2018 Secure DevOps: Fact or Fiction?

Jim Bird
Executive Summary

From large-scale multiyear waterfall projects to small two-pizza teams delivering and running online microservices, how we manage software has changed completely. Just as Capability Maturity Model Integration (CMMI) and Project Management Institute (PMI) gave way to Agile software development, Agile has made way for DevOps, taking Agile principles and tooling to full chain, from design to development to deployment and operations.

DevOps started in the cloud with digital startups such as Netflix and Flickr. Today, DevOps is becoming fact, not fiction, for many organizations, as enterprises and even governments adopt cloud services and containers, and implement automated continuous build and delivery pipelines. Today it’s not just about cutting delivery times and costs and risks. It’s about building iterative feedback loops between engineering and real users, running continuous experiments so that the organization can learn and adapt, fail fast or pivot quickly in response to rapidly emerging needs and priorities. It’s also about taking advantage of chaos theory to design and implement resilient, anti-fragile operational environments that can scale up and scale out on demand. And it’s about leveraging automation to cut delivery cycle time while at the same time reducing operational risk and satisfying compliance requirements.

This survey, the sixth in a series of annual studies by SANS on security practices in software development, is the first to explicitly focus on DevOps. Last year we looked at how organizations balanced speed of delivery against risk. This year we dove deep into how security fits into DevOps, where security risks are and how they are being managed, and the top success factors in implementing a Secure DevOps program:

- Have organizations been successful in adapting, or reinventing, their traditional approaches to InfoSec and AppSec to the realities of DevOps?
- Can security keep up with rapid iteration in continuous deployment?
- Is Secure DevOps fact or fiction?

The results of this study show that organizations are finding ways to keep up with rapid change through DevOps—but they have a number of challenges they still need to deal with.

New technology platforms and tooling in DevOps offer compelling new advantages—while at the same time creating new security risks that need to be understood and managed. But organizations must recognize and pay off the security debt that they have already allowed to build up, and must solve their legacy security problems, before they can safely move forward.

DevOps and DevSecOps

DevOps applies the values, ideas and practices from Agile and Lean product development methods to the full value chain of IT service delivery: from ideation to implementation and operations. It demands a fundamental change in the way that IT organizations think and work. Instead of managing IT projects and handing the “finished” system off to maintenance and operations to get it running and keep it running, cross-functional DevOps teams build, deliver and operate end-to-end services. DevOps breaks down the walls (organizational, communication and cultural) between developers, operations and the business.

Secure DevOps (DevSecOps) extends the same collaborative and iterative practices, tools and values to information security, making security an integral part of service delivery.

Key Findings

- The majority of our 229 respondents reported that their organizations have already moved applications to the cloud, and are in the process of adopting containers and serverless platforms.
- While the velocity of service delivery continues to increase, the frequency of security assessments is increasing to match.
- Fewer than half (46%) of respondents have “shifted left” to confront security risks up front in requirements and service design, which means that they are ignoring important risks and taking on unnecessary (and dangerous) security debt.
- Although most organizations are taking advantage of DevOps build/deployment pipelines and continuous delivery to patch vulnerabilities faster, only half of them are actually fixing the majority of vulnerabilities.
- Most organizations are still fighting significant legacy and technical debt issues.
In their quest to deliver working software faster-better-cheaper, DevOps teams are continuously experimenting with new platforms, from increasingly sophisticated and complex cloud services, to containers and unikernels, and now serverless computing. Engineers are still grappling with how to leverage these technologies across the software development life cycle (SDLC)—from design to testing, from how to configure and deploy applications in these environments to how to monitor and support them. This is a new world that the security industry is still trying to catch up to, that security teams must accept and understand … and a world that is still changing rapidly. Figure 1 shows that:

- A surprising number of organizations are already deploying serverless apps. This should be a concern for security teams, because security risks in serverless environments are not widely understood, and recognized serverless security guidelines are yet to be developed.
- Containers are still an emerging technology. Only 24% of respondents have 26% or more applications in containers with most still at 1% to 5%. But, as we shall see, the use is growing.
- The majority of respondents (43%) have 26% or more of their apps in the cloud.

Defending cloud-based and container-based applications also continues to grow as a much bigger part of application security programs. Seventy-three percent of respondents, a 30% increase from last year, are defending applications in public cloud, and 34% expect to do so in the next 12 months. Sixty-five percent are already involved in securing applications in containers (up from 28% in 2017), while 45% expect that this will increase in the next 12 months. See Figure 2 on the next page.

