Analyst Program

Survey

A SANS Survey: Rethinking the Sec in DevSecOps: Security as Code

Written by **Jim Bird** and **Eric Johnson**
Advisor: **Frank Kim**

June 2021

Executive Summary

As IT workloads move to the cloud, organizations face a fundamental shift in how to develop and deliver systems—and in their security practices. Deploying and running production systems has become abstracted from the underlying hardware and network. Infrastructure is defined through code, and operations work through cloud service APIs.

Security has moved away from selecting and implementing network appliances and writing checklists to Security as Code: reviewing infrastructure and service configuration templates, understanding how to correctly use cloud security services and APIs, and writing automated tests and continuous compliance policies.

Security professionals need to know how to read and write code. They must understand and use modern software development tools to catch security vulnerabilities and to build guardrails and secure defaults into software during code development. They need to understand and use Continuous Integration/Continuous Delivery (CI/CD) build pipelines and programmable configuration management tools to automatically check and enforce security and compliance policies on every change. They need to understand different cloud architectures and platforms, including both their strengths and weaknesses. They also need to do all of this at high velocity, without getting in the way of delivery.

Security as Code represents the future of security. What does this mean to security professionals, to their priorities, to their training, and to the investments that they make in technology and tooling?

This survey, the eighth in an annual series that focuses on application security and DevOps, examines the following with regard to DevSecOps in the cloud:

- What do security teams need to understand about software development to meet the demand of high-velocity delivery?
- What skills enable security teams to architect secure cloud services and ensure that they catch and fix vulnerabilities as early as possible?
- What impact do the different cloud architectures and platforms have on this effort, including risks, strengths, and weaknesses?

SANS surveyed 281 organizations across the world. Figure 1 on the next page provides a snapshot of the demographics of the survey respondents.

DevSecOps: Security as Code

Deploying and operating services in a cloud-native environment requires that we understand and write code. We abstract and present operations functions through services and API calls. System and network implementation and configuration occurs via code. Cloud providers, by abstracting the work involved in deploying and running services through a set of service API calls, have made operations and security into a coding problem.

Software engineering practices for writing and building good code extend from software development to system and network engineering and *even* to security. Thus, security becomes security engineering: writing security and compliance policies in code; reviewing, scanning, and testing application code and service configurations; and understanding how application development and system engineering teams work and helping them to find and implement tools to inject security testing directly into development.

Security as Code requires new skills and new ways of thinking and working: more collaborative and transparent, faster, and more iterative. It requires leaning on automation to solve common problems and to reduce costs and risks.

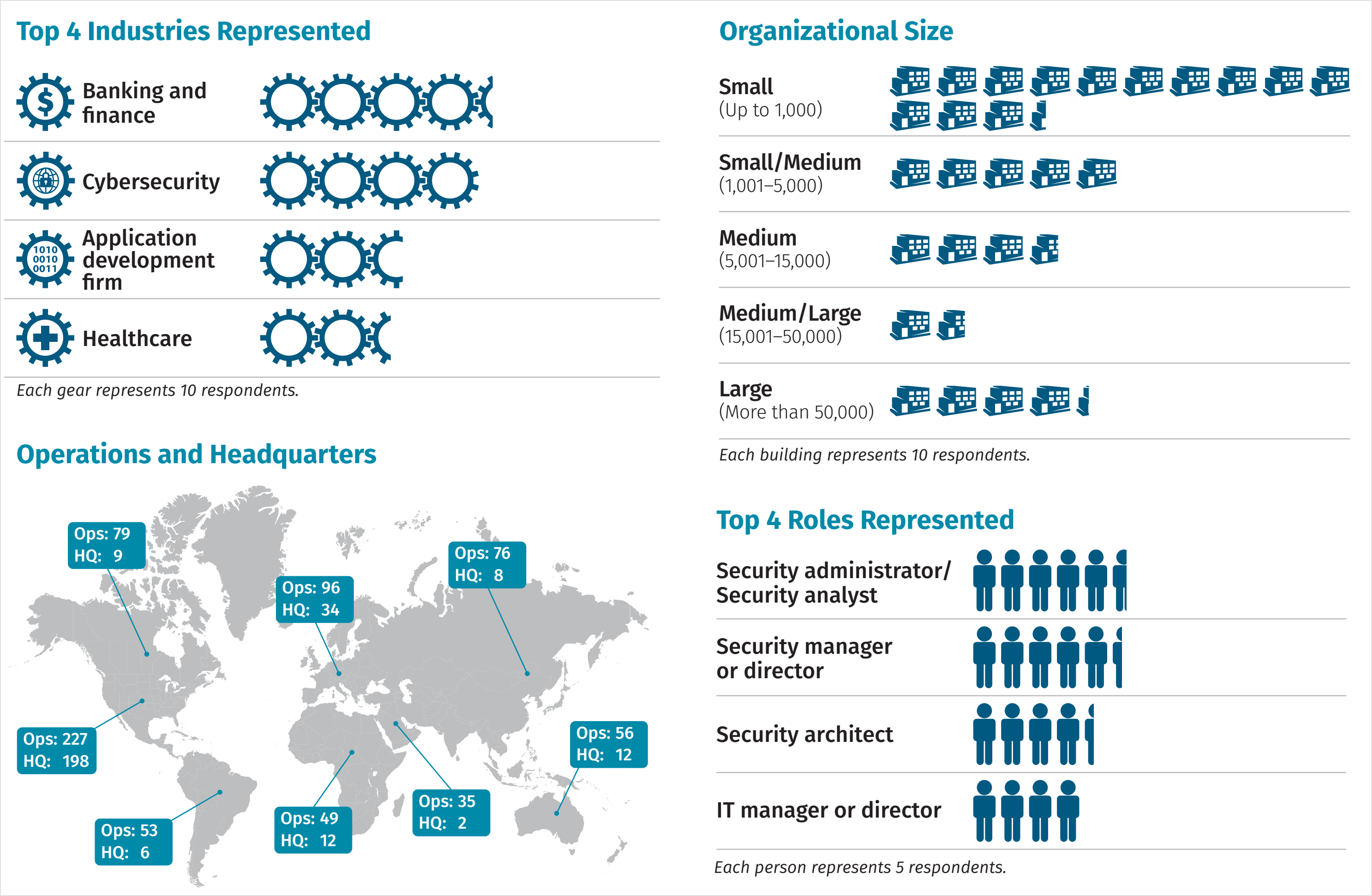

Figure 1. Survey Demographics

How prepared are organizations for these changes?

Key Findings

Cloud platforming:

- More than half (57%) of organizations use three or more cloud platforms. Each cloud platform is unique: The configuration models differ, as do the APIs and services. Therefore, operational and security risks differ, making them difficult to understand and manage. Increasingly, cloud-agnostic tools help to reduce costs and risks.
- Only a third (34%) of organizations have automated cloud configuration through code and platform APIs. Manual configuration is slow, less traceable, and more error prone. Organizations need to increase automation of configuration through programmatic infrastructure as code tooling and build pipelines to keep up with rapid change—and to help ensure continuous compliance.

Velocity of delivery and security testing:

- More organizations are taking advantage of the cloud, DevOps, and lightweight Agile practices and tools to deliver features and changes to production faster and more cost-effectively. The velocity of IT delivery has increased by 14% over the past five years, but the speed of security assessments has failed to keep up. Security testing continues to lag.

- Only 29% of organizations have automated most (75% or more) of their security testing. Fewer than half of organizations (44%) have included security tests and reviews as part of coding workflows.
- Foundational software development practices such as CI/CD and test automation are key to delivery velocity *and* continuous security testing. If development teams do not automate their build/test work, they will find it more difficult to implement automated security testing. Whereas 66% of organizations currently automate builds, only half of organizations (52%) follow CI and take advantage of automated testing.

Operations:

- As delivery teams continue to get faster, so do attackers. Only half of organizations (51%) patch or otherwise resolve critical security vulnerabilities and other critical security risks within a week of identifying these risks. Organizations need to leverage DevOps and Agile practices, and automated build chains and automated testing, to get patches out faster with confidence.
- Organizations, especially organizations that cannot keep up with security testing, need to consider the value of runtime shielding for cloud platforms, to provide continuous protection against configuration and deployment mistakes, emerging vulnerabilities, and evolving threats. Only a third of organizations (33%) rely on operational runtime protection solutions in the cloud. Using Security as Code principles, organizations can integrate protection services such as cloud web application firewalls (WAFs) with serverless functions to parse logs, identify scanners, and automatically block bad IPs.

Barriers and enablers:

- Successfully implementing DevSecOps is not a technical problem; it is an organizational problem. Lack of resources, lack of management and developer buy-in, bureaucracy, poor communication across silos, and poor prioritization hold organizations back, not technology.
- Training in secure coding practices is a key enabler—not just training for developers, but also training for operations, and security, and even compliance personnel. Understanding how to write secure code requires a fundamental change in mindset for everyone involved in IT in the cloud.

Understanding the Cloud Landscape

Mapping out the cloud landscape—the extent of cloud services adoption, the cloud platforms' runtime architectures used, and how applications are built and deployed to the cloud—helps security teams understand the potential risks that organizations need to manage.

Cloud Platform Analysis: The Big 3

A majority of survey respondents (63%) indicated that they spend at least half of their time on public cloud security and operational responsibilities. See Figure 2.

As DevOps teams move their workloads into the cloud, security teams need to learn how to apply operations, monitoring, and runtime security controls across public cloud providers. As with previous surveys, the Big 3 cloud providers—Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP)—continue to dominate the public cloud space. AWS is in use at 86% of respondents' organizations, with 27% of those organizations hosting 75% or more of their applications in the AWS cloud. See Figure 3.

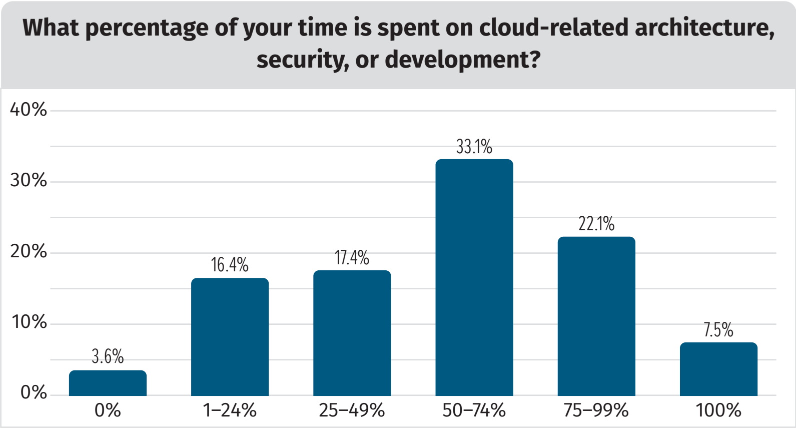

Figure 2. Time Spent in the Cloud

Microsoft Azure continues to close the gap, with 81% of respondents using Azure. However, only 18% of those respondents depend heavily (75% or more) on Azure for the running their workloads. GCP remains in third, with only 61% adoption. Perhaps the most telling gap in the numbers is that only 7% of the GCP users depend on it for 75% or more of their workloads. Almost a quarter of respondents (22%) run at least some part of their workloads on cloud platforms outside of the Big 3.

