O'REILLY®

10 Steps Every CISO Should Take to Secure Next-Gen Software

Cindy Blake

REPORT

10 Steps Every CISO Should Take to Secure Next-Gen Software

Cindy Blake

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

10 Steps Every CISO Should Take to Secure Next-Gen Software
by Cindy Blake

Printed in the United States of America.

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

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

Acquisitions Editor: Melissa Duffield
Development Editor: Alicia Young
Production Editor: Deborah Baker
Copyeditor: Octal Publishing, LLC

Proofreader: Christina Edwards
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

October 2019: First Edition

Revision History for the First Edition
2019-10-03: First Release

The views expressed in this work are those of the author, and do not represent the publisher’s views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

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

978-1-492-08288-0

[LSI]

Table of Contents

Foreword

My goal with this report is to ignite discussion and thought among a group of professionals who are hypervigilant about managing risk. Often managing risk takes a path of protecting the status quo. Yet, as the saying goes, no decision is a decision itself. Change will be necessary—and perhaps risky. Software development itself is changing rapidly and security programs must evolve if they are to be effective in this next generation of software. This report looks at the security implications involved in how software is changing: the code itself, the methodologies by which it is developed, and the infrastructure surrounding its use. I hope to provide some practical advice to help security leadership remain relevant—and maybe even become an innovative change agent—in this Agile world in which development speed is the holy grail.

10 Steps Every CISO Should Take to Secure Next-Gen Software

Context: DevOps Principles That CISOs Often Overlook

Sometimes, it can feel like developers and security are each speaking a different language. In some ways they are. Combine that with a massive sea change in which developers themselves are grappling with new and evolving development technologies, tools, and frameworks, and it's a bit of the Wild West! Without going into too much detail, let's highlight some critical areas to quickly get the security professional up to speed on the terms, what they mean, and why they are relevant to security.

Git What? Knowing the Lingo

Git, *GitHub*, *GitLab*: are they all the same thing? Should you care about the difference? It's all for developers anyway, right? Why does it matter to security? Let's begin by demystifying the terms around Git, understanding how it fundamentally changes the software development life cycle (SDLC), and looking at the security implications.

Git

Started in 2005, Git is a free and open source distributed version control system, used to help multiple software developers work on a given code base. The first distributed version control (BitKeeper) changed the workflow from the developer asking, "Can you add me to version control?" to making their own copy, changing the code, and then checking in their contribution. This change was revolutionary in that developers no longer needed to be invited to contribute to open source code repositories as well as proprietary code. (Fun fact: Linus Torvalds created Linux and also created Git.) A boom in open source code availability and code quality has resulted, as the number of contributors exploded and the code was more easily improved. At the same time, the ability to essentially farm out pieces of code among multiple developers has radically changed the way software is developed whether open source or proprietary.

GitLab and GitHub offer all of the distributed version control and source code management (SCM) functionality of Git as well as their own features with paid tiers of offerings. Although both started as code repositories, and share "Git" in their names, there are vast differences between the two.

GitHub

GitHub began in 2008 and was purchased by Microsoft in Fall 2018. GitHub has recently begun adding capabilities beyond the code repository but its security features have been mostly focused on securing the code itself, an area GitLab refers to as Compliance. Somewhat ironically, GitHub hosts the most open source projects (yet its code is proprietary), whereas GitLab estimates it hosts two-thirds of enterprises' proprietary code while offering an open source option of GitLab itself.

GitLab

GitLab began in 2011 and remains an independent company with a very transparent growth strategy. In 2018, GitLab was ranked the fourth fastest-growing private software company on the Inc. 5000 list, and now has more than 100,000 enterprises using GitLab.

GitLab is the only application for the entire SDLC, covering a larger scope of capabilities than other point-solution providers. At the same time, GitLab takes a different approach than traditional application security vendors. To truly "shift left" and empower developers to find and fix vulnerabilities early (DevSecOps), GitLab automates app security within its own fast-growing, already-popular developer tool, enabling a more seamless path-of-least-resistance for secure coding practices and a compliant, auditable process for securing the code itself.

Other Git repositories

BitBucket, another popular Git code repository, has 3.5 million build minutes per week, whereas GitLab has 100 million builds per month! Additional Git repositories can be found in Jason van Gumster's article, "6 Places to Host Your Git Repository", which was published to address the need of GitHub users (who are not fans of Microsoft) looking for alternatives.

Why You Need to Know Git

The use of Git and software code repositories has fundamentally changed the way enterprises do software development, for proprietary code and open source code alike. In addition, open source software has blossomed because some of the Git repositories host open source code for free. Most significantly, Git and code repositories have facilitated the DevOps methodology that is so impactful to software development, which puts enormous pressure on application security programs to evolve.

Additional important development security terms

In addition to the aforementioned terms, there are other security terms that will be important for you to know as you work to secure next-gen software.

Merge request and code commit. When using Git repositories, all of the code for a group, and projects within a group, is stored and managed by the repository. An individual developer "pulls" a copy of the code. This becomes a "branch" off of the master file. The developer is free to work on the branch, without affecting the master code. In the branch, the code is altered or more code is created. When a change is made, the code is committed to the branch upon which the repository can run automated scans for code quality, security, and more. Results of the scans are provided directly to the developer in a merge request pipeline report that shows the impact of their specific code changes—and only theirs.

With GitLab, a fully functioning review application is spun up with which the developer can also assess the usability of the change and run *dynamic application security testing* (DAST). These DAST results, together with *static application security testing* (SAST), dependency scans, container scans, and license compliance scans, provide a breadth of security insight right to the developer, before their code ever leaves their hands. This makes a very tight cause-and-effect feedback loop by which the developer can immediately assess the success or failure of their code changes. The developer can make additional tweaks, still on their own branch, until it works as intended and passes all required tests. When the code is deemed satisfactory, the changes are then "pushed," or checked back into the repository where they are combined with everyone else's changes. The repository identifies merge conflicts such as if two developers have been changing the same lines of code or created other conflicts like deleting libraries. When conflicts are resolved, the branch is successfully combined with the master file, delivering a functioning, tested code change capable of delivering incremental business value.

Agile. The Agile software development methodology, popularized by the *Manifesto for Agile Software Development* (2001), has as its key principle rapid iteration through an automated and instrumented SDLC in which processes can be improved using *Lean* methodologies. This approach enables development to be more responsive to rapidly changing business needs. It uses incremental, iterative work sequences commonly known as *sprints* (these are frequently two weeks long but can be of any length). *Scrum* teams are cross-functional groups that focus on delivering code required by a sprint. *Kanban* boards are frequently used to manage the work effort, where projects are pulled by development and completed as capacity permits, rather than work being pushed into a project based upon a more traditional work plan; for example, the *waterfall* methodology.

Agile is a precursor to DevOps. Although Agile is the methodology, DevOps takes the methodology further with cross-functional teams and automation. In 2013, *The Phoenix Project* (IT Revolution Press), by Gene Kim, Kevin Behr, and George Spafford, described how process discipline, helped by automation, along with measurements and feedback can change the efficiency of the modern software factory.

Continuous Integration and Continuous Deployment. The multithreaded development approach, enabled by both Git repositories and Agile methodologies, allows a much faster velocity, or cycle time, for software development and has led to *Continuous Integration* (CI) and *Continuous Deployment* (CD) to speed not only the development, but the delivery and use of the code and its resulting business value.

With CI/CD, these small, incremental code changes can be tested and deployed to test and to production. The process of code quality testing, a review environment, and assigning compute resources is all automated so that development is no longer at the mercy of an operations queue to set up the environment for them. With automation comes standardization and repeatability, along with closed-loop metrics that are helpful for process improvement, frequently found with Agile methodologies.

Secrets. Secrets are access controls that an application needs at build and runtime, including things like user access credentials, application program interface (API) keys, usernames, and passwords. As applications become containerized and abstracted from the hardware on which they run, managing secrets becomes a critical part of securing your applications.

Secrets can be managed in a decentralized manner adjacent to code (less secure) or through the use of a centralized, purpose-built solution such as Vault by HashiCorp or Amazon Web Services' AWS Secrets Manager.

According to a recent Stackshare survey (results originally posted here), "Vault, Docker Secrets, AWS Secrets Manager, Torus CLI, and Keywhiz are the most popular tools in the category 'Secrets Management.' 'Secure' is the primary reason developers pick Vault over its competitors, while 'Multi-Host aware' is the reason why Docker Secrets was chosen."

Some of the common mistakes in handling secrets (such as passwords) include putting them in the code itself, not rotating them, and not backing them up. In fact, one of the most frequently recurring mistakes related to application secrets storage is to simply store these secrets in a plain-text configuration file that is a part of the software project or in plain text as environment variables.

In his *Infosecurity* magazine writeup on a discussion at BSides San Francisco 2018 "Managing Secrets in Your Cloud Environment", Michael Hill notes that the participants included the following as good practices for secrets management:

Identity
: Using strong identities and *least privilege*

Auditing
Verifying the use of individual secrets

Encryption
Always encrypting before writing to disk

Rotation
Regularly changing a secret in case of compromise

Isolation
Separating where secrets are used versus managed

Traditional application security tools check for things like storing passwords in the code, but they are useless for actually managing the secrets themselves via rotation, isolation, and encryption. Therefore, most enterprises are turning to software solutions to help them manage their secrets.