However, while engineering teams are busy working with tomorrow’s technologies, security teams are still focused on defending existing legacy applications—fighting yesterday’s battles. Ninety-five percent are working at protecting legacy applications.
Securing these older apps has an important tie-in to organizational risk: Legacy apps are considered risky by 55% of survey respondents and are involved in 14% of incidents or breaches. Technical debt and security debt in legacy system environments is also a leading challenge that respondents (28%) report they face in implementing Secure DevOps at their organization.

Application Security Risks Today

Risk and the assessment of risk are key drivers for the investment an organization makes in its security posture and culture. In today’s environment, application security is a critical concern, not only for the attack surfaces that applications present, but also because many organizations are bound by constraints around privacy and access, federal regulations, and mandated oversight.

While organizations need to understand and manage new risks in cloud platforms and containers and serverless apps, they still have to deal effectively with legacy system vulnerabilities and legacy operational risks. Public-facing web apps pose the highest security risk to organizations, a fact that has not changed since our 2012 survey, and legacy apps are another major target of attacks. Commercial off-the-shelf applications, mobile apps and applications in private clouds are involved in a significant number of incidents, although they are not always seen as major security risks within those organizations (see Table 1).

This latter observation raises a concern as to whether AppSec risk programs effectively align perception with reality. It may take an event, such as a highly public breach, for organizations to be fully aware of the security risks inherent in a technology area.

### What types of applications are you currently protecting under your security program?

**Select those that most apply.**

<table>
<thead>
<tr>
<th>Application</th>
<th>Considered Risky</th>
<th>Involved in an Incident/Breach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public-facing web applications</td>
<td>59.2%</td>
<td>15.5%</td>
</tr>
<tr>
<td>Legacy applications</td>
<td>54.9%</td>
<td>14.1%</td>
</tr>
<tr>
<td>Commercial off-the-shelf applications</td>
<td>39.4%</td>
<td>12.7%</td>
</tr>
<tr>
<td>Mobile business applications</td>
<td>39.4%</td>
<td>11.3%</td>
</tr>
<tr>
<td>Applications in a private cloud/virtual environment</td>
<td>39.4%</td>
<td>9.9%</td>
</tr>
<tr>
<td>Third-party/open source dependencies</td>
<td>59.2%</td>
<td>8.5%</td>
</tr>
<tr>
<td>Applications hosted in the public cloud</td>
<td>54.9%</td>
<td>7.0%</td>
</tr>
<tr>
<td>IoT applications</td>
<td>52.1%</td>
<td>7.0%</td>
</tr>
<tr>
<td>APIs (commercially developed)</td>
<td>43.7%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Embedded applications</td>
<td>36.6%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Serverless functions</td>
<td>38.0%</td>
<td>5.6%</td>
</tr>
<tr>
<td>APIs (developed in-house)</td>
<td>52.1%</td>
<td>4.2%</td>
</tr>
<tr>
<td>Containerized applications (Docker, rkt)</td>
<td>45.1%</td>
<td>2.8%</td>
</tr>
</tbody>
</table>
Not surprisingly, application risk is also tied to the language or platform upon which the application portfolio is based, and the more popular languages used by developers also offer the greatest source of risk or exposure.\(^1\) Java continues to be one of the more popular development platforms, especially for legacy systems, commercial off-the-shelf applications, browser plug-ins and third-party libraries—and the riskiest to use according to 59% of respondents. Although Oracle has taken steps to improve the security of Java, especially in browsers, the Java JDK and JRE continue to be popular targets for attackers (see Figure 3).

JavaScript came in second (45%), as it continues to increase in popularity, both on the server (Node) and on the client side (popular user interface frameworks such as Angular and React). This year, Android finished third (nearly 32%), being seen as a serious source of risk due to Google’s announcement in 2017 that it would support Kotlin (a new JVM language) as a first-class language for developing Android apps. Given the number of legacy apps built on .Net and PHP, it should come as no surprise that these two rounded out the top four, each being selected by 30% of respondents.