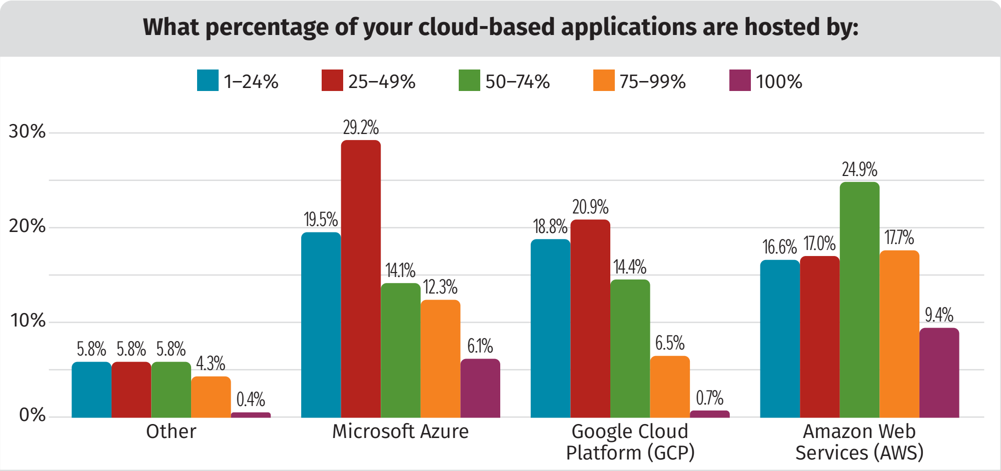

Figure 3. Cloud Hosting Providers

Many organizations, especially large enterprises, work with multiple cloud platform providers. SANS found that most organizations (97%) use at least one public cloud provider, with more half than (57%) running workloads on three or more public cloud providers.

TAKEAWAY

As the technologies and options multiply in heterogenous cloud environments, so do the risks. Security teams must learn different security and data privacy models, identity management and access control schemes, data storage and encryption capabilities, activity auditing functions, configuration languages and defaults, compliance and security controls and APIs, and the architectural strengths and weaknesses for each provider's service platforms.[1]

Having multiple options increases the value of tools and security solutions that work across different cloud service providers (CSPs). For example, Terraform, an open source platform for managing cloud services, enables teams to configure and provision services across multiple cloud platforms using the same toolset and one high-level language syntax (HCL), which reduces development and operational costs and security costs and risks. However, abstraction comes with a downside. You may run into problems configuring and provisioning services where it is unclear if there are mistakes in your Terraform code or bugs or limitations in the Terraform providers (API wrapper) you use.

[1] For more on multicloud security considerations, see www.sans.org/reading-room/whitepapers/cloud/top-5-considerations-multicloud-security-39505

Leveraging Cloud Services to Reduce Cost and Risk

With public cloud adoption rising, organizations are transitioning workloads from on-premises to a mix of cloud-hosted virtual machines (VMs), cloud-hosted container services, and cloud-hosted serverless platforms. As Figure 4 shows, organizations run roughly the same amount of work on cloud-hosted VMs and container services as on-premises. They also use serverless platforms for activities like back-end batch processing and task scheduling.

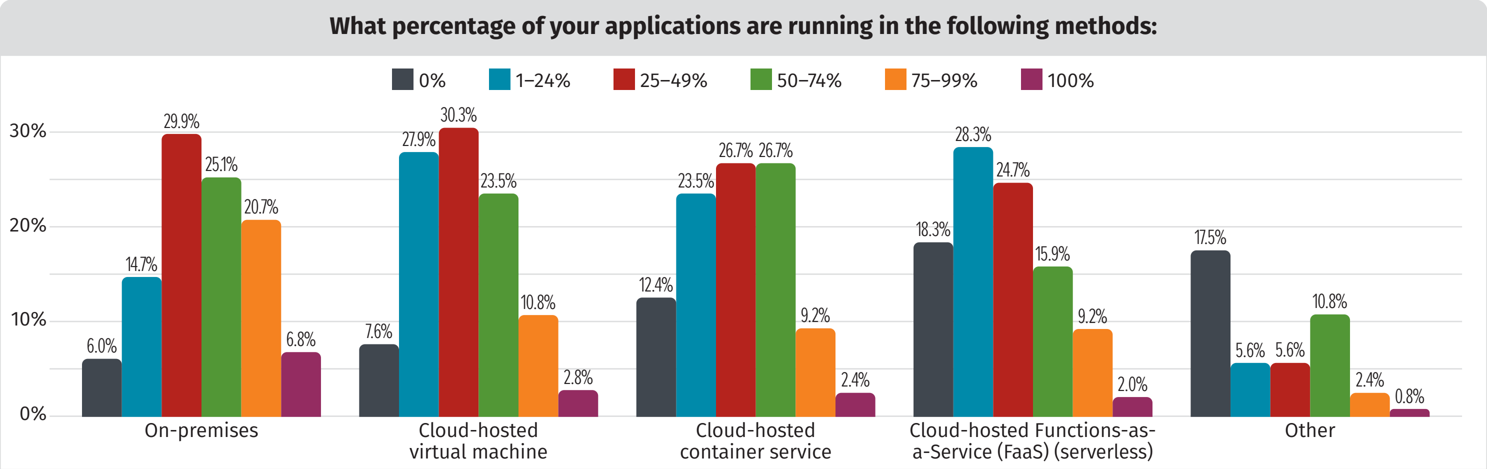

Figure 4. Runtime Platforms

As organizations move from on-premises infrastructure to cloud-based services, operational complexity and scale, development cost, compliance responsibilities, and security risk shift to the cloud platform:

- **On-premises—**The organization is responsible for managing everything.
- **Cloud VMs—**The organization is responsible for managing cloud operations, security, and configuration management. This includes carefully checking the VM platform: Each cloud provider auto-provisions default network and firewall rules for new VMs, which may unintentionally expose private resources.
- **Cloud container services—**The CSP is responsible for patching the VM and provisioning and hardening the container runtime. DevOps teams (and security) can focus on the application and its build/runtime dependencies, packaged and shipped in container images.
- **Cloud serverless—**The CSP is responsible for hardening container images and patching the underlying application runtime shift to the cloud provider. DevOps teams need to worry only about writing and testing application code, the permissions associated with the function's service account, and major upgrades to the application runtime environment. However, even careful coding and testing is not enough to protect your code against common attacks, and a lack of visibility into the serverless runtime makes it difficult to understand what attacks you are facing. This is where cloud workload protection platforms (CWPPs) and other runtime protection solutions come in.

Securing Cloud Container Services

Containers enable developers to package an application and its runtime dependencies in a simple, portable way, and they provide a measure of isolation at runtime. These capabilities make containers a core technology in DevOps. But as container use has exploded, teams need orchestration tools like Kubernetes to deploy, manage, and operate containers at scale.

Installing, configuring, hardening, and managing an on-premises container cluster is a heavy lift and introduces a new set of operational costs and security risks. It is easier and less expensive to shift most of these responsibilities and costs to a proven CSP, using a containers-as-a-service (CaaS) platform. CaaS platforms, such as Google Kubernetes Engine (GKE), Amazon Elastic Container Service (ECS), Azure Kubernetes Service (AKS), and Red Hat OpenShift, abstract the details of deploying and running containers, letting developers focus on what goes into the container rather than how the container itself works. See Figure 5.

The Cloud in Code

The concept of treating infrastructure as code was introduced to help manage runtime infrastructure in on-premises data centers. Tools like Ansible, Chef, and Puppet allow operations engineers and developers to configure and provision runtime technology stacks (i.e., operating systems, VMs, and network and database configurations) in high-level code.

The cloud takes this further. DevOps teams can provision their own IT infrastructure without needing to understand the details of the underlying technology (i.e., the servers, storage, and network) or having to operate it, reducing cost and time to roll out application services. High-level, declarative configuration languages, such as AWS CloudFormation, Azure Resource Manager (ARM), and Google Deploy Manager (GDM), cross-platform tools such as Terraform, and toolsets for containers enable DevOps teams to model, configure, and deploy cloud technology stacks in the same way that developers build and deploy applications, through automated pipelines. You can make iterative and incremental changes to runtime configurations safely and roll out the changes across your infrastructure at velocity and scale. All changes are versioned and audited and repeatable.

As with application code, DevSecOps teams can leverage static analysis tools to find problems in cloud configurations at build time, before changes are rolled out. Modern scanning tools look beyond syntax checking and basic semantic analysis to check for common security mistakes and vulnerabilities, and to validate configurations against compliance policies, up and down the stack (e.g., cloud services, containers, and Kubernetes configuration), and in some cases across cloud platforms. Open source examples include the following:

- **terrascan: https://github.com/accurics/terrascan**
- **checkov: https://github.com/bridgecrewio/checkov**
- **tfsec: https://github.com/tfsec/tfsec**
- **cfn_nag: https://github.com/stelligent/cfn_nag**
- **kics: https://github.com/Checkmarx/kics**

The benefits of configuring the cloud in code are obvious. However, a high learning curve applies to understanding cloud architecture and cloud configuration languages and coding practices and tools at the same time. DevOps teams and security teams need time to learn the concepts and run technical proof-of-concept exercises to evaluate the tools and identify risks.