Tools like HashiCorp's Vault help teams to avoid *secrets sprawl* by keeping their management in one location. Tools like this also provide an audit log and enforce access-control policies on a least-privilege basis. They can also rotate credentials or generate temporary credentials that immediately expire upon task completion, the credentials are revoked.

The Next-Generation Software Evolution

According to this *Medium* article by Sid Sijbrandij, the next generation of software development is upon us. It involves DevOps, punctuated by extensive automation of the development and deployment process. It also involves digital transformations with software-defined infrastructure (or software-defined-everything), cloud native applications, and newer technologies such as containers, that come with amped-up use of open source third-party code and microservices.

If you are uncertain about how much of DevOps is hype and how much is real, you need only look at recent M&A activities in the market. Investors and venture capitalists see it as a game changer, offering up big money to bet on transformational solutions. Recent acquisitions reinforce this point:

- TwistLock acquired by Palo Alto Networks for $410 million

- The 2018 GitHub acquisition by Microsoft for $6.5B, undertaken as an onramp to Azure and an attempt to win back developers
- CloudBees acquired Electric Cloud to advance its CD

In addition to M&A, venture capitalists have plunged additional investment in DevOps companies. XebiaLabs announced a $100 million funding round and CircleCI $31 million, whereas GitLab raised $268 million in Series E funding and achieved a $2.7 billion valuation in 2019. DevOps use affects security programs heavily as it introduces new attack surfaces, brings new challenges for collaboration, and requires new skill sets.

Now that you have a better understanding of what all the terminology around this area means, let's look at three key shifts involved in next-generation software as well as the security impact of each.

Three Critical Shifts of Next-Generation Software and How They Affect Security

Software code repositories and open source technologies are lowering the barrier of innovation at an accelerating pace, driving businesses' time to market to be their competitive advantage. Successful companies are modernizing the business through software and IT as both a cultural and technological shift. Enterprises need solutions that empower their teams to adopt a flexible, continuous, and iterative culture toward improvement. Less successful ones struggle to create processes inherently limited by tool choices made in advance of thinking through overall IT policy and workflow changes.

As enterprises try to increase the speed with which they create business value via new and revised software, there are multiple variables evolving, each with its own security impact, including:

- How software is composed and executed
- How it is delivered and managed
- How it complies with regulatory requirements

Now, let's walk through these shifts and assess what you need to know about each.

Shift 1: How Software Is Composed and Executed

Software was once created by a solo programmer. That person would be assigned a new software program or an update, and they were responsible for crafting the code, ensuring that it would properly execute and taking it through a change management process to move it to production. With the advent of software repository managers (aka "repos"), one program can be divided up and assigned to multiple software engineers. The repo ensures version control so that one person's changes do not clobber another person's updates. This version control allows not only more people to concurrently work on a given project, but with publicly accessible code libraries, it has simplified the use of open source code written by third parties.

Software becomes a multithreaded assembly of building blocks

Next-generation software is increasingly crafted from third-party software modules that are put together like building blocks to quickly meet a business requirement. Public software repositories typically house this open source code making it readily accessible to both developers and hackers alike. Growth in the use of open source and of code repositories goes hand in hand; they each fuel the other.

The 2019 *TechCrunch* article "How Open Source Software Took Over the World" explains that open source is no longer an anomaly but is now rapidly becoming mainstream. Demonstrating this growth, Red Hat is being acquired by IBM for $32 billion, MongoDB is now worth north of $4 billion, and Elastic's IPO now values the company at $6 billion.

Much of this third-party code is open sourced wherein everyone has access to it. In a democratized world of open source software, developers basically vote with their feet. Only the good software survives. The appeal for the developer is that it's been well tested by many users and it works. The expectation is often that security bugs have been found and worked out, as well, although that is often not the case. Even when bugs are widely reported, if your enterprise is using an older version that has not been updated, you are vulnerable. As described in this Brighttalk webinar, of those who have suffered a breach from an external attack, more than 30% were due to exploited software vulnerabilities. Such attacks are largely preventable with timely patch installation. An additional concern for security is that hackers, too, have access to the code and can easily plan exploits. Open source code is an attractive target due to the sheer "blast radius" of so many users.

According to the 2019 GitLab Global Developer Report, a decisive 95% of developers use Git for source-code control. Needless to say, Git repositories are here to stay. And security professionals need to understand which repositories their developers are using (most likely GitHub, GitLab, or BitBucket, as the current market leaders) in order to take appropriate measures to secure them.

Nearly 70% of the 4,000 developers responding to the GitLab Global Developer Report, said they are expected to write secure code, but it's clear from the comments that the mechanisms to make that happen remain elusive in most organizations. Comments included the following:

- "It's a mess, no standardization, most of my work has never had a security scan."
- "We're starting to care about it, just now."
- "We don't have clear guidelines about security, so the different services present different levels of security."

At the same time, 45% of developers said they receive and are able to address security feedback during the development process. From the comments, it's clearly not enough.

Security Challenge

So how does a security person get their arms around what third-party code is used, what vulnerabilities it carries, and the state of patches? You must worry not only about new code, but what lurks within production already, along with what third parties have created beyond your enterprise. *Technical debt* accumulates every day as the developers pull in more and more open source code without testing it. Do you have a way to identify it? How do you ensure that it is secure if you don't even know what your enterprise has used?

Software execution becomes dynamic

Enterprises have learned valuable lessons from legacy software. When software is tied closely to hardware, upgrades of hardware and software become intertwined and interdependent. Companies become beholden to sunsetting hardware, which forces sometimes unwanted changes to the software. Similarly, software upgrades can be held captive to hardware limitations.

Cloud computing avoids hardware vendor lock-in. In fact, top drivers of cloud service adoption include: low-cost elastic scalability, flexibility of choice, and self-service capabilities. Furthermore, your code can now run literally anywhere. It could be in one cloud provider today and another one tomorrow, facilitated by containers and orchestrators. This abstraction of software from hardware facilitates software development while at the same time reduces costs of commercial software licenses. Technologies enabling this shift are exploding, and CISOs should take note.

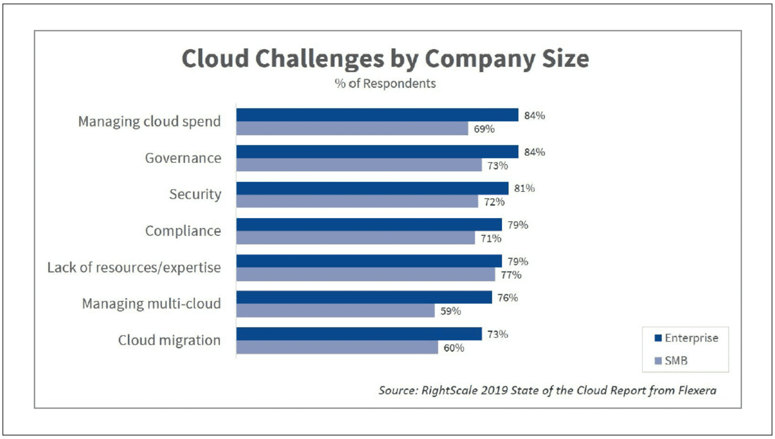
Cloud computing: The start of shared security accountability. It is important that the CISO understands that the movement to cloud computing is increasingly much more than a simple shift in compute resources from onsite to offsite. That might have been the primary driver 10 years ago, but today's business drivers are much more focused on flexibility and developer productivity to allow an enterprise to capitalize on faster time to market. The cloud is only the beginning. Cloud is the foundation that enables next-gen software and the benefits it provides. The following studies provide evidence of increasing reliance on cloud computing:

- According to a survey by LogicMonitor, 41% of enterprise workloads will be run on public cloud platforms (AWS, the Google Cloud Platform, IBM Cloud, Microsoft Azure, and others) by 2020. Suggesting an even more aggressive pace, the 2018 IDG Cloud Insights survey claimed that 73% of organizations are using cloud computing.
- RightScale's 2017 State of the Cloud report showed Azure adoption grew from 20% to 34% of respondents, whereas AWS stayed flat at 57% of respondents. Google also grew from 10% to 15% to maintain third position. Azure also reduced the AWS lead among enterprises; Azure increased adoption significantly from 26% to 43%, whereas AWS adoption in this group increased slightly from 56% to 59%.
- The subsequent 2019 report showed Azure adoption grew even further to 52%, more or less splitting the market with AWS.

In addition to adoption, a Forrester study commissioned by GitLab, "Manage Your Toolchain Before It Manages You", found that, "This flexibility is especially important as 77% of IT professionals agree that their organizations are moving to the cloud and they want to avoid cloud lock-in. With that in mind, leveraging a tool that is cloud-agnostic will provide the highest level of cloud independence and leveraging one that has that capability, out of the box, will accelerate a team's ability toward achieving a multicloud strategy." This hybrid cloud offers organizations many benefits but requires cloud management solutions to maintain visibility and overcome complexity.