### The Equifax Effect

Heartbleed and Shellshock opened many people’s eyes to the risks of vulnerabilities in open source code. But it was the massive Equifax breach in 2017 that showed how serious this risk really was. Attackers exploited a widely publicized remote execution vulnerability in the Apache Struts web framework that had been unpatched in Equifax’s systems for more than two months, and stole personal credit details of over 147 million (and counting) people.

The costs and consequences of this breach are still not completely understood. But it made it clear that security risks in open source components must be taken seriously, and that scanning and patching these components should be an essential part of any security program.

### Increased Java Security Risks

The security risks of developing in Java may continue to increase in the future, due to recent changes that Oracle has announced to Java’s release management and support policies. Teams that want to take advantage of new language features will need to jump on Oracle’s rapid six-month feature release cycle. But as of January 2019, teams that want security patches to a stable and proven Java runtime will need to pay for long-term support from Oracle. Organizations that can’t afford support contracts must stay on JDK 8 without security updates, or consider switching to the community-supported OpenJDK and hope that the community will step up to provide security patches.

---

\(^1\) The TIOBE Community Index is a useful, public measure of the relative popularity of different programming languages: [www.tiobe.com/tiobe-index](http://www.tiobe.com/tiobe-index)
DevOps is based on, and enables, rapid and iterative change. The speed at which teams deliver continues to increase year over year. In 2017, 60% deployed system changes to production apps more than once per month, whereas in 2018 this increased to 66%. While weekly and daily deployments have dropped, the percentage of those applying system changes on a continuous basis doubled from 5% to 10%. Almost half (47%) of respondents report that their organizations are continuously deploying at least some apps directly to production, forcing security teams to make radical changes to how they work in order to keep up—or risk being left behind. See Figure 4.

The faster that engineering teams deliver to production, the faster that security teams must be able to identify and understand security risks and help manage them. They can do this by leveraging common core DevOps practices and techniques: automated builds, automated testing, automated deployment, programmable configuration management (infrastructure as code), proactive monitoring (measurement of systems in production), and immutable infrastructure provisioning. These are all important touchpoints for security teams to engage with engineering, and where security checks and testing can be added directly into engineering workflows. Of these, we see that automation for build and deployment are rated the highest by those respondents who have adopted a continuous approach to assessment of the security of their business-critical applications. See Figure 5.

---

**Figure 4. Frequency of Deploying System Changes to Production Apps (2017 vs. 2018)**

**Figure 5. Frequency of Assessment vs. Best Practices Used**
According to this survey, the DevOps security teams are finding ways to keep up. The number of organizations assessing or testing the security of business-critical applications more than once per month has increased from 13% in 2017 to 24% in 2018, and those testing daily and continuously have almost doubled over the same period (see Figure 6).

We Are Getting Faster, But Are We Getting Better?

One of the key value propositions of DevOps and automated build and delivery pipelines is that engineering teams can deploy fixes into production faster, cheaper and safer. The ability to patch vulnerabilities faster is considered a key “defensive weapon” in the DevOps team arsenal.

Comparing 2018 with 2017, we see a marginal increase in the percentage of critical security vulnerabilities repaired satisfactorily and in a timely manner. In both 2017 and 2018, 70% were able to repair more than 50% of vulnerabilities satisfactorily. In 2018, 51% repaired more than 75% as opposed to 50% in 2017. See Figure 7.

The typical window of vulnerability does not provide enough time for patch installation. And this is an ongoing concern, as the time between a vulnerability being uncovered and an automated attack against it being developed and released is decreasing every year.2

---

But is the corresponding velocity to patch really helping to close the “window of vulnerability?” From our survey results, the answer appears to be no improvement or, at the very least, not enough. In 2018, 54% of respondents reported that they could patch a critical application security vulnerability in seven days or less for systems already in use, as opposed to 41% in 2017. However, when patch times are two days or less, we see a decrease in performance from 2017 to 2018. See Figure 8.

**Figure 8. Time to Fix and Deploy a Patch to Critical Application Security Vulnerability (2017 vs. 2018)**

Getting There—Remediation and Practice

Security vulnerabilities can be remediated in many different ways:

- Patching or upgrading software components, including upgrading third-party or open source software or updating containers
- Modifying the configuration through updates or changes to the operating environment, network architecture and other runtime protection mechanisms (including the creation of new web application firewall [WAF] rules)
- Making “quick and dirty” temporary fixes
- Leaning on runtime defensive technologies such as firewalls, Runtime Application Self-Protection (RASP) or virtual patching
- Disabling functions or decommissioning legacy apps to reduce the attack surface
- Making fundamental root cause corrections and improvements to how services are delivered, including applying secure SDLC practices and tooling. This approach is the most effective over the long term, but also the most expensive, and requires the commitment of engineering and the support of management.