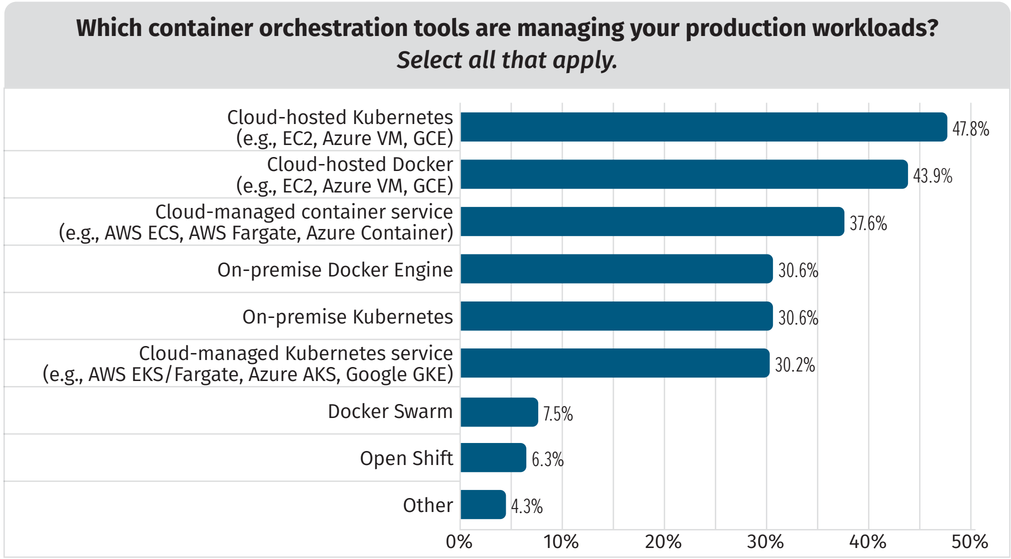

Figure 5. Container Orchestration Platforms in Use

While the most popular platforms for managing containers are now based in the cloud, a still significant number of on-premises hosted container platforms exist, likely due to investments made before cloud-hosted CaaS became available. Simple Docker Engine implementations are common in development and testing—and for security—because security tools are often packaged in Docker containers.

TAKEAWAY

Managed container service platforms still require careful review by DevSecOps engineers:

- **DevSecOps engineers need to evaluate container platforms for misconfigurations and insecure defaults. For example, CaaS resources created through the cloud web console might run workloads in default cloud-managed virtual private cloud (VPC) networks without traditional network security controls such as network logging and egress traffic firewall rules.**
- **Container images stored in cloud-managed registries are not automatically scanned for vulnerabilities. Cloud security teams need to understand each cloud provider's image scanning capabilities and know how to enable this.**
- **Container workloads can be assigned an identity (or service account) with excessive permissions that can allow a compromised container to move laterally through the cloud account and perform data exfiltration.**
- **Runtime detection and incident response tools do not automatically monitor compromised container workloads. If a workload is in fact compromised, having the ability to immediately alert (if a malicious actor actually performed data exfiltration) and to capture a detailed activity record are critical.**

Managing these risks requires security engineers to carefully review the container service configuration, create secure by-default templates (e.g., Terraform modules) for development teams, and have runtime threat detection and response capabilities in place.

Programming Environments and Risks

As development teams move to the cloud, they adopt new programming languages and tools. With these new capabilities come new risks, as shown in Figure 6.

The security risk of any programming language generally tracks its popularity of use:[2]

- **Scripting languages—**Languages such as JavaScript and Python are easy to learn and use, making them increasingly popular in cloud environments, especially with nonprofessional programmers, who are more likely to make dangerous programming mistakes. These languages also make extensive use of open source software packages (e.g., npm, pip), which introduce another class of vulnerabilities that we must manage.

Which languages and platforms in your application portfolio have been the greatest source of risk or exposure to your organization? ***Select your top three.***

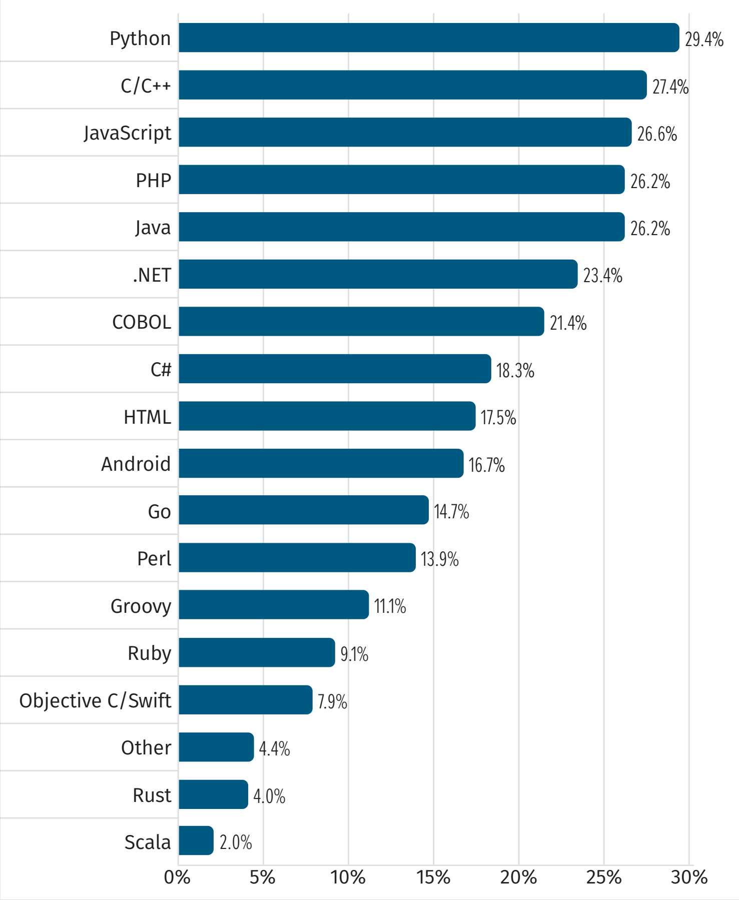

Figure 6. Risk in Programming Languages and Platforms

[2] A useful indication of popularity of use is GitHub's "State of the Octoverse," which ranks development languages used in open source projects: https://octoverse.github.com

- **Memory-unsafe languages—**Languages such as C (especially) and C++ will continue, by definition, to present a major source of security risk. Static analysis checking and manual code reviews are essential to catch coding vulnerabilities and enforce practices such as the Computer Emergency Readiness Team (CERT) secure coding standards.[3]
- **Java and PHP—**These have been mainstays of web development, and although the use of these languages has declined over time, many organizations still have significant legacy investments in this code, as well as code written in legacy languages like COBOL.

Security practices and tooling must be adapted to the needs of development teams as well as the development environments, languages, and frameworks that these teams use.

TAKEAWAY

Newer coding languages and frameworks, and special-purpose languages (including configuration template languages), come with new risks: a lack of secure coding guidelines, limited integrated development environment (IDE) support, and limited tooling for static analysis and software component analysis (SCA). In these cases, developers need to fall back on general defensive coding practices and rely on code reviews, including third-party expert reviews, and other kinds of testing to find security problems. Security teams should identify the risks of working with new or niche languages early, in upfront risk assessments, as they evaluate development tools and architecture candidates.

Security at Velocity

High-velocity, incremental delivery enables development teams to innovate and run experiments at low cost, collect feedback, respond, and change. Founded on these priorities, modern tooling and lightweight Agile practices minimize friction and cost and encourage teams to learn and change as they go. However, the relentless focus on velocity of delivery can force DevOps teams to take shortcuts, putting speed ahead of everything else, taking *lightweight* too far. Teams build up all kinds of technical debt: code that is not clean and safe to change, lack of test coverage, and outdated dependencies. The faster that teams move, the more debt they build up.[4]

Continuous change means that the targets and risks for security and compliance constantly change. Security must become agile and iterative:

- Ruthlessly automate security testing to keep pace with delivery. However, the emphasis on speed and fidelity/accuracy means that you will not catch some problems. You need to recognize these risks, and add deep scanning, fuzzing, and penetration testing or bug bounties to find problems that fast, incremental scanning and testing can't find.

[3] https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=494934

[4] "2017 SANS State of Application Security: Balancing Speed and Risk," October 2017, www.sans.org/reading-room/whitepapers/analyst/2017-state-application-security-balancing-speed-risk-38100

- Continuously enforce compliance on every change. You cannot rely on manual checklists and audits if you change code and infrastructure every day; you will find it too difficult to go back and review so many changes by hand. Encode compliance rules and test them automatically.
- Because code and third-party dependencies and cloud configurations constantly change, you need to track and understand all of this code and how it is built, and think of code and the code build infrastructure as part of the organization's attack surface.
- Working in an iterative, incremental way means that everyone (including the security teams) "learn on the go." Look out for incorrect assumptions, and for holes in knowledge and understanding on the part of the delivery teams and on the part of the security team (you don't know what you don't know). Be prepared to respond quickly to new knowledge and to new risks.

To understand how organizations manage these risks and challenges, we asked a series of questions:

- How often do organizations deliver changes to production?
- How do they deliver these changes? What technical practices and toolchains do they use?
- How often do they test or assess security?
- How much of this testing is automated?
- What security tools and practices do they rely on to manage security risks?

We describe the survey results and our findings related to these questions in the following sections.

Delivery Velocity Is Accelerating

The velocity at which organizations deliver IT changes continues to accelerate. In the early days of Agile development, Scrum teams and XP teams delivered working software every month, which most considered radical at the time. Today, 75% of organizations deliver changes more than once per month, an increase of 14% over the past 5 years. See Figure 7.

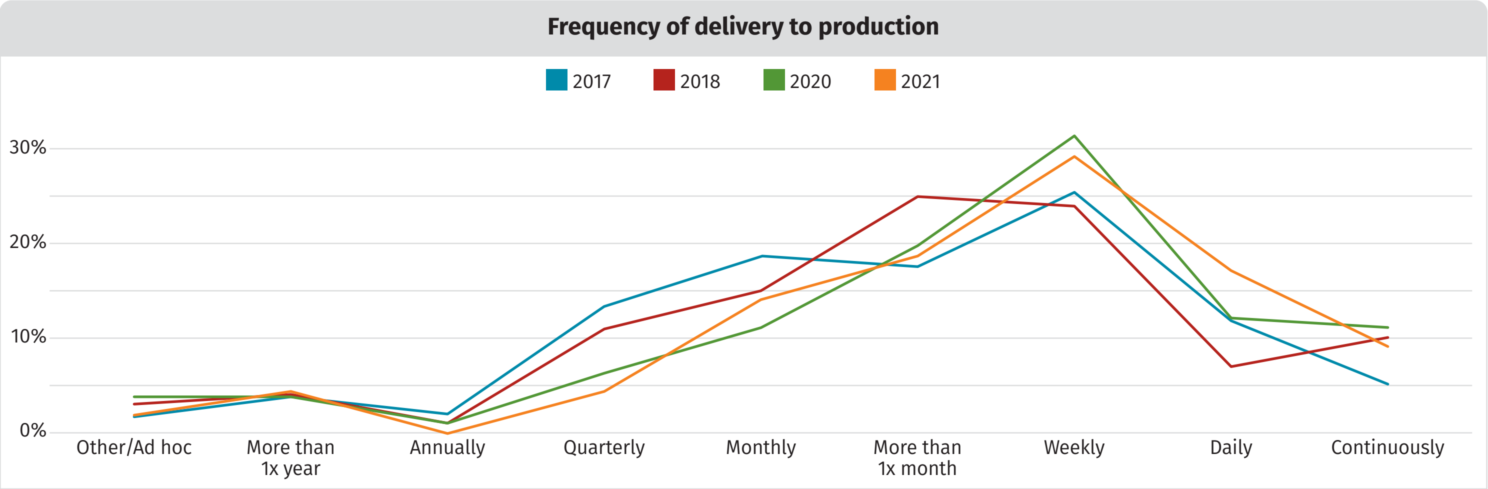

Figure 7. Velocity of Delivery to Production 2017–2021

DevOps Foundational Practices

To deliver changes at high velocity, DevOps teams follow a core set of common technical practices and automation technologies, including those shown in Figure 8. More than half of this year's respondents indicated that their organizations follow iterative, incremental design and development using core Agile and DevOps technical practices, such as automated builds (66%) and CI (52%). See Figure 8.