If you agree that securing applications in the cloud presents a shared accountability with the cloud provider and does not inherently make you more secure, feel free to jump to "Application execution on multicloud" on page 14. If you are one of those who I've heard as recently as the 2019 RSA Conference telling others that you are inherently more secure because of the cloud, please read on.

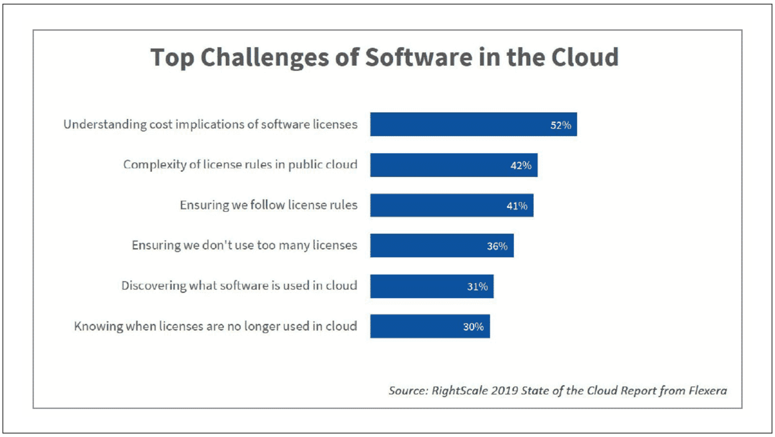
Even with the surge in cloud adoption, security remains somewhat of an afterthought with lingering concerns. In Flexera's 2019 State of the Cloud Survey, security came in a close third among concerns about cloud challenges overall (Figure 1).



Source: RightScale 2019 State of the Cloud Report from Flexera

Figure 1. Four out of five enterprises are concerned about the security of cloud computing.

At the same time, nearly one-third of the survey respondents struggle to know what software is run in the cloud (Figure 2). How can you secure what you do not know you have?



Top Challenges of Software in the Cloud

Source: RightScale 2019 State of the Cloud Report from Flexera

Figure 2. Nearly one-third struggle to know what software their enterprise is using in the cloud.

Most cloud users understand that security is a shared accountability —a paradigm illustrated in Figure 3. The cloud provider takes care of securing the underlying hardware and basic network that it uses to provide cloud services, but the user must do plenty. Unless you are using their serverless offering, cloud providers generally do not provide network segmentation and perimeter security for your guest virtual machines (VMs) out of the gate. Your engineer usually configures that. Your cloud provider might implement virtualization and root access, yet engineers at GitLab report that they are involved in every cell on the chart in Figure 3 in some way. So take this chart with a grain of salt. Note that the purple boxes are primarily the responsibility of the cloud provider and the white boxes are primarily that of the user, but the responsibilities are shared.

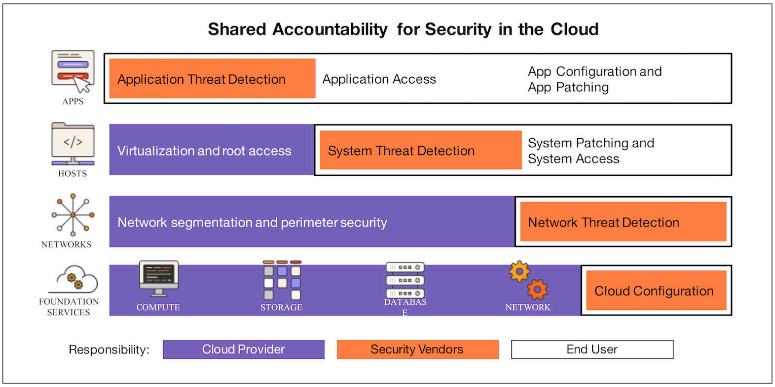

Shared Accountability for Security in the Cloud
APPS
Application Threat Detection
Application Access
App Configuration and App Patching
HOSTS
Virtualization and root access
System Threat Detection
System Patching and System Access
NETWORKS
Network segmentation and perimeter security
Network Threat Detection
FOUNDATION SERVICES
COMPUTE
STORAGE
DATABAS E
NETWORK
Cloud Configuration
Responsibility:
Cloud Provider
Security Vendors
End User

Figure 3. Protecting applications in the cloud requires shared accountability between the enterprise and the cloud service provider. As cloud services add more security capabilities, the burden for configuration settings will still fall to the user.

As if these aren't already enough considerations for the cloud, you will also need to be aware of the challenges presented when multiple cloud providers become involved, which is increasingly common. Let's turn to that topic now.

Multicloud: Pushing shared accountability even further. With the flexibility afforded by containers and orchestrators, enterprises can use multiple cloud providers. CI/CD simplifies the actual deployment of code to a variety of environments.

Findings substantiated this trend in 2017 with RightScale showing that 85% of enterprises had a multicloud strategy, up from 82% in 2016. Cloud users were already running applications in an average of 1.8 public clouds and 2.3 private clouds. The most cited challenge among mature cloud users is managing costs (24%), whereas among cloud beginners, it is security (32%). The report was updated in 2019 by Flexera (which acquired RightScale), revealing that 84% of the respondents are using more than four cloud platforms.

The Cloudability State of Cloud 2018 report, based on actual spend data, collaborates this evidence, reporting that 84% companies have multicloud plans.

Application execution on multicloud. There are two primary approaches to creating the layer of abstraction that enables a multicloud strategy: cloud native computing and serverless. Serverless is philosophically one step beyond cloud native, so let's begin our discussion with cloud native. It's important to understand how these approaches are achieved, the components involved, and the security implications of each.

The first thing you need to know is that containers are the key enabler to next-generation software and the multicloud evolution, the place where you begin this journey. They offer portability through a flexible approach to running, packaging, and deploying code. We discuss them as an attack surface in the next section, but, briefly, here's what you need to know up front.

Containers are used to package an application and its dependent resources so that the application can run reliably on any Linux server. This enables enterprises to achieve greater application flexibility and portability to run anywhere, including on-premises, in a public cloud, or a private cloud. This not only avoids vendor lock-in for compute platform, but it minimizes the risk of becoming locked in to legacy hardware environments. Remember when music and video was locked in to their respective hardware platforms—how if you bought a phonograph, VHS/Beta, or cassette tape, you could only listen to it or view it on those devices? As devices improved, your previous music investments became outdated. Software has been similarly held captive by the hardware environment for which it was developed. With containers, software can enjoy portability similar to that of digitizing your music so that you can enjoy it regardless of hardware. With digitization, you might be locked into a subscription service, but you are not beholden to a hardware technology. Serverless takes this model a step further to where you don't even need to worry whether you have enough storage capacity to store the songs, you simply subscribe and access them. When you ask Alexa to play a song for you, you need only concern yourself with whether you have a subscription to license its use, not where or how it's stored.

Now that you have a primer on containers, let's look at some of the security vulnerabilities for cloud native applications that containers enable.

Cloud native applications: New attack surfaces

According to the Cloud Native Computing Foundation (CNCF), cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.

The CNCF has spawned several technologies in support of its mission as an "open source software foundation dedicated to making cloud-native computing universal and sustainable." Together the technologies supported by the CNCF help enterprises avoid vendor lock-in and gain flexibility to run their applications on any cloud environment, whether private or public.

CNCF is hosted by the Linux Foundation, a nonprofit organization whose mission is "to train the next generation of open source professionals." Some of the open source technology projects created by the CNCF include Kubernetes (orchestration), Prometheus (monitoring), FluentD (logging), and more. To clear up a popular point of confusion, note that like many other free and open source solutions, these are hosted on repositories housed in GitHub. They are not owned by GitHub (although the CNCF website could cause one to believe that), and, in fact, Kubernetes, Prometheus, and others are used extensively by GitLab, AWS, and others.

Although there is a plethora of tools available from the CNCF and elsewhere, the three most significant building blocks, regardless of provider, that characterize "cloud native" applications are containers, orchestrators, and microservices. We summarize them here with a much deeper dive to follow after a brief primer. Each building block brings with it new risks with a new attack surface for potential exploit.

Table 1. Containers, orchestrators, and microservices and service meshes

	Purpose	Popular vendors	Risks
Container	Containers make the applications portable and can reduce costs of commercial software licenses via a shared operating system.	Docker, Linux Containers	Vulnerabilities within images, misconfigurations

	Purpose	Popular vendors	Risks
Orchestrator	These direct how and where containers run.	Kubernetes, Docker Enterprise Edition, Google Kubernetes Engine, Red Hat OpenShift Container Platform, Rancher	Exploit of these “keys to the kingdom,” misconfigurations.
Microservices and Service Meshes	*Microservices* an architecture in which applications are composed of smaller parts (microservices) to make them easier to maintain and scale based on load. A *service mesh* is a communication layer between services that handles east-west traffic between microservices.	Linkerd and Istio	Exploit of permissions between containers, clusters, and apps.

The growing complexity and configurability of these components, especially when taken together, represent a substantial new security risk that must be incorporated into an enterprise security program. Common security risks stem from misconfiguration, vulnerable container images, and orchestrator exploits.