Looking at those respondents who reported deploying system changes to production on a continuous basis (including daily), we see these organizations favor root cause analysis, followed by an approach that emphasizes updates either to the environment (operating system, network, runtime environment) or to relevant software components (third-party or open source). See Figure 9 on next page.
These approaches to risk management and remediation rely on familiar and fundamental practices inherent to all secure application development, and are not specific to DevOps. But they need to be adapted to DevOps constraints and priorities, implemented in ways that are as efficient and frictionless as possible.

Table 2 ranks the application security tools, practices or techniques used in respondent organizations, in terms of overall usefulness, with 1 being the most useful. Not surprisingly (and taking into account a natural bias in the survey base drawn largely from the SANS community, which views training as an integral part of all security programs), respondents rated security training for engineers to be the most useful practice.

Significantly more organizations are addressing risks by upgrading third-party/open source dependencies (an increase from 47% in 2017 to 64% to 2018). This is another sign of the “Equifax Effect.”

Table 2. Application Security Tools, Practices or Techniques Ranked by Most to Least Useful

<table>
<thead>
<tr>
<th>Ranking</th>
<th>Tools, Practices or Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Security training for developers/engineers</td>
</tr>
<tr>
<td>2</td>
<td>Continuous vulnerability scanning</td>
</tr>
<tr>
<td>3</td>
<td>Threat modeling, attack surface analysis or architecture/design reviews</td>
</tr>
<tr>
<td>4</td>
<td>Continuous monitoring for signs of attacks and indicators of compromise (IOCs)</td>
</tr>
<tr>
<td>5</td>
<td>Periodic vulnerability scanning</td>
</tr>
<tr>
<td>6</td>
<td>Internal penetration testing</td>
</tr>
<tr>
<td>7</td>
<td>Dynamic Application Security Testing (DAST)</td>
</tr>
<tr>
<td>8</td>
<td>Upfront risk assessments before development starts</td>
</tr>
<tr>
<td>9</td>
<td>Open source/third-party dependency analysis</td>
</tr>
<tr>
<td>10</td>
<td>Web Application Firewall (WAF)</td>
</tr>
<tr>
<td>11</td>
<td>Third-party penetration testing</td>
</tr>
<tr>
<td>12</td>
<td>Container/image security scanning</td>
</tr>
<tr>
<td>13</td>
<td>Compliance reviews or audits by a third party</td>
</tr>
<tr>
<td>14</td>
<td>Manual code review</td>
</tr>
<tr>
<td>15</td>
<td>Security stories, abuser stories or evil user stories to inject security into requirements backlog</td>
</tr>
<tr>
<td>16</td>
<td>Next-Generation Web Application Firewall (NGWAF)</td>
</tr>
<tr>
<td>17</td>
<td>Interactive Application Security Testing (IAST)</td>
</tr>
<tr>
<td>18</td>
<td>Virtual patching</td>
</tr>
<tr>
<td>19</td>
<td>Runtime Application Self-Protection (RASP)</td>
</tr>
<tr>
<td>20</td>
<td>Bug bounties</td>
</tr>
</tbody>
</table>

(Duplicate numbers represent a tie.)
Rapid iteration and fast cycling in DevOps makes continuous vulnerability monitoring and scanning (of code, configuration and infrastructure) a necessity. Threat modeling, an important tool for finding problems early in design, ranked higher than in previous surveys, despite upfront assessment of security risks being done only in roughly half of respondent organizations.

Practices and tools such as bug bounties, RASP, IAST, and NGWAF get media attention but are still new to most organizations. Organizations need more time to work with these techniques and tools and understand how and when to take advantage of them.

**Shifting Left: Engaging Security in DevOps**

Security needs to be considered throughout the development and delivery lifecycle. In DevOps terms, this requires “shifting left” to include security in all stages of the delivery chain, from the business case and requirements, to planning and platform selection, to service design, coding, deployment and operations. Our respondent population definitely engages information security resources during the early phases of inception and requirements (see Figure 10).

