParse detection requires no tuning or maintenance



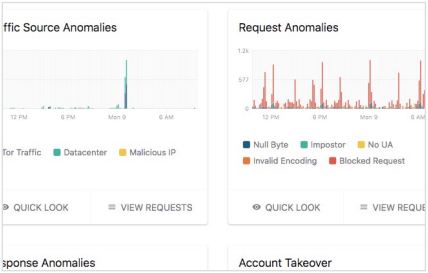
Focused on DevOps

- Easily deployed by operations teams
- Cross-team visibility into metrics, performance and trends
- Integrated into toolchains for quick access and collaboration



Any platform, one UI

- Functions anywhere: in containers, on-prem, or in the cloud
- One unified view across your entire footprint
- Protects and monitors both internal and external services



Coverage against all threats

- Immediate blocking of common OWASP attacks
- Meets PCI 6.6 compliance requirements, but doesn't stop there
- Blocks account takeovers, bad bots, application denial of service, and more

Learn more at signalsciences.com.