These components add to the complexity of cloud computing and change the landscape of application security, as shown in Figure 4. Now, let’s take a look at each in greater depth.

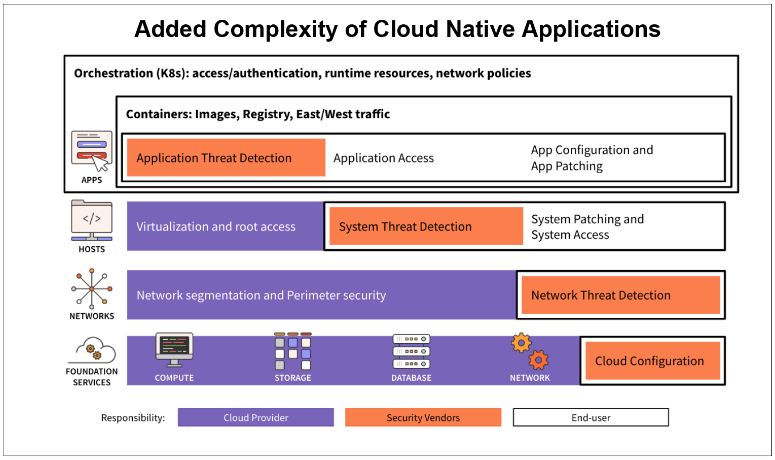

Added Complexity of Cloud Native Applications
Orchestration (K8s): access/authentication, runtime resources, network policies
Containers: Images, Registry, East/West traffic
APPS
Application Threat Detection
Application Access
App Configuration and App Patching
HOSTS
Virtualization and root access
System Threat Detection
System Patching and System Access
NETWORKS
Network segmentation and Perimeter security
Network Threat Detection
FOUNDATION SERVICES
COMPUTE
STORAGE
DATABASE
NETWORK
Cloud Configuration
Responsibility:
Cloud Provider
Security Vendors
End-user

Figure 4. The added complexity of cloud native applications presents new risk from misconfiguration, vulnerable container images, and orchestrator exploits. These new burdens rest primarily on the enterprise today in a fragmented market of emerging vendor solutions.

Containers. Docker, which started in 2013, is probably the most popular container software, but as described in this *Container Journal* article, there are others. Datadog, whose software monitors infrastructure and applications, published a useful research report quantifying Docker adoption and usage among its customers. Here are some of their findings, further punctuating Docker's popularity and growth:

- Nearly one quarter of companies have adopted Docker, whereas approximately 21% of all hosts now run Docker, as of April 2018.
- Docker usage rates increase with infrastructure size. Among the organizations with at least 1,000 hosts, 47% have adopted Docker, as compared to only 19% of organizations with fewer than 100 hosts. An additional 30% of organizations with 1,000 or more hosts are currently dabbling with Docker.
- 50% of Docker environments use an orchestrator (discussed next) such as Kubernetes or Mesos, or a hosted orchestration platform from AWS, Azure, or the Google Cloud Platform.
- At companies that adopt Docker, containers have an average lifespan of about two days.

Similarly, Flexera's 2019 State of the Cloud Survey found that 39% of respondents are expanding the use of containers as one of their top cloud initiatives.

Linux containers are implementations of OS-level virtualization in which the Linux kernel's functionality is used for resource isolation for CPU and memory, and separate namespaces isolate the application's view of the OS. Unlike a VM, containers are run by a single OS kernel, and don't require a separate OS. Using containers can therefore reduce the cost of packaged software where licenses are tied to an OS. Whereas containers can share a host's OS, VMs require their own, indicating the compelling reason why VMs, once the standard for distributed computing, are being replaced by containers.

Securing Containers. With containers, the image and the registry introduce new attack surfaces. The traffic between the applications in a container does not cross perimeter network security but should be monitored for malicious traffic between apps and their images. For instance, an HR system probably shouldn't communicate with a point-of-sale app. Though orchestrators contain tools to help with security (e.g., by setting policies for how long processes should run or the maximum resources they should consume). But there are still many ways in which attacks can succeed. After breaking down each component, an attack scenario will help the CISO envision how attack surfaces combine to represent new risks.

There are two schools of thought on container security. Some believe containers to be inherently more secure because they compartmentalize applications, whereas others believe them less secure due to potential access to the host kernel. Let's examine these ideas further.

- Containers are isolated from one another and bundle their own software, libraries, and configuration files. A program running on an ordinary OS can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs running within a container can see only the container's contents and devices assigned to the container. This isolation can provide some element of risk reduction by limiting the access of a compromised application. Although this might prevent some attacks from traversing app to app, they can, however, communicate with one another in a way prescribed by configuration settings.

Those configuration settings are the weak link and offer additional risk.

- Containers are created from *images* that specify their precise contents including applications and their requisite resources. Similar to images used to deploy the same PC configuration to multiple desktop or laptop users, images are used to store and deploy applications. One vulnerable image can have a significant blast radius via reuse. Images are often created by combining and modifying standard images downloaded from public repositories, which carry risks of their own. This assembly of ready-made parts is another factor contributing to the growth in open source software.

A Docker registry is a repository for Docker images. Docker clients connect to registries to download ("pull") images for use or upload ("push") images that they have built. Registries can be public or private. As with any software repository, container registries are subject to misconfiguration and lax policies.

An incredibly short life span (such as the aforementioned two days) requires an automated approach to securing containers. Furthermore, containers can run on a single server or across multiple VMs. Because their environment spins up and down so quickly, perimeter security is useless, yet their super-dynamic nature can at least frustrate hacking attempts. This Datadog article summarizes the challenge: "Containers' short lifetimes and increased density have significant implications for infrastructure monitoring. They represent an order-of-magnitude increase in the number of components that must be managed and monitored, and the rapid churn of containers makes it all but impossible to track and monitor containers manually."

In this *CIO* article, writer Paul Rubens offers: "Many people believe that containers are less secure than virtual machines because if there's a vulnerability in the container host kernel, it could provide a way into the containers that are sharing it. That's also true with a hypervisor, but because a hypervisor provides far less functionality than a Linux kernel (which typically implements filesystems, networking, application process controls, and so on), a hypervisor presents a much smaller attack surface."

Noting further complications, Rubens points out that "virtualization and containers are also coming to be seen as complementary technologies rather than competing ones. That's because containers can be run in lightweight virtual machines to increase isolation and therefore security, and because hardware virtualization makes it easier to manage the hardware infrastructure (networks, servers, and storage) that are needed to support containers."

Security Considerations of Containers

Given all of these considerations for securing containers, this is what CISOs need to think about:

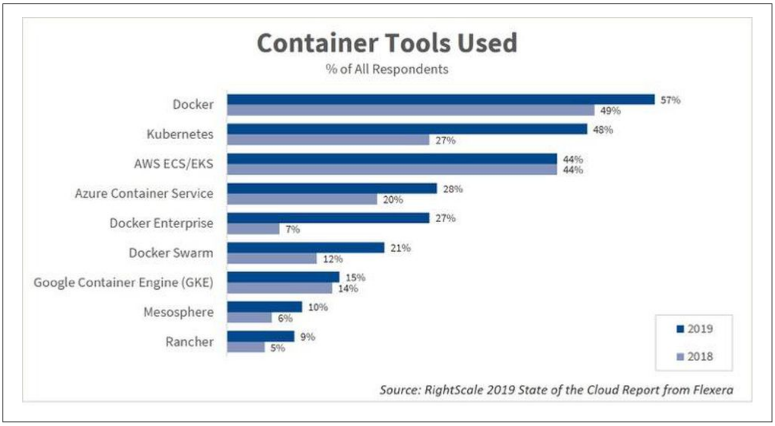
- Use container authentication to ensure authorized access.
- Use signed containers to prevent untrusted containers from being deployed.
- Scan container images for vulnerabilities.
- Monitor and protect runtime behavior within and among containers. Look for tools that profile a container's expected behavior and whitelist processes and networking activities (such as source and destination IP addresses and ports) to alert you of malicious or unexpected behavior.

Orchestrators. Just as containers have grown in popularity, so have orchestrators. Orchestrators help you scale the use of containers and automate the process of managing or scheduling the work of individual containers for applications based on a microservices architecture. Popular container orchestration platforms are based on open source versions like Kubernetes, Docker Swarm, or a commercial version from Red Hat called OpenShift.

The 2019 report by Flexera, The State of the Cloud Survey, summarized the sometimes explosive growth of container-enabling technologies, as shown in Figure 5. Docker and Kubernetes lead the way and their use is becoming mainstream:

- The use of Docker containers grew from 49% in 2018 to 57% in 2019 after growing from 35% in 2017 when they overtook Chef and Puppet, when each had 28% of the market. An even higher percentage of enterprises (versus small- to medium-sized business) use Docker (40%) with 30% more planning to do so, according to the 2019 report.

- Similarly, Kubernetes, a container orchestration tool that uses Docker, achieved even faster growth, increasing from 27% to 48% adoption in 2019 after growing from 7% in 2016 to 14% in 2017.
- At the same time, Azure Container Service adoption reached 28% in 2019, up from 20% in 2018.