However, 53% of respondent organizations still wait to engage their information security resources until later stages in development, ignoring the advantage of early involvement, where important design flaws and missed security or privacy requirements can be caught and prevented.

The point at which SDLC security is engaged influences what tools and techniques teams find useful in DevOps environments. As teams “shift left,” bringing security into play earlier, at the inception and requirements stages, they get more value from the common core DevOps

**Best Practices: Secure DevOps**

- AppSec training for engineers is a fundamental prerequisite for self-service security in DevSecOps, and an important enabler of the cultural changes that must be made.
- Continuous vulnerability scanning can be (and should be) embedded into automated build/deployment pipelines in continuous integration and continuous delivery to catch problems as soon as they are introduced.
- DevOps teams build continuous monitoring feedback loops to run A/B experiments in production and to detect operational problems quickly. The same practices can be used to identify attacks and IOCs.
practices and techniques discussed previously: automated builds, automated testing, automated deployment, programmable configuration management (infrastructure as code), proactive monitoring (measurement of systems in production), and immutable infrastructure provisioning (see Figure 11).

Fewer than half of organizations (47%) are using infrastructure as code/programmable infrastructure (using toolsets such as Chef, Puppet, Terraform and AWS Cloud Formation). This is partly because many organizations, especially enterprises, are still dealing with legacy configuration management issues and following legacy approaches to managing infrastructure. As they roll out new solutions, organizations today can leapfrog over many of the problems in infrastructure configuration management by catching the container-as-a-service wave (using platforms such as AWS Fargate, or Google Kubernetes Engine) or adopting serverless platforms from AWS, Google Cloud, Microsoft and others.

Who’s Responsible?

If DevOps teams are capable of fixing vulnerabilities faster, cheaper and more safely—why aren’t they? Fundamentally, this is not a problem of technical capability or engineering competence. It’s a problem of resourcing and prioritization: in other words, a management problem. Management decides what work gets done—and what doesn’t. If management doesn’t understand, or recognize, the need to address critical security risks, engineering and compliance and information security teams won’t be given the and resources they require.

In a DevOps culture, these problems can extend into how teams are structured and how they work. Engineering teams are held responsible for keeping their own code secure while the majority of the work to correct vulnerabilities is done by developers with help or oversight provided by the security team, whose members may be distanced from engineering by virtue of training or role definition (e.g., network-centric as opposed to app-centric).
Figure 12 shows the breakdown of responsibility for 1) managing the application security testing (e.g., setting policy), 2) conducting the testing, 3) accepting the final testing results and 4) performing any corrective actions resulting from that testing.

Based on this information, we found:

- **Managing.** This function is mostly under the direction of the internal security team (60%) but business owners retain a significant stake (49%). The role of business unit owners is, in our opinion, surprisingly high but may reflect a needed direct involvement of systems users and stakeholders in the security of the application.

- **Conducting testing.** Most assessments are being done by internal security teams (61%), followed by cross-functional teams (55%) and then external consultants (54%). We expect that cross-functional DevSecOps teams will be taking on more of this work through “self-service” security testing and scanning models.

- **Accepting.** According to 61% of respondents, the business unit owner is responsible for accepting. This should be no surprise. This role sets priorities, trading off security/operational risk against delivery time/cost. However, this role may make decisions without understanding the consequences, going back to the fundamental problem faced by many organizations with the lack of understanding by management in establishing funding and resource allocations related to application security.

- **Corrective Action.** According to 65% of respondents, corrective actions are in the hands of developers. This helps explain why vulnerabilities don’t always get fixed: Developers are forced into a difficult situation, under conflicting pressures to deliver changes quickly and cheaply, while also being held responsible for fixing vulnerabilities and other bugs.

Roles affect what ongoing security processes are considered the most useful, however slight. Although overall results support security training for developers and engineers as most important, continuous monitoring also is ranked as highest by cross-functional teams involved in accepting and corrective actions and internal security teams involved with managing. This supports the expectation that cross-functional teams rely more on automated, iterative/incremental techniques that fit into build and delivery pipelines, or that can be made part of engineering workflows/practices.
Making Secure DevOps Programs Effective

The challenges or barriers to successful application security programs fall into one of two major categories: organizational or technical.

From the data we see that lack of tool support and inadequate test automation are not holding security programs back, although a significant percentage of organizations report that security scanning/testing tools are too slow to support continuous build/deployment pipelines.