However, close to half of respondents' organizations do not use automated testing. This implies that they still rely on manual testing (which does not scale and cannot keep up with rapid, iterative development and delivery) or, worse, that they do not test at all. And only 34% are provisioning and configuring infrastructure using modern programmatic tools, such as Chef, Terraform, or AWS CloudFormation, instead of point-and-click admin console interfaces.

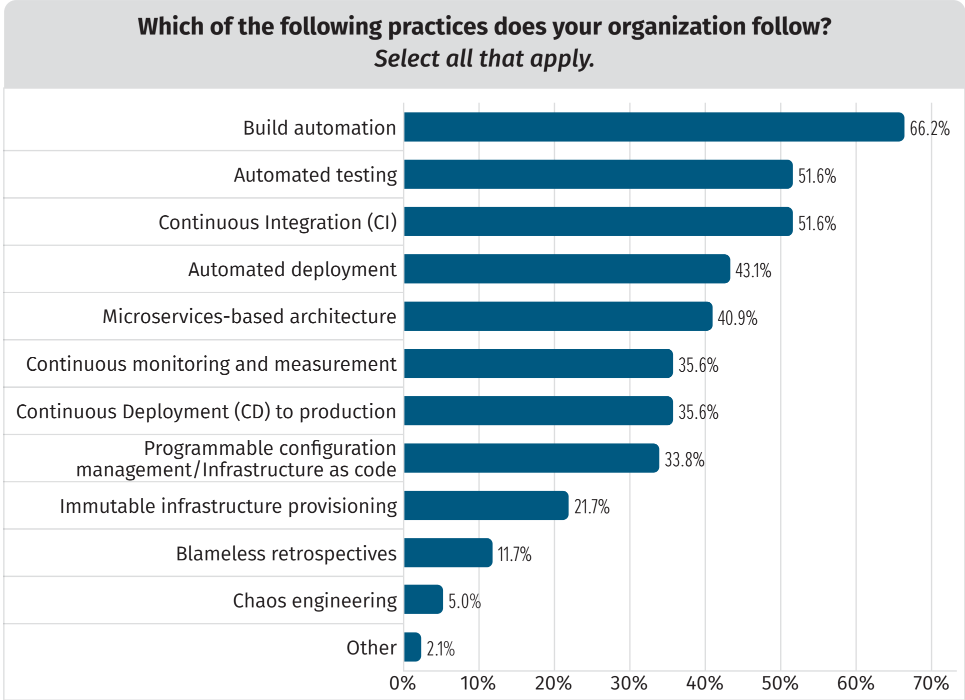

Figure 8. Technical Practices and Automation Technologies in Use

A lot of room exists to improve automation, by moving manual work into code. Consider the following foundational practices and what they mean from a security perspective:

- **Incremental design and development—**Modern Agile development imposes constraints on how security is practiced: must be lightweight, performed in-phase, iterative, and continuous.
- **Microservices-based architecture—**A common architectural pattern for cloud applications, where small teams work independently, designing and delivering small services with each offering discrete responsibilities through APIs. This trades off development simplicity for operational complexity and greatly increases the system's attack surface. Common security risks include holes in API authentication and access control. Protecting these applications requires close collaboration between development and runtime security approaches, including next-generation web application firewalls (NGWAFs), Runtime Application Self-Protection (RASP), CWPP, and API fuzzing.
- **Blameless postmortems—**Retrospectives extended from development to operations to learn from outages and other operational problems and near misses. Teams work together to walk through incidents and incident response and to conduct root cause analysis to identify improvements. These practices can (and should) be extended to security breaches, compliance violations, and near misses.

- **Automated testing—**Developers write their own automated tests that execute on each check-in, to provide confidence in builds and iterative changes, ensuring that code is not broken by accident (regressions). Automated testing is a key prerequisite for continuous security testing. If developers have not already widely adopted automated testing, introducing automated security testing into their workflows will prove problematic.
- **Continuous monitoring—**Infrastructure and culture to detect and respond to exceptions can be used to enlist DevOps teams in monitoring for evidence of attacks on application services and infrastructure and on other security events.
- **Programmatic configuration/infrastructure as code—**Configuring cloud infrastructure in code creates a control structure for security and compliance. We can check configuration changes into version control, and automatically scan and test using CI/CD build pipelines to catch configuration errors and enforce hardening policies. It also provides transparency into runtime architecture and configuration, making it easier to understand and assess risks.
- **Immutable infrastructure—**Changes are made by tearing down and rebuilding runtime instances (for example, using containers). Prevents configuration drift, prevents tampering with runtimes, and ensures that infrastructure configuration is known and traceable.
- **Automated deployment—**Make deployment steps repeatable, testable, automated, auditable, and safe. Enforces change control and reduces the cost and risk of making changes, including deploying patches.
- **Blue/green deployment—**An incremental deployment technique to reduce the cost and risk of rolling out changes.
- **Chaos testing—**Operational failure injection in running systems, either in test or in production. Netflix made this concept famous through their "Chaos Monkey"[5] service. Effective at uncovering problems in technical architecture and for incident detection and response. Chaos testing ensures that security controls remain resilient to DOS attacks and other types of attacks.
- **Build automation/CI/CD—**Moving from manual builds to automated build chains, and then to Continuous Integration and Continuous Delivery or Continuous Deployment, is a prerequisite for iterative and incremental development. Standardizing and streamlining these steps, making them repeatable and transparent, reduces costs and risks for the delivery teams (and for security).

Security and compliance can build on and leverage these enabling practices and tools—if we have the practices and tools in place and working effectively.

[5] https://netflix.github.io/chaosmonkey

CI/CD

Iterative, Agile development requires a repeatable, automated build process. Continuous Integration (CI) is one of the foundational practices in modern software development. With CI, each time code changes are checked in, the code is automatically scanned, built, and tested, providing developers with immediate feedback on its fitness. DevOps teams use Continuous Delivery (CD) to automate packaging, configuration, and deployment, so that each change can be automatically pushed directly to production.

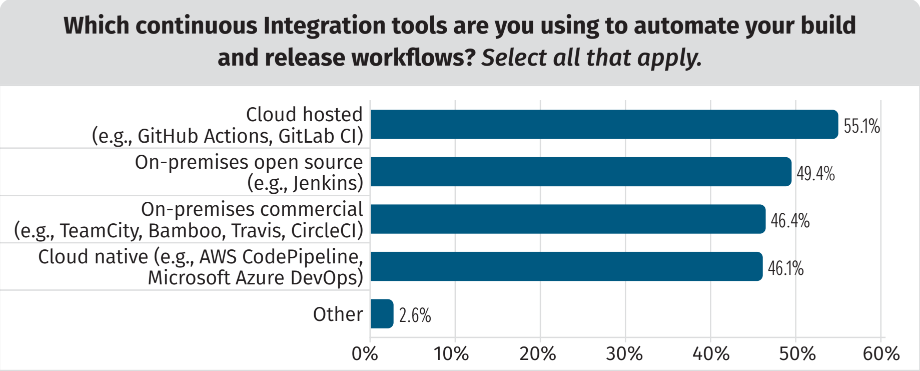
Which continuous Integration tools are you using to automate your build and release workflows? ***Select all that apply.***

Figure 9. Continuous Integration Tools in Use

The CI/CD pipeline provides a control plane for both software engineering teams and security and compliance. As shown in the survey results in Figure 9, a roughly 50/50 split exists between on-premises CI/CD and cloud-hosted CI/CD services. Cloud-hosted platforms enable DevOps teams to offload the responsibilities and costs for provisioning, configuring, hardening, monitoring, and managing these tool sets onto the CSP, allowing teams to focus on the work of coding instead of on the undifferentiated heavy lifting involved in setting up and managing the build infrastructure.

GitHub and GitLab offer managed source code control and CI/CD as a service. These services offload the work of managing code repositories and check-in workflows and of creating, integrating, and running automated build chains. Their enterprise-tier services include built-in code scanning for most common programming languages, secrets scanning, dependency analysis, and other security testing capabilities.

Development teams, and security and compliance teams, can take advantage of these capabilities out of the box, lowering the bar of entry for security testing, and incrementally add additional testing solutions for higher coverage using standard APIs.

Protecting the CI/CD Pipeline

Automated build and deployment pipelines provide a direct path from development to production, making them a tasty target for attackers. If attackers can compromise the repositories or build chain, they can inject malware into production systems without penetrating them. The CI/CD environment all need to be protected: the source and artifact repositories, the CI/CD toolchain configuration and runtime, the keys and other secrets used, and the infrastructure for running these services.

The recent attack on SolarWinds's customers through that company's build environment, where attackers compromised SolarWinds's automated build process to insert a backdoor that was then distributed to customers, has proven the seriousness of the risks. According to a statement from SolarWinds's new CEO, the company is taking the following steps to improve the security of its development and build environments:[6]

- **Increased hardening of the development and build environment and build pipelines**
- **Auditing and surveillance of the development and build processes**
- **Scanning and analysis of source code and build artifacts to ensure that they match**
- **Enforcing multifactor authentication (MFA) and strict access controls across the development and build environment**

These steps offer a roadmap to all organizations on how to protect their own CI/CD environments.

[6] https://orangematter.solarwinds.com/2021/01/07/our-plan-for-a-safer-solarwinds-and-customer-community

Let's now look at speed of delivery from the security perspective.

Is Security Keeping Up with the Velocity of Change?

The faster that engineering and development teams deliver changes, the faster security teams need to identify and assess risks. By comparing the velocity of delivery to the velocity of security testing, as shown in Figure 10, we see that most organizations cannot keep up with the pace of delivery.