Source: RightScale 2019 State of the Cloud Report from Flexera

Figure 5. Results from Flexera's 2019 survey of container tools. Docker, Kubernetes, and AWS lead the way, with Kubernetes having experienced explosive growth since 2018.

Kubernetes is arguably the most popular open source orchestrator. First developed by Google and nurtured by the CNCF, it's described by the CNCF as "help[ing] users build, scale and manage modern applications and their dynamic lifecycles...The cluster scheduler capability lets developers build cloud native applications, while focusing on code rather than ops. Kubernetes future-proofs application development and infrastructure management on-premises or in the cloud, without vendor or cloud-provider lock-in." Kubernetes, Docker, and GitLab automatically test software changes against user requirements and deploy to staging for feature branches and to the production master.

Datadog's research report on Docker adoption also addresses the use of orchestrators. The key points in the report are as follows:

- The rise of orchestration correlates with a greater number of containers per host. The typical organization that uses a container orchestrator runs 11.5 containers per host, versus 6.5

containers per host in unorchestrated settings. Orchestrators can place workload containers on any node with sufficient resources, leading to more efficient use of host resources and increased container density.

- The rapid adoption of orchestrators associated with container growth appears to be driving containers toward even shorter lifetimes, as the automated starting and stopping of containers leads to a higher churn rate. In organizations running an orchestrator, the typical lifetime of a container is about 12 hours. At organizations without orchestration, the average container lives for six days.

Container orchestration automates the following tasks at scale:

- Configuring and scheduling of containers
- Provisioning and deployments of containers
- Availability of containers
- Identifying which containers a given application runs within
- Allocating resources between containers; connecting containers to storage
- Load balancing, traffic routing, and service discovery of containers
- Scheduling deployment of containers into clusters and determines the best host for the container; after a host is chosen, the orchestration tool manages the life cycle of the container based on predetermined specifications
- Monitoring of containers
- Securing the interactions between containers.

NOTE All of these settings' configurations are vulnerable to human error or misuse.

Securing Orchestrators. The most recent CNCF survey asked users about the general challenges they face in using containers. The New Stack analysis of the CNCF survey Fall 2017 data, published in

"Kubernetes Deployment & Security Patterns", shows security as a top challenge, though this varied by size of enterprise (Figure 6). The report shows that 55% of organizations with 1,000 or more employees said security is a challenge, whereas only 39% of organizations with fewer than 100 employees said the same.

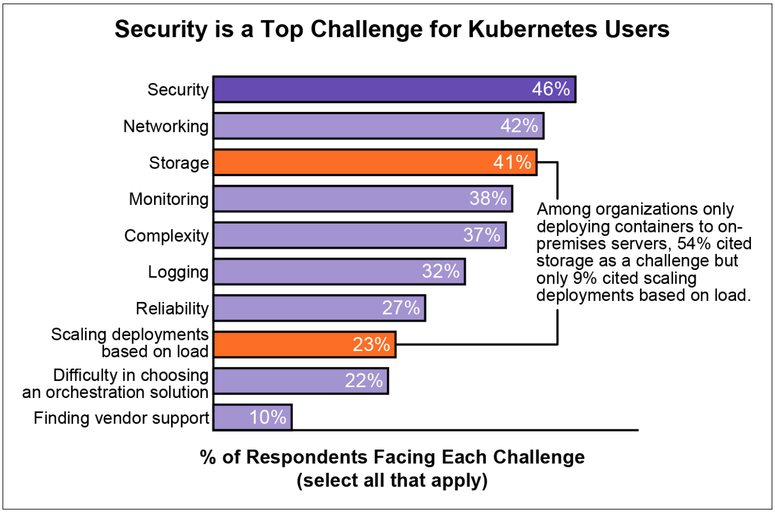


Figure 6. Security is a top challenge cited by Kubernetes users, with its importance differing by size of enterprise.

Security Considerations of Orchestrators

Orchestration tools offer substantial automation. The automation requires configuration and it is the configuration of all these settings that are vulnerable to human error or misuse. In "Knowing your risk: An attack scenario" on page 26 we look at an attack scenario that describes how these vulnerabilities enable exploit.

Microservices and service meshes. In modern software, microservices are an architectural design for building a distributed application in which each function of the application operates as an independent service. An application is split into modules such as the database, the application frontend, and so on. A code repository (such as GitLab or GitHub) is what enables version control, organization, and collaboration among these various teams. It also allows concurrent development by multiple teams in order to accelerate development.

Microservices are not new—just an old concept taken to a new extreme. In an attempt to break down monolithic applications into more manageable and maintainable apps, as far back as the late ’80s and early ’90s, when modularity was in vogue, subroutines of software that each had one purpose were created. The academic terms used for these were *coupling* and *cohesion*. *Coupling* refers to the indication of the relationships *between* modules, and *cohesion* to the indication of the relationships *within* a module, the degree to which a component or module focuses on a single thing. Good software design has *high cohesion* and *low coupling*. You wanted to *decouple* subroutines (services) that could stand on their own. The overall program reflected the logic flow that called subroutines (or microservices in today’s language; what’s old is now new, it just goes by different terminology).

Taking the old modularity a step further, today’s microservices approach allows for each service to scale or update without disrupting other services in the application. Applications designed this way are also easier to manage because each module is relatively simple, with a more singular purpose and less interdependency. As mentioned, in traditional terms microservices have high cohesion and low coupling. Changes can be made to modules without having to rebuild the entire application.

Modern software development has taken this to a new extreme. By taking advantage of the building blocks of third-party open source code, development can be accelerated by reusing code created by others. These subroutines/microservices tend to be used often, so vulnerabilities discovered become well known—by the good guys *and* the bad guys.

Microservices can be managed by a centrally orchestrated, universal service mesh composed of a fabric of services providing dynamic load balancing, service discovery, security, microsegmentation, and analytics for containerized applications running in Kubernetes environments. Individual microservices can be initiated only when they are needed and are available almost immediately.

The Codeship blog offers a good explanation of a service mesh, describing it as a communication layer between microservices (handling what is called East-West traffic). It plays a network communications–like role within the Kubernetes pods, providing load balancing and service discovery among microservices.

Traditional network communication relayed messages from client to server and back. You could easily trace the route that messages take and debug latency issues and errors. But when an application is made up of loosely coupled microservices where each microservice is made up of multiple containers (or pods in the case of Kubernetes), every message now touches multiple services, each of which is dynamic. Containers are created and destroyed automatically as the system changes and as deployments are made. A service mesh helps manage this communication layer.

Security Considerations of Microservices

Microservices inherently tend to use more open source code. It is critical that you have tools and processes for scanning open source dependencies along with container images and custom code. Furthermore, as apps are broken down into smaller, more singular functions, traditional application security that charges by the application could become even more costly and those costs less predictable. Lastly, the network security investments of the past will likely be replaced by investments at the service mesh layer in which the "network" becomes APIs and function calls between the various microservices. Even though perimeter defenses are still needed, they become a less-relevant means of protecting applications that are executed on more dynamic platforms.

Summary: Security challenges of cloud native applications

Each of the layers that we just examined has its own ability to interact with the applications, whether directly or indirectly. Each represents a path for the adversary. And, today, each of the layers has an array of user-defined settings intended to help you translate your security policies into settings used by each element/layer. This manual configuration is fraught with opportunities for user error and misconfiguration that opens the enterprise to potential exploit. One recent example is the Capital One breach. In addition, even the default settings are not always very secure:

- For instance, AWS, Google, and Azure all have settings for cloud access—what can be run, for how long, under what subscription, and so on.
- Containers such as Docker also have settings that determine parameters for how much resources they can consume, what apps they hold, and more. By isolating apps within their own containers and having those containers spin up/down frequently, you can indeed make traditional network-based attacks more difficult. But that's not where the puck is going. Attackers can target the containers and exploit user configuration errors and misjudgements.
- Orchestrators have their own settings vulnerable to misconfiguration and misuse. Tesla suffered a cryptocurrency mining malware infection enabled by a misconfiguration in the Kubernetes console that left the console without password protection. Attackers were able to access one of the pods that included access credentials for Tesla's larger AWS environment.

Let's look at an attack scenario to better understand the potential vulnerabilities of a cloud native application. Then, we look at some strategies for securing it.

Knowing your risk: An attack scenario

During a roundtable discussion on cloud security at RSA Conference 2019, Jay Beale from Inguardians demonstrated a hack on Kubernetes. In Table 2, I've summarized and somewhat condensed the steps he took here, along with his advice on preventative measures that would have stopped this attack.

Table 2. Example attack path on a cloud native app

Path of attack	Prevention
Vulnerable plug-in allows remote code execution	Dependency scanning
Lateral move to another pod	Egress policy to whitelist pod interaction (not default settings)
Other pod could talk to API server	Role-based access control with minimal permissions
From cluster, asks AWS who am I (using curl), which provides AWS credentials	Minimal permission—pod should not need metadata access
AWS S3 using default token can read (not write) every container; secrets are not exposed	Minimal permissions; ecrets encryption

All of the vulnerabilities he demonstrates result from configuration mistakes, and many are simply caused by using default settings. The default that every pod can reach every other pod no matter where they are physically brings a very real threat. As you can see from the path of attack in Table 2, entry is made from vulnerable third-party code. The attacker moves laterally until they find a pod with access to the API server, and then from the API server accesses the AWS credentials, simple storage service (S3) buckets, and eventually the secrets used for access control.