The biggest problems are more fundamental, and organizational: shortage of skills, inadequate budgets, poor prioritization, lack of management buy-in—and the crushing weight of technical debt and security debt built up from taking too many shortcuts and fast-tracking delivery in the past (see Figure 13).

### Technical Debt and Security Debt

Technical debt is a recognized fact of modern development. Iterative, incremental design and development is erosive—and the rapid feedback-response cycles in DevOps amplify this effect. As the requirements and design emerge from incremental code changes and feedback from users, developers make false starts and sudden changes in direction, and mistakes in assumptions and implementation.

Teams under constant pressure to deliver are forced to take shortcuts, and don’t get time to clean up the code or fix bugs. The code gets bigger and buggier, more complex, harder to understand, and riskier to change. They are building up technical debt: debt that must be paid off sooner (by refactoring and fixing the code as they work) or later (by throwing the system out and starting over again).

Security debt works on the same principles. If dev or ops teams don’t get a chance to fix vulnerabilities when they are found, to patch libraries and runtimes, and add defensive safeguards, the risk of a security breach compounds. Organizations that don’t engage security early in requirements and design and that don’t fix vulnerabilities when they are found are writing checks that they won’t be able to cover.
DevOps and DevSecOps—by breaking down silos between technical disciplines, improving transparency and collaboration, and introducing better technical practices and tools—can overcome many organizational barriers to success. But grassroots initiatives require top-down management support to effect real change and to scale effectively. As seen in Figure 14, the top three factors that have contributed to respondents’ success are:

1. Integrating automated security testing into build/delivery tools/workflows (38%)
2. Developing security champions in dev/ops teams (35%)
3. Training engineers on secure development/operations (30%)

Putting responsibility for security into the hands of DevOps delivery teams—and providing self-service security assessment tools and safeguards—are the only practical ways to scale an application security program. Ultimately, like the challenges, the success factors are also organizational in nature and require:

- Making security work transparent and adding it directly into engineering backlogs
- Getting dev/sec/ops teams to work together and integrate security testing into workflows and pipelines
- Making engineers responsible for building secure code, and giving them the training and the tools they need to accomplish this
- Creating security champions at all levels throughout the organization
- Improving communication and collaboration between dev/sec/ops and management, and building cross-functional teams across silos

### Building a DevSecOps Program: Stay CALMS

CALMS is an acronym coined by John Willis, Damon Edwards, and (later) Jez Humble to describe the keys to DevOps: Culture, Automation, Lean, Measurement, and Sharing. CALMS can also be used to define the pillars of a successful Secure DevOps program:

- **Culture.** Break down barriers between development, security, and operations through education and outreach.
- **Automation.** Embed self-service automated security scanning and testing in engineering build and delivery pipelines and workflows.
- **Lean.** Perform value stream analysis on security and compliance processes to optimize flow and eliminate waste.
- **Measurement.** Use data to shape design and to drive decisions.
- **Sharing.** Share information on threats, risks and vulnerabilities by adding them to engineering backlogs.

---

3 "SANS Secure DevSecOps" poster, January 2018, [www.sans.org/security-resources/posters/secure-devops-toolchain-swat-checklist/60/download](http://www.sans.org/security-resources/posters/secure-devops-toolchain-swat-checklist/60/download)
Metrics or “You Can’t Improve What You Don’t Measure”

It is often said, “You can’t improve what you can’t measure.” And for that, you need both visibility into the supporting processes (i.e., workflows) and the proper metrics (i.e., indicators).

According to respondents, the key metrics for our respondent’s security programs are (in order):

1. Time to fix vulnerabilities
2. Number of security issues discovered after deployment
   (vulnerability escape rates)
3. Builds delayed due to security issues
3. Post-audit remediation steps required (tie)
4. Human hours spent resolving security issues

All of these metrics are about delivery velocity. The first two focus on the organization’s capability to respond to problems, and the risks and costs of moving too fast in delivery, calculated by how many vulnerabilities escape the delivery cycle to be caught in production.

Looking at the major KPIs used to measure success versus when security is introduced in the SDLC provides some interesting insights on the use of metrics. Inception and requirements phases favor more traditional metrics such as time to fix security issues and number of issues discovered after deployment. Looking at later phases of the SDLC, however, these metrics may not be as critical as the other key metrics, which may more accurately indicate drags on velocity: measures of inefficiency, friction and waste caused by poorly implemented security programs. See Figure 15.