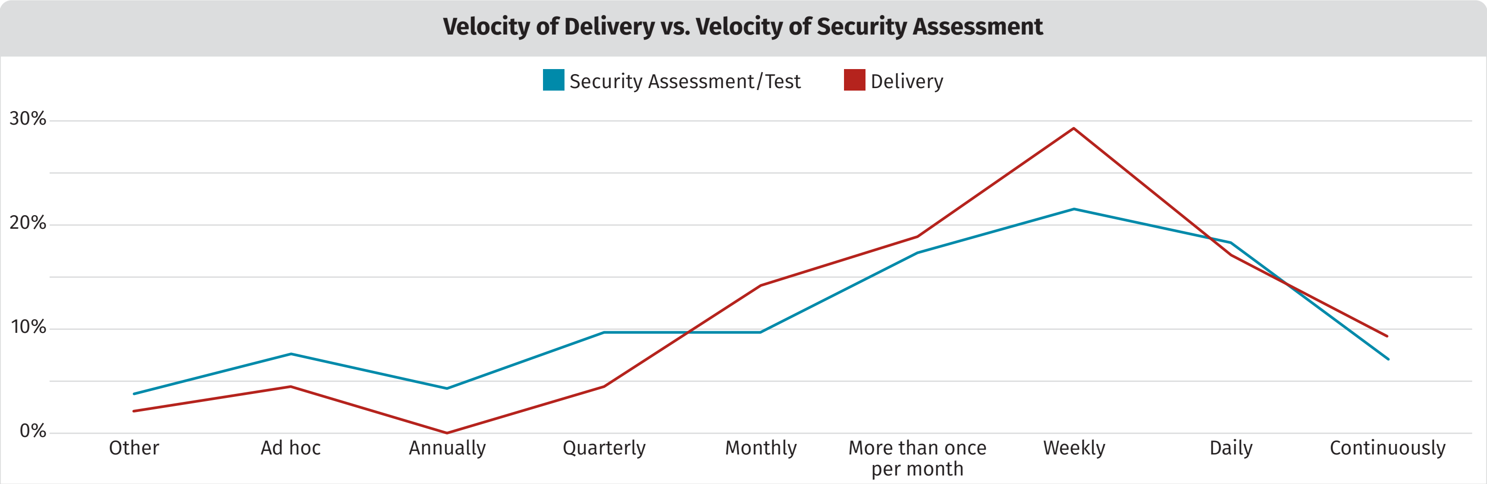

Figure 10. Velocity of Delivery vs. Velocity of Security Assessment

Without testing, we leave vulnerabilities undetected. The earlier in the delivery life cycle that we identify vulnerabilities, the easier and cheaper we can fix them (along with a higher likelihood that we will do so)—the whole point of *shifting security left*.

When Is Security Testing Performed in DevOps?

As shown in Figure 11, security testing occurs multiple times during the development cycle, from upfront requirements to design, through coding and developer/QA testing workflows, and into the predeployment and deployment stages.

However, fewer than half of organizations (49%) have shifted security reviews into requirements, design, and coding workflows, and 29% still rely on release gate reviews, which cannot keep up with high-velocity CD.

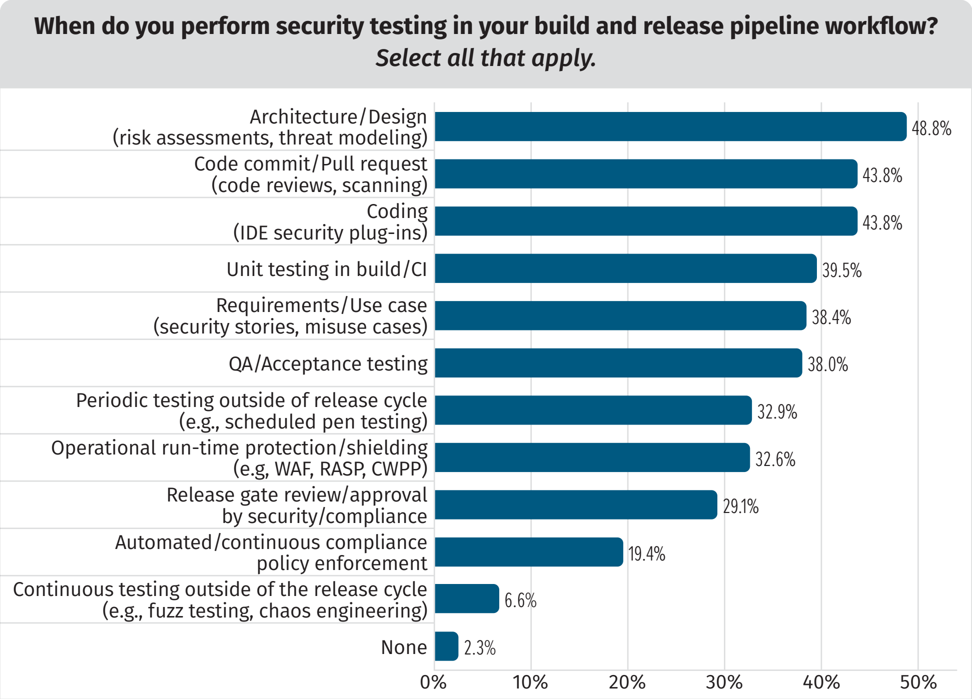

Figure 11. When Security Testing Is Done in Development/Deployment

How Much of Security Testing Is Automated?

Obviously, because manual reviews and pen testing cannot keep up with continuous changes in high-velocity developments, organizations require some level of automated testing. Test automation enables continuous testing, increasing confidence on each change. Unfortunately, only 29% of respondents indicated that they have automated the majority (75% or more) of their security testing, as shown in Figure 12.

A contributing factor to this is that only 52% of organizations automate testing of any kind. Before DevOps teams will accept automated security testing, organizations need testing discipline, automated test infrastructure, and pipeline workflows in place.

Test coverage determines your level of confidence in a team's ability to quicky and safely make changes, including rolling out patches. Tracking test coverage also highlights risks associated with areas in code that cannot be trusted as correct.

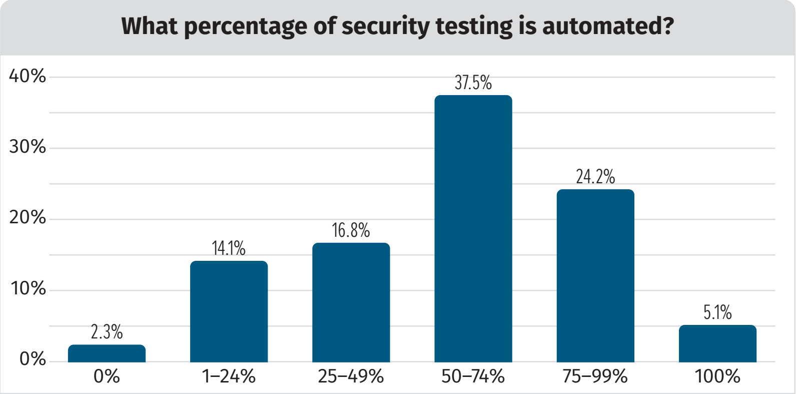

Figure 12. Automation of Security Testing

Hijacking Developer Testing

As development teams adopt test automation, you can hijack the work that developers are already doing, the tests that they are already writing, for security purposes. For example, unit tests are automated tests written by developers to exercise code logic, which generally make up most of the tests in an automated test suite. These tests can play an important role in security testing. Ensure that you have good coverage for security-critical code, including authentication, encryption, logging, and especially access control/permissions (which is driven by business logic and organizational structure). Also, encourage developers to get off the happy path[7] and write negative tests (tests that should always fail) to exercise error/exception handling code. Doing so makes the code more robust and better protected from attackers.

TAKEAWAY

Lack of automation is a key reason that security does not keep up with the pace of delivery. Automation needs commitment from development/engineering teams and their managers. Such commitment proves easier with modern CI/CD platforms that include some security testing capabilities and that make it easy to add more testing.

Some of the different test techniques and tools also build on each other:

- **Dynamic Application Security Testing (DAST)—**We can run testing as a separate testing stage (inside or outside of the build pipeline) or by proxying automated functional tests (e.g., Selenium WebDriver test suites) through an attack scanner to include passive and dynamic scanning of web interfaces. The coverage of these tests relies on the functional tests (and the amount of time we allow the tests to run).
- **Interactive Application Security Testing (IAST)—**Results also depend on the coverage of the automated test suite to catch security vulnerabilities, by analyzing code paths while we exercise the code.

[7] https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=29002

The type and success of testing techniques and tools depends on who holds responsibility for security testing:

- **Development teams—**Static Application Security Testing (SAST) and SCA code scanners (and container scanners) prove a natural fit provided that the tools are fast and accurate. Automated unit testing is also a developer responsibility, although security teams can help by identifying areas of code that need coverage and by writing negative tests.
- **Security teams—**DAST and fuzz testing prove popular with security professionals, who tend to think more like pen testers and less like developers.

This leaves testing technology like IAST in an awkward, in-between situation—one reason why IAST represents a niche solution. Implementing IAST requires a higher level of maturity on the part of DevOps teams. It also calls for cooperation between development, operations, and security to set up the runtime, manage the tests, and own the results.

Who Is Responsible for Security Testing?

Who owns writing the tests, running them, and keeping them up to date within most organizations? Have organizations shifted responsibility for testing left onto DevOps teams, or do they still silo testing in the security team or outsource it to security consultants or bug bounty testers? As shown in Figure 13, in most organizations the security team—whether an internal security team (58%) or external security consultants (42%)—still owns the responsibility for security testing. Even with increasing automation, however, these small teams cannot keep up with the increased speed of delivery.

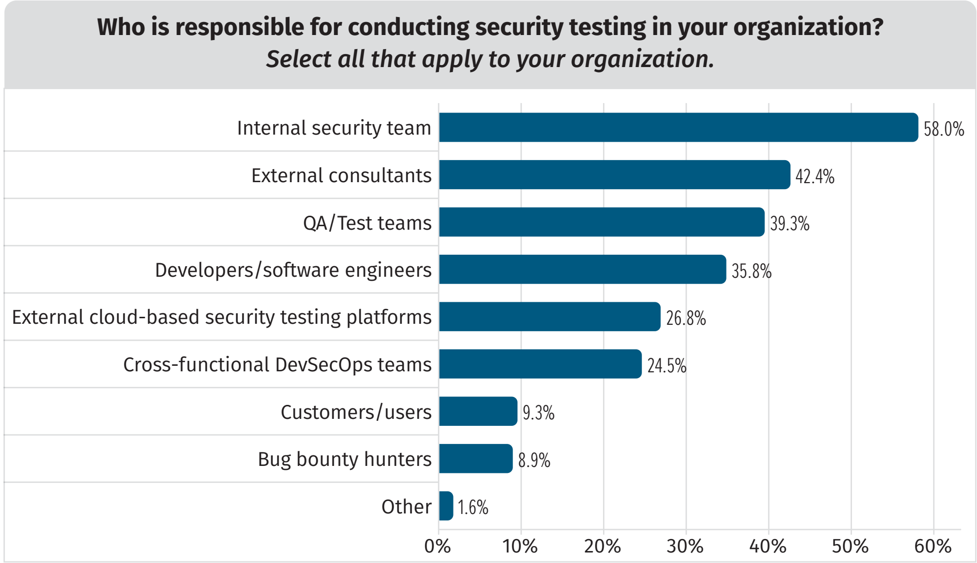
Who is responsible for conducting security testing in your organization?
Select all that apply to your organization.