Beale did point out that the Center for Internet Security (CIS) benchmarks are very helpful. They cover a broad range of topics, so it can be difficult to find relevant information. You can find Kubernetes under Virtualization Platforms and Cloud. In Table 3, I've summarized some of the benchmark recommendations relevant to cloud native.

Table 3. Abstract from CIS benchmarks for securing Kubernetes

Threat	Cause	Prevention
External attacks	API server, kubelet, or etcd compromise	• Only expose necessary Kubernetes services • Authentication • Configure security policies for exposed services
Compromised containers	A container escalates privilege to control another container or the cluster	• Kubernetes isolation via namespaces or network segmentation
Compromised credentials	A malicious user gains access	• Enforce least-privilege access, role-based access control, and other access controls
Misuse of legitimate privileges	Misconfiguration	• Harden system components • Design and implement proper authorization • Kubernetes plug-ins that allow sophisticated authorization logic

The CIS Benchmarks claims to be "the only consensus-based, best-practice security configuration guides, both developed and accepted by government, business, industry, and academia." While Beale says the benchmarks are useful, he also suggests that Security Configuration Guides fall short.

Beale does penetration testing and will tell you that traditional network-based penetration testers will miss most of these vulnerabilities. Some of this comes down to hygiene, like dependency scanning and applying the latest release patches. It also requires a Zero-Trust approach—one that extends beyond traditional network security and endpoints into the application infrastructure.

This great ebook offered by the New Stack, *Kubernetes Deployment and Security Patterns*, offers similar advice, whereas Kubernetes has put together a comprehensive security guide. Key themes here are least-privilege access, utilizing the strength of Kubernetes, and careful configuration. It's this configuration that's troublesome. One additional resource is "9 Kubernetes Security Best Practices Everyone Must Follow", hosted on the CNCF blog and written by Connor Gilbert, product manager at StackRox.

In addition to the challenges of securing individual components, complexity is building. Cloud providers offer new settings (to learn) and are starting to offer more and more security capabilities themselves. Kubernetes is launching new releases frequently, and keeping up with the changes alone can be daunting. Time and experience have proven that people are generally not good at staying current with their patches, yet patches might be easier to digest than a regular diet of new configuration settings made available each month.

You need guidelines but you also need automation to abstract the complexity. Watch the industry for capabilities to translate policies into settings in each of these elements—and across cloud providers. Vendors will need to fill in the gap until industry standards are developed.

Beyond configuration vulnerabilities, infrastructure is created and defined just-in-time by developers. It can spin up and spin down. And, it can live on leaving vulnerabilities like open doors. It's important that vulnerable code is identified and remediated.

The Docker security report "Shifting Docker Security Left" captures data based on a survey of open source developers, data from public application registries, publicly available Docker images, GitHub repositories, and Snyk's own vulnerability database. Key findings reveal the following:

- 50% of developers don't scan their Docker images for vulnerabilities at all.

- The top 10 most popular Docker images contain at least 30 vulnerabilities each.
- 45% of developers never discover new vulnerabilities disclosed in their production containers.
- 44% of Docker images have known vulnerabilities for which newer and more secure base images are available.

It's clear that cloud native applications, and the new attack surfaces that come with them, represent a new frontier for application security. Containers must be scanned, all components checked carefully for misconfigurations, and vulnerabilities must not be allowed to linger, waiting for the lucky attacker to happen upon them.

Now that you better understand the considerations for cloud native applications, we can turn to serverless applications, which take the concepts of cloud native a step further, pushing the security implications along with it.

Serverless applications: When network security is irrelevant

Whereas cloud native is an operations paradigm, serverless is for practical purposes, about not doing any operations. With serverless computing, the cloud provider runs the server and dynamically manages the allocation of machine resources. Pricing is more like utility computing and is based on the actual amount of resources consumed by an application rather than on prepurchased units of capacity. Applications are designed to take advantage of this and are made up of functions that are instantiated only when a user accesses that part of the code and persist only while in use or for a maximum interval. This affords radically optimized use of compute resources. In addition, serverless simplifies code deployment for both the developer and operations by abstracting away deployment and operations tasks like capacity planning, scaling, and maintenance. A serverless application accomplishes this by running the applications in stateless compute containers (Function-as-a-Service [FaaS]). The compute function is initiated and eliminated dynamically based upon event triggers. Pricing is based upon function execution, not prepurchased compute capacity.

The serverless framework, first introduced in October 2015, is a free and open source web framework originally developed for building applications exclusively on AWS Lambda, a serverless computing platform offered by AWS. Applications developed as serverless can be deployed to any FaaS providers such as Microsoft Azure with Azure Functions, IBM Bluemix with IBM Cloud Functions (based on Apache OpenWhisk), Google Cloud Platform using Google Cloud Functions, or Kubeless, the Kubernetes-native serverless framework. A serverless application can be a couple of Lambda functions to accomplish a specific task (a microservice); an entire backend composed of hundreds of Lambda functions; or a combination of services managed by cloud providers (such as Amazon RDS for DB or SQS for queues), with app functions as "glue code." Serverless can also be done in a private cloud environment or even on-premises, building upon the Kubernetes platform. For example, with Kubernetes and Knative, an enterprise operations team can use an on-premises infrastructure to act as a cloud provider, simulating a serverless environment for their development teams. This also gives companies full control over privacy while still providing advantages to the developers.

The Flexera 2019 State of the Cloud Survey shows that serverless is the top-growing extended cloud service for the second year in a row, with 50% growth over 2018 (24% to 36% adoption). Let's look at the reasons behind this growth. Serverless has many advantages:

Cost

Dynamically scaling compute resources is more cost efficient than setting aside resources and then managing their utilization for optimum efficiency. Because with serverless you are charged based upon the time and memory actually used when the function executes, costs incurred are measured in milliseconds instead of hours. (Note: because it is dynamic, costs can also be less predictable and place a higher burden on monitoring the app's execution.)

Elasticity

The cloud provider scales the capacity to the demand; thus, it scales down as well as up. This elasticity is what affords a lower cost.

Productivity

You save not only on cost of the resources consumed, but also on maintenance. Serverless eliminates the management overhead required in the resource optimization effort. Developers can run the code themselves without waiting on service tickets from infrastructure nor operations. The developer need not worry about multithreading, because each function independently consumes its own compute resources.

Given the advantages of serverless for the development teams, the CISO can expect a push for greater adoption. Understanding the security implications of serverless is paramount. Even though cloud computing pushed more of the security burden to the cloud provider, serverless pushes even more of the risk, and fundamentally changes not only your application security program but your network security, as well. Let's look at why this is so.

Securing serverless applications. Serverless, and sometimes cloud computing in general, is mistakenly considered as more secure than traditional architectures. Although operating system vulnerabilities are identified and patched by the cloud provider, new attack surfaces are introduced by the additional components (such as containers and orchestrators) leaving each component an entry point to the serverless application. Moreover, traditional network security solutions like an intrusion detection/prevention system (IDS/IPS) become irrelevant to the serverless application, whose traffic is multithreaded and bandwidth dynamically deployed. Serverless, FaaS environments from AWS, Google, and Azure provide compute, storage, database, and network resources, whereas their IT user is responsible for code, configuration, data, and endpoints. The challenge for security is the inability to control anything below the application layer, leaving many entrenched security methods irrelevant.

You can expect cloud providers to add security capabilities to help, potentially even monetizing the added capabilities. In the meantime, however, the added complexity of these additional components puts a greater burden on the CISO to understand and incorporate policies into the security program.

The security complexity is additive. In addition to securing the components of cloud native applications (cloud, orchestration, microservices), serverless applications have additional security considerations. The solution to securing serverless apps can be equally complicated. Hackernoon offers a great explanation of the pros and cons of serverless and I've incorporated some of their ideas here to break serverless down further for a security perspective.

Serverless changes the attack surface and requires distribution of security controls over a larger, more diverse set of systems that build upon a microcosm of little functions. Serverless functions don't translate to entire applications that are human understandable, and so a proper web application is used to tie all of the serverless functions and microservices together. The characteristics of serverless that affect security include the following:

Everything becomes an API

- Serverless functions are accessed only as private APIs, not directly through the usual IP. Are APIs part of your security program scope? Many enterprises do not have a thorough application security scanning program, much less consideration for securing the APIs used by the applications. The Open Web Application Security Project (OWASP) offers a Cheat Sheet for securing RESTful APIs.
- Traditional network security solutions like web application firewall (WAF) must be reconfigured. The WAF doesn't become entirely irrelevant, because the underlying request/response architecture still runs on TCP/IP and some port and protocol is still used, but changes are needed because the WAF might struggle with the protocols and complex message structures of a more diverse ecosystem.
- To access these, you must set up an API gateway. This becomes a single point of failure (target).