Are security controls unnecessarily delaying builds? And what are the human costs involved in remediating audit findings or dealing with other security problems that aren’t being prevented or caught in the delivery value stream? Using Lean Engineering techniques, DevOps teams can feed these metrics into continuous improvement feedback loops to identify bottlenecks and delays, and then work together to resolve them and optimize productive workflow.

---

**Figure 15. KPIs vs. Lifecycle Phase**

<table>
<thead>
<tr>
<th>Major KPIs Used to Measure Success vs. When Security Introduced in SDLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Builds failed due to security issues</td>
</tr>
<tr>
<td>Builds delayed due to security issues</td>
</tr>
<tr>
<td>Inception</td>
</tr>
<tr>
<td>8.6%</td>
</tr>
<tr>
<td>15.7%</td>
</tr>
<tr>
<td>17.7%</td>
</tr>
</tbody>
</table>

---
Conclusion

We know that traditional, gate-based software security methods can’t scale and can’t keep up with modern development. Manual audits, pen tests, compliance checklists, waterfall handoffs and other stage-gate security controls don’t fit into high-velocity, iterative delivery. Even techniques that worked for Agile product development teams can’t keep up with continuous deployment of code changes several times per day in DevOps.

We’ve also known for a while that there are not enough AppSec skills to go around. DevOps makes this even more urgent. Application security needs to be built into continuous delivery cycles, following a “trust-but-verify approach,” where DevOps teams are given self-service security tooling and safe-by-default templates and frameworks to follow. Security specialists must provide training and oversight without getting in the way of delivery, engaging delivery teams as early as possible and helping them to make smart and safe decisions on their own, stepping in and stopping the delivery pipeline only when needed to avoid serious security problems.

From a risk perspective, organizations must look in two different critical directions:

1. The prospective view: Organizations need to look forward to understand how to deal with security risks posed by cloud platforms, containers, and serverless architectures and continuous delivery pipelines.

2. The retrospective view: Organizations must also look backward, and they can’t afford to ignore legacy risks and technical and security debt that has already been taken on.

While achieving DevOps is still aspirational for most organizations, Secure DevOps is even more challenging. The challenges aren’t just technical problems—they are also, and maybe primarily, problems that management must take on and solve.

Management must recognize and understand the security risks in DevOps, and when and how security needs to be included. Management must implement—or at least permit—the organizational structures and incentives that facilitate, support and encourage collaboration, trust and accountability among engineering and operations and security, and provide DevOps teams with the training, tools, and time needed to “do it now” and “do it right.”

Without management commitment, DevOps will end up repeating the mistakes of the past, creating bigger problems for someone else to deal with in the future, and Secure DevOps will remain a fiction.

---

About the Authoring Team

**Jim Bird**, SANS analyst and co-author of SANS courses on Secure DevOps, is an active contributor to the Open Web Application Security Project (OWASP), and the author of books, blogs and articles on secure Agile software development and Secure DevOps. He is the co-founder and CTO of a major global institutional trading platform, where he is responsible for managing the company’s technology organization and information security program. Jim is an experienced software development professional and IT manager, having worked on high-integrity and high-reliability systems for stock exchanges and banks in more than 30 countries. He holds PMP, PMI-ACP, CSM, SCPM and ITIL certifications.

**Barbara Filkins**, a senior SANS analyst who holds the CISSP and SANS GSEC (Gold), GCIH (Gold), GSLC (Gold), GCCC (Gold), GCPM (Silver), GLEG (Gold) and GSNA (Silver) certifications, has done extensive work in system procurement, vendor selection and vendor negotiations as a systems engineering and infrastructure design consultant. She is deeply involved with HIPAA security issues in the health and human services industry, with clients ranging from federal agencies (Department of Defense and Department of Veterans Affairs) to municipalities and commercial businesses. Barbara focuses on issues related to automation—privacy, identity theft and exposure to fraud, as well as the legal aspects of enforcing information security in today’s mobile and cloud environments.

Sponsors

SANS would like to thank this survey’s sponsors:
### Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>SANS OnDemand</th>
<th>OnlineUS</th>
<th>Anytime</th>
<th>Self Paced</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>