Figure 13. Responsibility for Security Testing

TAKEAWAY

The key to scaling security testing is to shift this work to development or DevOps teams. However, only 36% of organizations have done this. The more that security teams understand the code, and the build and delivery processes and technologies, the more they can help DevOps teams take ownership for testing their own code.

Compliance as Code

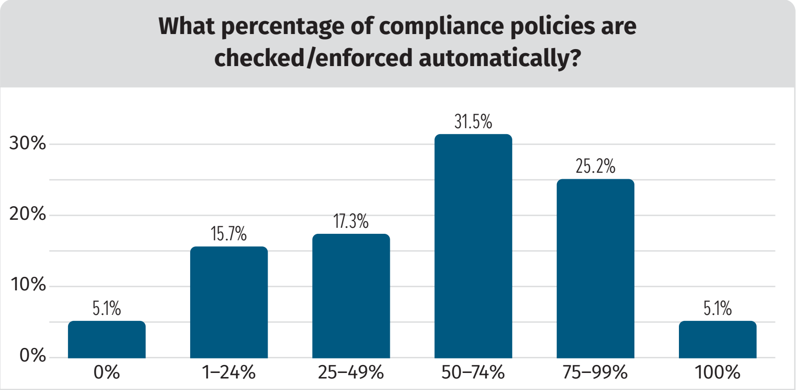

Figure 14. Percentage of Compliance Policies Enforced Automatically

Just as security is moving to code, so is compliance. Instead of following paper checklists and enduring manual audits, organizations are taking advantage of configuration in code, automated build and deployment pipelines, and automated testing to reduce the time and cost required for compliance reviews and audits, while at the same time providing a higher level of compliance assurance. DevOps teams can demonstrate their staying continuously onside of compliance, automatically checking and enforcing policies on every change. See Figure 14.

Surprisingly, more than half of organizations (62%) automatically enforce 50% or greater of their compliance policies, taking advantage of their investments in automation to perform continuous compliance.

Moving compliance into code means that DevOps teams can have more direct ownership of compliance responsibilities. Rather than reading through generic guidelines and responding to auditor requests, developers can add tests and asserts into their build pipelines and can take advantage of the built-in audit trails in version control and CI/CD to provide compliance evidence. This turns bureaucratic overhead into concrete, straightforward coding work.

Compliance in Code Tooling

In addition to the tests and automated build infrastructure that DevOps teams already work with, special-purpose tools make it easy to express compliance requirements and policies into code, at a level of abstraction that both developers and auditors can understand. Developers and auditors can write asserts, or *guardrails*. They can add these concrete pass/fail checks to build and delivery pipelines, with metadata that tie the tests back to specific policies or compliance standards, making them traceable for auditors.

Organizations now have expanding choices of high-level toolsets for authoring compliance tests. Using open source tools like AWS CloudFormation Guard, Chef InSpec, Conftest, Dev-sec.io, or Terraform Compliance, developers can write simple tests, and walk auditors through the code and the automated test results, to prove that they follow policies consistently and continuously.

Speed remains an important factor in security testing, and its criticality is even higher when it comes to resolving vulnerabilities and other security problems when found in production.

Is Vulnerability Remediation Keeping Up?

When teams find serious vulnerabilities, they need to respond and patch quickly to close the *window of exposure*. As Figure 15 shows, 71% of respondents said they patch or remediate critical vulnerabilities within 30 days (a common cutoff for regulatory compliance).

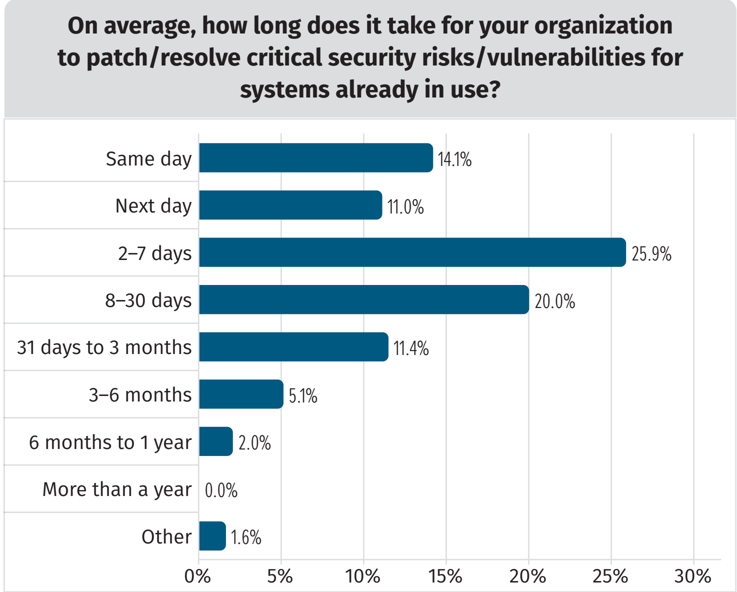

Figure 15. Time to Fix and Patch Critical Vulnerabilities

In the same way that development teams have become more agile and much faster, however, so have attackers. A 30-day patching cycle that was state of the art 20 years ago is not good enough today. Attackers exploit critical vulnerabilities within days and even hours of a vulnerability becoming known.[8] Only half (51%) of organizations patch or remediate critical security risks within a week of identifying the risk.

TAKEAWAY

To outrace attackers, organizations need to take advantage of the high-velocity delivery capabilities of modern Agile development and DevOps teams, infrastructure in code, automated builds, and CI/CD pipelines to patch code and infrastructure faster and with high confidence. Teams must also take advantage of visibility into code repos and configuration, and continuous vulnerability scanning at multiple levels, including SCA for open source dependencies, container and image scanning, scanning applications and configuration code, and cloud security posture management (CSPM), to catch vulnerabilities as early as possible.

Organizations should use virtual shielding—that is, use runtime protection solutions such as a web application firewall (WAF), next-generation WAF (NGWAF), RASP, CSPM, or CWPP service—to help minimize the risk of exploits, especially those organizations that cannot keep up with vulnerability patching.

Vulnerability management plays a vital role in security programs, but Security as Code involves much more.

DevSecOps Tools and Practices: What Works?

Organizations can use a wide range of security tools and practices to understand and manage risks, both early in development and at runtime.[9] We asked respondents to identify which tools and techniques they believe are most useful. Table 1 shows the results.

Table 1. Usefulness of Security Testing Practices/Tools and Position in Development Cycle

Practice/Technique	Very Useful	Useful	Not Useful	Total Useful
Security training for developers/engineers	29.3%	53.1%	7.8%	82.4%
Upfront risk assessments before development starts	27.0%	54.3%	5.1%	81.3%
Automated scanning of code for security vulnerabilities and other defects (SAST)	27.7%	51.2%	10.2%	78.9%
Periodic vulnerability scanning	24.2%	48.0%	15.2%	72.3%
Open source/third-party dependency analysis	24.6%	46.5%	13.7%	71.1%
Third-party penetration testing	25.0%	46.1%	13.3%	71.1%
Network detection and response (NDR)/ network traffic analysis (NTA)	21.9%	49.2%	11.7%	71.1%
Manual code review	24.6%	45.7%	18.4%	70.3%
Threat modeling, attack surface analysis, or architecture/design reviews	22.3%	48.0%	12.1%	70.3%
Continuous vulnerability scanning	27.0%	43.0%	11.3%	69.9%
WAF	29.7%	39.5%	17.6%	69.1%
Internal penetration testing	25.8%	41.4%	13.3%	67.2%
Container/image security scanning	25.4%	40.6%	12.5%	66.0%
Compliance reviews or audits by a third party	19.5%	45.7%	17.6%	65.2%
NGAF	23.8%	41.0%	10.5%	64.8%
File integrity monitoring/host-based intrusion detection system (HIDS)	21.5%	43.4%	13.7%	64.8%
Security stories, abuser stories, or evil-user stories to inject security into requirements backlog	15.2%	47.3%	12.5%	62.5%
DAST	21.1%	40.2%	10.9%	61.3%
CSPM	17.6%	38.7%	10.9%	56.3%
Virtual patching	15.6%	40.2%	13.7%	55.9%
Fuzz testing	18.8%	35.5%	11.7%	54.3%
IAST	15.2%	38.7%	11.7%	53.9%
RASP	16.4%	35.9%	11.7%	52.3%
CWPPs	15.6%	35.9%	10.9%	51.6%
Bug bounties	16.0%	34.8%	14.5%	50.8%
Other	6.6%	12.1%	5.1%	18.8%

[8] www.fireeye.com/blog/threat-research/2020/04/time-between-disclosure-patch-release-and-vulnerability-exploitation.html

[9] SANS has mapped security practices and open source tools in a secure DevOps toolchain, found at www.sans.org/security-resources/posters/cloud-security-devsecops-practices/200/download

Let’s look at these practices and techniques as they fit into DevSecOps, in order of ranking:

Secure Coding Training (82%)

Training developers/engineers in defensive coding and secure programming concepts, risks, and techniques is key to shifting responsibilities for security early into the design and coding life cycle. Developers also need security training to be effective participants in threat modeling, perform code reviews, understand and accept SAST tools, and so on.

Risk Assessments (82%)

Lightweight risk assessments identify applications/services that have compliance or security or operational risks early in design/concept stages. Development teams work with security engineers to identify sensitive information, map threat scenarios, and identify data and services that need to be protected. See Mozilla’s Rapid Risk Assessment as an example.[10]

Static Analysis (SAST) (79%)

Organizations can perform automated code scanning at multiple points in engineering workflows to catch common coding mistakes and vulnerabilities:

- High-fidelity, immediate checks in IDE using integrated plug-ins
- Custom rules in precommit hooks (high-level open source frameworks like semgrep and CodeQL make it easier to write your own checks)
- High-fidelity, fast, incremental checks in CI (time-bound)
- Checks for embedded secrets in code and repositories, later in build (e.g., git-secrets, OWASP Sedated, truffleHog)
- Deeper scans of the entire code repo on a regular frequency (e.g., nightly), outside of the build pipeline.