Secrets management

A Security Token Service (STS) generates temporary cloud service credentials (API key and secret key) for users of the application. These temporary credentials are used by the client application to invoke the cloud API. As the attack scenario that we presented earlier shows, access to secrets can lead to keys to your cloud service kingdom.

Statelessness

- Because with serverless everything is stateless, you can't save a file to disk on one execution of your function and expect it to be there at the next.
- Any two invocations of the same function could run on completely different containers.

- There is a greater burden on the application itself for logging.

Ephemeral

Containers are designed to spin up quickly, do their work, and then shut down again. They do not linger unused. As long as the task is performed the underlying containers are scrapped. Container security must be equally dynamic.

Event-triggered

Although functions can be invoked directly, they are usually triggered by events from other cloud services such as HTTP requests, new database entries, or inbound message notifications. Rules used to automate security policies will need to be reviewed and potentially modified.

Scalable by default

- With stateless functions, multiple containers can be initialized, allowing as many functions to be run (in parallel, if necessary) as needed to continually service all incoming requests.
- This scaling process for serverless is automatic and seamless. Monitoring or configurations are needed to prevent runaway processes and cryptomining.

Microapps

- Time limits on compute resource utilization force applications to be segmented into more bite-sized functions, with the frontend and backend broken into separate pieces. The concept of a web app (for application security scan charges and work effort) becomes more piecemeal. Look at how scanning vendors are charging for this rapidly growing number of smaller apps.
- Because single-function apps are reused across a variety of software solutions, a single flaw can be applied globally. Understanding where they are used (via a Bill of Materials) can help you quickly isolate vulnerable code.
- It goes without saying that breaking down apps into microapps will increase the communication among them. Monitoring the growing number of APIs becomes much more important.

Self-contained

- Most if not all of your projects have external dependencies that rely on libraries for subroutines like cryptography, image processing, and so on. Without system-level access, you must package these dependencies into the application itself.

- If you have instrumented your apps (e.g., RASP or IAST) to monitor core library calls, their effectiveness will depend upon the overall architecture but reconfiguration might be necessary.

Flexible environments

Setting up different environments for serverless is as easy as setting up a single environment. Given that it's pay per execution, this is a large improvement over traditional servers, you no longer need to dedicate dev, staging, and production servers. For security, it might no longer be as simple as protecting what is moved into production. Policies will need to be added to consider this newfound flexibility and potentially monitor and inspect development practices.

Composed

Debugging options for serverless-based applications are limited and more complex. As a result, developers might use error messages that divulge sensitive data. Data loss prevention and other security methods should consider the connections between applications and the logs that each creates.

The Cloud Security Alliance (CSA) together with PureSec (acquired by Palo Alto Networks) have crafted a serverless security guide, highlighted in DarkReading in March 2019. This guide, "The 12 Most Critical Risks for Serverless Applications", was written for both security and development audiences dealing with serverless applications but goes well beyond pointing out the critical risks for serverless applications. It also provides best practices for all major platforms. The risk categories are defined as follows:

1. Function Event-Data Injection
2. Broken Authentication
3. Insecure Serverless Deployment Configuration
4. Overprivileged Function Permissions and Roles
5. Inadequate [Application] Function Monitoring and Logging
6. Insecure Third-Party Dependencies
7. Insecure Application Secrets Storage
8. Denial-of-Service and Financial Resource Exhaustion
9. Serverless Business Logic Manipulation.
10. Improper Exception Handling and Verbose Error Messages

11. Legacy/Unused Functions and Cloud Resources
12. Cross-Execution Data Persistency

Traditional security has focused first and foremost on the network, endpoints, and users. To protect serverless applications, investments will be needed in nontraditional security solutions. These examples explain why traditional security tools will be challenged, at best, and obsolete at worst, in a serverless environment:

- Serverless functions consume data from a wide range of event sources, such as APIs, message queues, cloud storage, and more. This diversity increases the potential attack surface dramatically, and some messages use their own unique protocols that cannot be inspected by standard application-layer protections, such as WAFs.
- Even though insecure third-party libraries are not unique to serverless, being able to detect malicious packages is more complex in serverless environments given the inability to apply network and behavioral security controls.
- Serverless architectures bring automated scalability, but they also bring risk of distributed denial of service (DDOS) attacks and cryptomining because resources are scaled automatically as consumed. Monitoring application execution becomes as important as monitoring network traffic and traditional resource consumption.
- Just as debugging tools can find it challenging to traverse applications composed of many microservices, security scanners might also encounter similar challenges. At the very least, application security scanners that charge by the application will find it difficult to appropriately price their software and/or services in this new environment dominated by bite-sized application functions.
- Because serverless applications contain multiple distinct functions that are stitched together using event triggers and cloud services (e.g., message queues, cloud storage, or NoSQL databases), SAST will fall short when analyzing data flow in these new scenarios.
- DAST and interactive application security testing (IAST) can also be impaired when serverless applications use non-HTTP interfaces to consume input. Furthermore, depending upon the

IAST solution, the instrumentation agents might be impaired in a serverless environment.

- One direction that some security vendors had started popularizing around 2016 was connecting endpoint security with server security, using the endpoint as a sentinel. When one endpoint encountered an attack, it could alert the network and protect the servers. However, with serverless applications, not only does this method not work, but because you do not have access to the physical (or virtual) server nor its OS, you cannot deploy traditional endpoint protection to the server, host-based intrusion prevention, WAFs, and so forth. Additionally, existing detection logic and rules have yet to be "translated" to support serverless environments.

Serverless security is coming from nontraditional sources. The cloud providers are working to help secure serverless applications. Analyst firm 451 had a take on RSA 2019 that suggests that cloud vendors and SDLC platforms will take on more application security responsibilities. It says, "These [traditional security] players face a direct challenge, not from other vendors of stand-alone products in markets segmented from the rest of IT, but from the cloud hyperscalers and innovators with regard to how IT is developed, deployed and run." The report goes on to say, "These disruptors are redefining the very nature of technology, and are incorporating security more directly into concepts from 'GitOps' to cloud-native techniques."

As evidence, Bill Kleyman advises the following in his article, "Serverless 101: Why It Matters for Data Center Professionals":

- Amazon allows you to securely control access to your AWS resources with AWS Identity and Access Management (IAM).
- Manage and authenticate end users of your serverless applications with Amazon Cognito.
- Use Amazon Virtual Private Cloud (VPC) to create private virtual networks that only you can access.
- Azure Active Directory (Azure AD) provides cloud-based identity and access management. Using it, developers can securely control access to resources and manage and authenticate the users of their serverless apps.

But be aware that with this service comes limitations, as well. Kleyman adds that "you will be very limited in the kinds of security tools you can deploy within the network or the endpoint. For example, data loss prevention (DLP), intrusion prevention and detection (IPS/IDS), and even endpoint detection and response (EDR) may not work with your serverless compute platform."

Amazon provides an Architecture Overview for how a WAF is included within its architecture. The AWS WAF acts as central inspection and decision point for all incoming requests to a web application. Currently its capabilities include whitelisting and blacklisting IP addresses, SQL Injection, cross-site scripting (XSS), HTTP Flood, scanners/probes, IP reputation, and bad bots. These capabilities have to be more generic than a WAF used by only one enterprise and tuned uniquely for their applications.

At the same time GitLab and Microsoft are adding security capabilities to the SDLC. GitLab has not only embedded security scanning into its CI capability, but has a vision to provide security monitoring and remediation for cloud native and serverless applications in its runtime environment.

Serverless and DevOps have a symbiotic relationship. The significant advantages of serverless are going to increase demand from development teams whether they consider themselves "DevOps" shops or not. Yet, the very framework itself will push a change toward how software is delivered and executed and will drive enterprises toward adoption of DevOps practices.

Recognize that serverless computing is still very new. Expect to see many new startups jump into this vacuum, with healthy market consolidation yet to follow. Be wary of single-purpose software making claims that they will secure your serverless applications. Given the breadth of the security challenges, a holistic security program will be required.

Security Considerations of Serverless

Given the dynamic nature of serverless applications, the only way to protect them will be through automation, embedded within operations for monitoring and automated runtime response. Inventory the serverless applications used at your enterprise and evaluate options for their runtime protection via runtime container security tools. Watch the market for emerging tools that will protect FaaS architectures that do not use containers. In the meantime, pay special attention to configuration settings across the ecosystem and focus on good security hygiene like scanning all code changes across the entire code base.

Also note that because serverless requires little or no coordination between dev and ops for resources, traditional change management processes might also become irrelevant. If security relies on change management to notify security about new applications, new methods of notification and engagement might become necessary.

Summary: How software is composed and executed

Cloud native and serverless are emerging architectures, still early in their adoption. Yet, the benefit they bring is substantial and security teams should expect this trend to continue rapidly. Entirely new attack surfaces must be considered that will affect what is scanned during development and what is monitored and protected in production. The makeup of next-generation software is important, but so is the methodology used to deliver and manage it. This too will affect security via new workflows and processes, which will challenge traditional security tools.