Different engines catch different problems, so you need multiple scanners to get high levels of confidence.

SAST should prove an easy sell to DevOps teams because it works at a level that they care about and understand. SAST also has a training effect. Over time, developers will change their coding practices in response to tool findings. But to be accepted, the tools must be fast, accurate (focused on problems that developers and engineers will actually fix), and integrated early into coding workflows.[11]

Periodic Vulnerability Scanning (72%)

Regular, periodic (e.g., monthly) automated vulnerability scans are a fundamental part of vulnerability management programs. Like

[10] https://infosec.mozilla.org/guidelines/risk/rapid_risk_assessment.html

[11] For a good overview of success factors in implementing SAST, see “How Google and Facebook do code analysis,” https://techbeacon.com/devops/how-google-facebook-do-code-analysis

other point-in-time assessments, periodic assessments leave a window of exposure open in rapidly changing environments.

SCA Dependency Analysis (71%)

Software dependency or SCA tools automatically identify licensing problems and known vulnerabilities in code dependencies (libraries and frameworks). Organizations can perform this scanning as a gating process on pull requests to identify added dependencies, and later during the build process to check for newly reported vulnerabilities.

Third-Party Penetration Testing (71%)

The effectiveness of periodic manual testing by outside experts depends heavily on testers' expertise, their familiarity with the domain and environment, and the time they have available to conduct the tests. Testing is generally done to meet compliance requirements.

Point-in-time assessments like this offer limited value in continuously changing, Agile environments, but they are one of the most effective ways to find real security vulnerabilities. Good pen testers find problems that scanners and other tests cannot. Although organizations cannot use penetration tests to verify that a system is secure, they can and should use them as a health check on the state of the secure software development life cycle (SDLC). Teams should swarm on the results, conducting a postmortem review to understand what they missed and improve their training and testing.

NDR/NTA (71%)

Network detection and response (NDR) and network traffic analysis (NTA) capabilities are important for understanding, managing, and securing east-west network communications in hybrid cloud architectures and container-based microservices. They provide deep visibility into network traffic and use machine learning to identify real-time threats and attacks. Public cloud provider traffic-mirroring services have made it easy to deploy these technologies, as part of a shift right approach to operational security.

Manual Code Reviews (70%)

Lightweight peer code reviews, done through pull requests, have become a common practice in modern software development, but their effectiveness in finding vulnerabilities depends on the reviewers' skills and knowledge, the time they have available, and their priorities. Research at organizations like Microsoft[12] and Google shows that reviewers spend more time on readability

[12] www.cabird.com/pubs/bacchelli2013eoc.pdf

and maintainability problems than looking for real defects. While readability is important (difficult-to-read code likely contains more defects and proves more risky to change), reviewers need to be trained to look for more fundamental problems—including training in secure coding and defensive coding. Manual code reviews are more effective when combined with SAST: Automated scans can find subtle problems that reviewers may not be aware of.

Threat Modeling (70%)

Threat modeling or design reviews focused on identifying security threats and risks are difficult to implement in iterative, incremental development, where the design is under continuous refinement.

Continuous Vulnerability Scanning (70%)

Continuous scanning for changing threats and vulnerabilities allows organizations to rapidly identify new risks and priorities and to track the status and success of patching programs. It involves frequent scanning and incremental analysis of results to surface changes in security posture and risks.

WAF (70%)

A WAF is a plug-in or appliance that filters out specific signatures in HTTP traffic. The major cloud platforms offer native WAF services that provide basic protection against common HTTP attacks and DDoS.

Internal Penetration Testing (67%)

Internal testers are generally more familiar with the domain and environment, but most organizations lack the technical expertise required for effective penetration testing.

Container/Image Scanning (66%)

Organizations can and should do static scanning of containers at different points for different threats:[13]

- Scanning container images or code templates (e.g., Dockerfiles) for common configuration mistakes and checks against best practices for setup and use of containers (e.g, DockerBench)
- Looking inside container image layers to identify dependencies and to identify known vulnerabilities in this software
- Scanning containers in registries for vulnerable images (now provided by most registry services)

Third-Party Compliance Reviews (65%)

Audits and assessments of control programs such as SOC 2, ISO 27001, and PCI are required for regulated industries. In DevOps environments, these assessments present challenges around continuous incremental change, CD, and "you build it, you run it," which can violate requirements for separation of responsibilities.

[13] Container scanning at these different points could be included in CWPPs.

NGAF/NGWAF (65%)

Next-generation application firewalls provide higher-fidelity, cloud-based runtime protection for web applications/APIs, which consolidates threat information with application context. The boundary between NGWAF and RASP can be blurry.

HIDS (65%)

File integrity monitoring and host-based intrusion detection systems such as Tripwire and OSSEC track changes to files and configuration data. In rapidly changing Agile environments, this results in lots of noise, making it difficult to separate out authorized changes from suspicious activity.

Security Stories (63%)

Developers can be asked to identify personas for "bad users" (e.g., fraudsters, hackers, insider threats) and write requirements (user stories) from a negative perspective, to identify threats to the system or service that need to be protected against. "As a hacker, I want to…." A variation of standard Agile practices, this encourages development teams to think outside of functional user stories and to address security requirements and risks in design and development.

DAST (61%)

Organizations can perform dynamic scanning of web applications/APIs later in the CI/CD pipeline, after the code is built and the system is provisioned for functional testing, integration testing, and acceptance testing. They can proxy these tests through a scanner to automatically execute passive and active vulnerability scans.

CSPM (56%)

CSPM tools/services automatically scan cloud instances to detect and catch common configuration mistakes and known threats and to enforce security and compliance policies. Open source examples include the following:

- Cloud Custodian: https://github.com/cloud-custodian/cloud-custodian
- Prowler: https://github.com/toniblyx/prowler
- Cloudsploit: https://github.com/aquasecurity/cloudsploit

In any cloud environment, we have too many configuration options and too many compliance and security policies to review manually.

Virtual Patching (56%)

Virtual patching refers to applying protection in WAFs, RASP, and CWPP at runtime to block known attack signatures as a rapid response to newly discovered vulnerabilities. Organizations can use virtual patching as a temporary stopgap until they can apply and roll out a patch to the software.

Fuzzing (54%)

Fuzzing refers to automated negative testing for APIs, file-based applications, and protocols. Security researchers use fuzzing because it can find real vulnerabilities with few false positives. It proves especially useful and important for applications written in memory-unsafe languages like C/C++ (although fuzzing is also available for languages such as Go, Rust, Java, and Python). Fuzzing can be difficult to add into CI/CD, because good fuzzing takes time to set up, run, and interpret the test results. However, modern cloud-based fuzzing services make this easier, using limited test cases in the build pipeline (for example, GitLab now offers fuzzing options) or continuously outside of CI/CD.

IAST (54%)

IAST instruments the application runtime and detects application vulnerabilities as the system or service runs. This passive testing technique depends on other tests (e.g., functional tests) to exercise the code.

RASP (54%)

RASP instruments the application runtime (e.g., JVM or .NET CLR) to provide operational visibility into attacks and to defend against common application security attacks and language/framework-specific vulnerabilities. RASP adds some runtime overhead and operational complexity in return for runtime protection.

CWPP (52%)

CWPPs provide runtime protection for containers and cloud instances. This broad technology category covers services that offer different levels and types of runtime shielding, including network segmentation, system integrity protection, application control/whitelisting, behavioral monitoring, host-based intrusion prevention, and optional anti-malware protection. Open source examples include the following:

- Falco for runtime security: https://github.com/falcosecurity/falco
- Open source sysdig for troubleshooting and incident response: https://github.com/draios/sysdig

Bug Bounties (51%)

In these wider-scale programs, organizations invited outside white hat testers (security researchers, ethical hackers) to continuously test systems and report vulnerabilities, on a reward basis. Bug bounty programs at organizations like Google, Apple, Microsoft, and Facebook attract a lot of attention, but they also require a lot of work to set up and manage.

Keys to DevSecOps Success

Finally, we want to look at the key drivers, the enablers and barriers to success, of DevSecOps programs and how to measure a program’s success and risks.

KPIs

Measurement and feedback loops, using data to drive decisions, represent a fundamental part of DevSecOps. See Figure 16.

DevSecOps programs have two broad categories of metrics:

- **Vulnerability management—**The number of open vulnerabilities, how long they stay open, where vulnerabilities are found (early in development and build, or later), false positive rates, how long it takes to detect vulnerabilities, and how much it costs to fix them. Organizations rely on these metrics to understand security risks and trends.
- **DevOps delivery—**How often builds are delayed or fail due to security vulnerabilities, automated test coverage, and change lead time for delivery. These metrics can help the organization to understand the costs of their security program and how to address security risks in DevOps.

What are the major KPIs you use to measure the success of your DevSecOps activities? ***Select all that apply.***

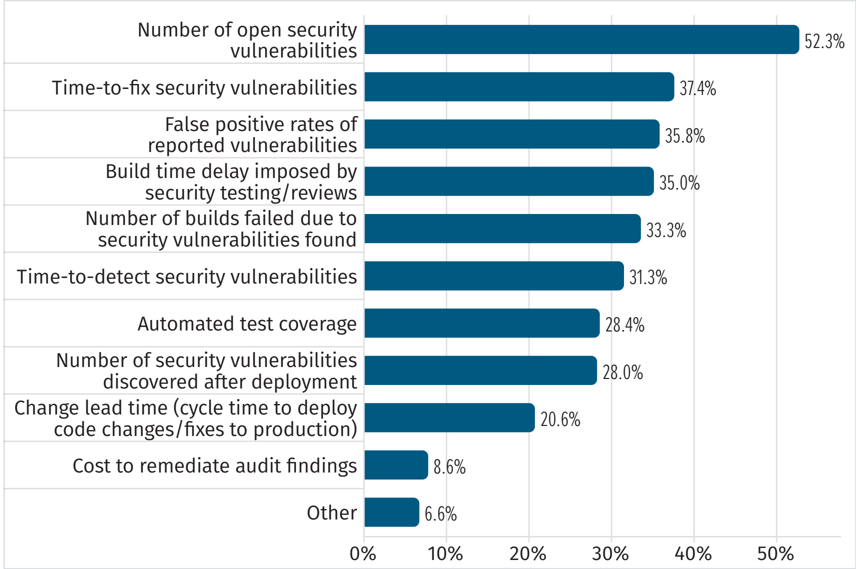

Figure 16. KPIs in Use to Measure DevSecOps

Organizations still have a long way to go in implementing comprehensive metrics programs. Only half (52%) use basic vulnerability measurements to understand and manage risks. Roughly a third of respondents collect and use data on delivery (false positives, delays, and failures).

TAKEAWAY

Optimizing DevOps efficiency and velocity not only reduces IT costs and improves an organization’s agility, it also helps improve the organization’s security posture:

- ✓ **Change lead time/cycle time—The speed at which teams deliver changes is the heartbeat of DevOps. It also determines the organization’s ability to push out security patches.**
- ✓ **Build time delays caused by security testing and number of builds failed due to vulnerabilities found—Organizations can use these to improve the effectiveness of security controls added to delivery (to find vulnerabilities early) without degrading the team’s velocity of delivery.**
- ✓ **False positive rates of reported vulnerabilities—This important measure of efficiency improves the fidelity of scanning and reporting, minimizing noise, which allows security and DevOps teams to focus on real risks.**
- ✓ **Automated test coverage—This involves identifying how much automated tests in the build pipeline protect the code base (or not). As discussed earlier, this metric helps you understand where your blind spots are in testing and compliance. Test coverage also determines the organization’s confidence in its ability to roll out patches successfully and how challenging it will be to add security testing into build and delivery pipelines.**

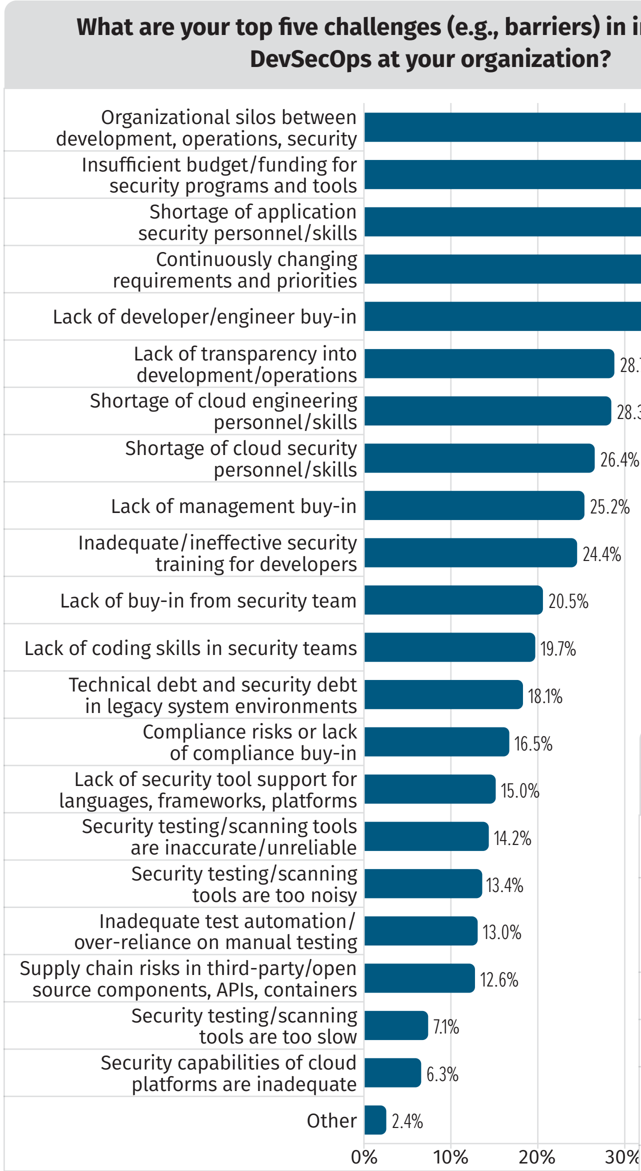

Figure 17. Top Challenges in Implementing DevSecOps

The success of DevSecOps programs involves both technical and organizational factors. See Figure 17.

While tools could be faster and more accurate and easier to use, technology (that is, the capabilities and speed of tools and the capabilities of cloud service providers) does not hold organizations back. As shown in Figure 17, the major challenges to implementing DevSecOps successfully continue to be organizational: siloed specialties, insufficient funding for security programs, shortage of skills, continuously changing priorities and requirements, and lack of buy-in from the different stakeholders.

Management commitment and organizational changes also serve as key drivers to success (see Figure 18). Improving communications across organizational silos, securing management buy-in and buy-in from developers and engineers, and sharing goals and KPIs across teams, are all key nontechnical success factors.

Success starts with training developers and engineers in security concepts and secure coding. This needs buy-in from management and developers, and improved communications across specialties, before you can get to the heavy-lifting work: integrating automated testing into build chains, automating builds, and enforcing security and compliance policies in code.

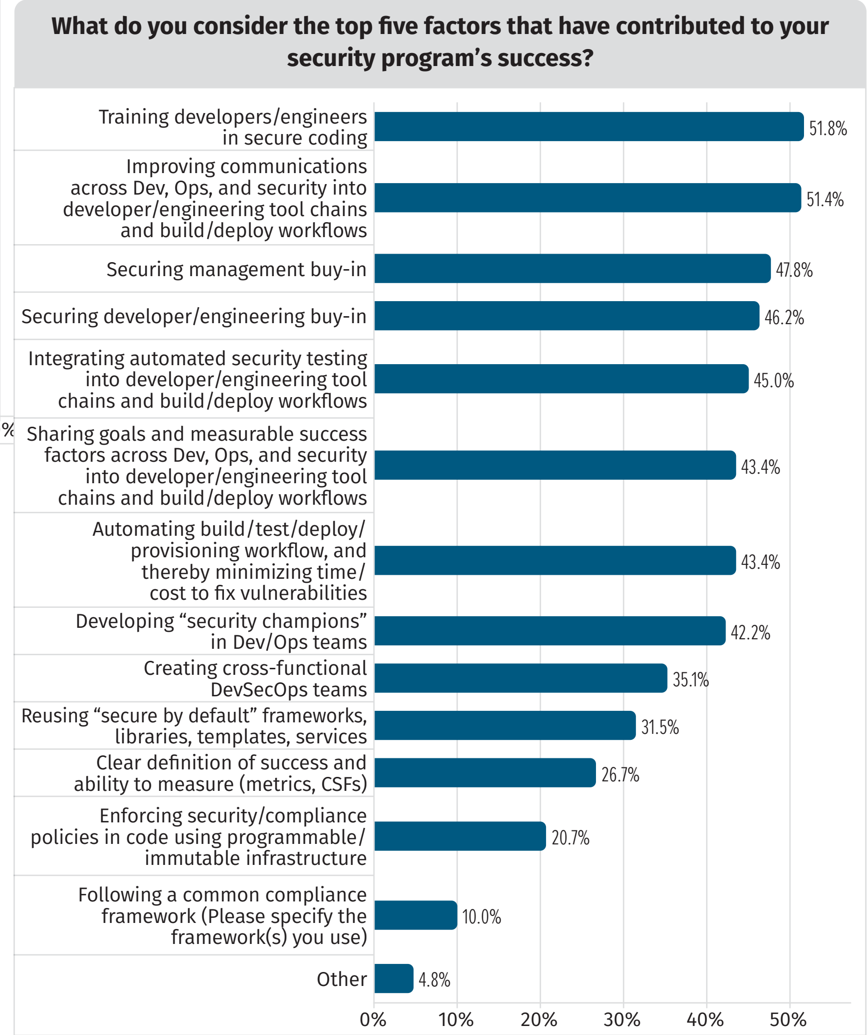

Figure 18. DevSecOps Success Factors

Conclusions/Moving Forward

Security as Code consists of three parts:

- **Security in coding—**Application security practices and techniques to secure the SDLC, training developers on secure coding practices, building secure by default frameworks and libraries, and integrating security checks inside the workflow and build pipelines
- **Coding in security—**Understanding the SDLC workflows and tools and how to use them for security and compliance purposes, moving from guidelines and checklists to guardrails and automated tests; as security teams mature, leveraging version control and CI/CD to deploy detection, alerting, and response solutions
- **Securing the code—**Treating the code repos and build and test toolchains and infrastructure as part of the organization's attack surface; securing the code and toolchains and infrastructure from end to end; getting visibility and control over what code is changing and who can change it

Code becomes the common language and currency between dev, ops, security, and compliance. It brings together these different specialties, enabling collaboration and improving transparency.

With Security as Code, training requirements change. We have focused on addressing developer skill gaps around security. Now we need to address coding skill gaps for security specialists.

Fundamentals of Security as Code

Moving security into code involves a significant learning curve. Security engineers need to understand a long and constantly changing list of things, including multiple cloud service platform services and APIs, containers and VMs, serverless architectures, templating languages, and scanning tools.

However, to work with developers and engineers, and to move security into code, you need to start with understanding the mechanics of modern software development, including version control concepts, branching and merging, Git workflows, automated build pipelines, and CI and CD. This process offers a key opportunity to work with developers—to learn and to build bridges.

An ongoing transformation of software development and operations engineering focuses on speed, iteration, and confidence. The same transformation needs to occur for security: achieving speed and agility through coding and automation; becoming more iterative, risk based, and pragmatic; and continuously learning and adapting.

Putting security into code allows organizations to accelerate and scale. Organizations can reuse policies and controls across services and teams (and even outside of the organization). The communities built up around open source toolsets like semgrep, CodeQL, and Open Policy Agent attest to their success.

To achieve this, security teams need to collaborate with developers and operations engineers and share their tools and practices. Security engineers need to learn to think and work like developers and apply the same Agile, incremental methods to continuously improve the security program. Organizations must recognize and reconcile this fundamental tension between agility and control.

With Security as Code, organizations face enormous challenges but will also benefit from the associated potential to accelerate, learn, and scale.

About the Authoring Team

Jim Bird, SANS analyst and co-author of SEC540: Cloud Security & DevOps Automation, is an active contributor to the Open Web Application Security Project (OWASP) and an author of books on agile security and secure DevOps. He has worked at major technology organizations and financial institutions around the world in management, software development, IT and business operations, and security.

Eric Johnson is Principal Security Engineer at Puma Security and Senior SANS Instructor focusing on cloud security, DevSecOps automation, and building static analysis tools. His experience includes cloud infrastructure and security automation, cloud security assessment, static source code analysis, web and mobile application penetration testing, security development lifecycle consulting, and secure code review assessments.

Frank Kim leads the management and software security curricula for SANS, developing courses on strategic planning, leadership, and application security. He is also a SANS certified instructor, helping to shape, develop, and support the next generation of security leaders. Previously, Frank served as CISO at the SANS Institute, leading its information risk function, and as executive director of cybersecurity at Kaiser Permanente, where he built an innovative security program to serve one of the nation's largest not-for-profit health plans and integrated healthcare provider. Currently, as founder of ThinkSec, a security consulting and CISO advisory firm, Frank helps leaders develop business-driven security programs.

Sponsor

SANS would like to thank this paper's sponsor:

MICRO FOCUS