Shift 2: How Software Is Delivered and Managed

The world is becoming dependent upon software. Delivering that software faster and more efficiently is making a difference between explosive growth and rapid irrelevance of startups and blue-chip companies alike. Punctuating this sentiment, Marc Andreessen famously said, “Cycle-time compression may be the most underestimated force in determining winners and losers in tech.”

To achieve cycle-time compression and increase software velocity, businesses of all sizes, along with public sector, are modernizing their software development and delivery through a combination of changes involving people, processes, and tools. This change is often characterized as *DevOps*. These DevOps methodologies rely on *Lean Six Sigma* constructs of measurement, automation, and iteration to improve processes to increase software velocity,

NOTE *DevOps* can be defined as the convergence of people, processes and tools to enable adaptive IT and business agility.

In *The Phoenix Project* book mentioned earlier, the authors describe how process discipline, helped by automation, along with measurements and feedback, can change the efficiency of the modern software factory. DevOps empowers the developer to spin up hardware environments, storage, requisite libraries, and other resources to quickly begin building code while eliminating friction and dependencies with other departments.

Enterprises are rapidly adopting DevOps methodologies as a way to speed deployment, improve efficiency of developers, and even reduce risk to the business.

DevOps adoption

As far back as 2017, RightScale's report on the State of the Cloud showed rapid adoption of DevOps. It found that overall DevOps adoption rose from 74% in 2016 to 78% with enterprises reaching 84%. 30% of enterprises were adopting DevOps company wide, up from 21% in 2016.

DevOps and this new world of borderless IT and portable apps can be scary, requiring new ways to manage and secure it. DevOps is typically started by a somewhat rogue group within IT and often seen by the CISO as running hell-bent to move faster without necessarily contemplating, nor understanding, the potential risks. When security is added to the mix, it often takes one of two paths. Either the speed and efficiency of DevOps grinds to a halt when it must pass a more sequenced and traditional security gate at the end of the process; or, DevOps is given a sort of hall-pass where it can push fast and security flaws are removed after the code is pushed to production. Neither path is optimal.

To achieve more rapid velocity and reduced cycle time, automation is key. The challenge has been that tools to provide the automation have been very single-purpose tools with very narrow scope. Stitching them together to solve the end-to-end needs of a modern SDLC can consume countless resources. Yet new tools continue to spring up and flourish.

This rapidly evolving environment of software development tools makes integration a bit of a moving target. Many have tried to capture the list of literally hundreds of tools, but the lists quickly become outdated. For instance, though the image in this blog post has become popular, but it doesn't reflect the fact that GitLab now has capabilities in most of the boxes, including CI, where Forrester even placed it as a leader in the CI Wave in 2018. Zebia Labs' attempt in its Periodic Table of DevOps tools is similarly behind.

According to the 2019 IDC Worldwide DevOps Software Tools Forecast 2019–2023 abstract, "The worldwide DevOps software tools market achieved a level of $5.2 billion in 2018. The market is forecast to reach $15 billion in 2023, driven by continued enterprise adoption of highly automated CI/CD, infrastructure provisioning, DevSecOps best practices and advanced infrastructure, and application monitoring and analytics for production as well as dev and test use cases," impacted by "enterprise production support for DevOps-driven applications."

Corroborating this growth, this May 2018 *Forbes* article, "DevOps Dollars: Why There's Big Money In Fast Software Development", claims the DevOps market size could be $12.85 billion by 2025 saying, "These numbers are easily comparable with the 5G mobile network market and Amazon's cloud business."

What's behind the growth of DevOps. The benefits of DevOps can be broad, including not only speed and efficiency of the SDLC, but also flexibility and response to market shifts and happier developers. One of the more impactful benefits is how more rapid, iterative development can help companies rapidly launch small but meaningful changes, assess their success, and frequently adjust. A DevOps methodology is how they achieve these results.

GitLab is one example of a software business using advanced DevOps methodologies with rapid iteration to achieve explosive growth in both software capabilities and in associated revenue. With a formal launch every month and production deployments averaging more than 60 per day, the company has honed the velocity of its SDLC through automation (using its own GitLab software) and iteration (see GitLab values).

By making small, iterative improvements (as shown in Figure 7) instead of lengthy, complicated software updates, the software's success can be evaluated in terms of performance, usability, and purpose. These smaller steps, called Minimal Viable Changes (MVC) or Minimal Viable Products (MVP), provide benefits while allowing the software owner to evaluate results and quickly adjust next iterations, as needed.

Also within the DevOps methodology, cross-functional collaboration and concurrent, efficient workflows have contributed to GitLab's success. Although these capabilities can be achieved across the entire SDLC, from collaborative planning, to multithreaded development, to CI/CD, if we look solely on code development as an example, the impact is easy to see.

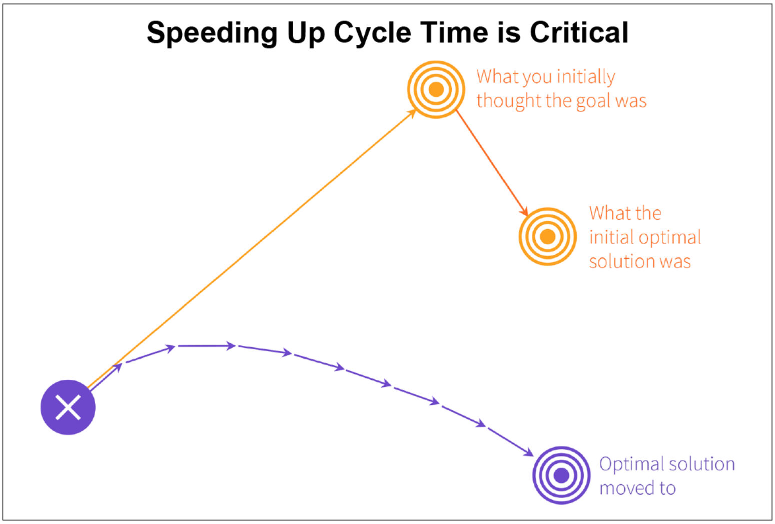
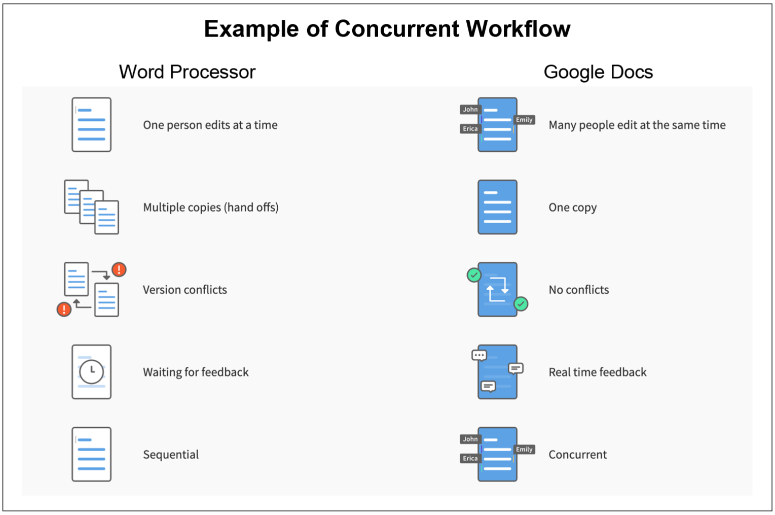

Speeding Up Cycle Time is Critical
What you initially thought the goal was
What the initial optimal solution was
Optimal solution moved to

Figure 7. Incremental improvement affords frequent goal calibration. MVC requires more small, frequent software updates rather than lengthy projects.

Because individual developers can check out a branch to work on (essentially make a copy of the code they want to work on) and then have the repository keep track of checking code back in and identifying conflicts, multiple developers can work on a given project concurrently. Figure 8 shows how this workflow is analogous to working in Google Docs versus a more traditional single-threaded document review. This more-collaborative approach allows more developers to contribute to a given project and push more, smaller code improvements.

Code repositories were just the beginning. The next big area of impact has been CI/CD. *InfoWorld* has a great article, "What Is CI/CD?" that concisely describes CI/CD, stating, "The CI/CD pipeline is one of the best practices for DevOps teams to implement, for delivering code changes more frequently and reliably."



Example of Concurrent Workflow

Word Processor	Google Docs
One person edits at a time	Many people edit at the same time
Multiple copies (hand offs)	One copy
Version conflicts	No conflicts
Waiting for feedback	Real time feedback
Sequential	Concurrent

Figure 8. Concurrent development workflow analogous to using a code repository to manage code among many developers.

The GitLab 2019 Global Developer Report, with feedback from more than 4,000 respondents, reveals that CD—a cornerstone of DevOps—is an area developers see as critical. Of those surveyed, 43% said their organizations continuously deploy (meaning on demand deployment and/or multiple deployments each day) and 41% said deployments happen between once per day and once per month. Just 13% reported deployments occurring between once per month and every six months. These are aspirational results for most and reflect the fact that many of the survey respondents are fairly mature in their DevOps programs.

The benefits of CD are clear: developers surveyed go on to say product/project managers are 25% more likely to have a better sense of dev team capacity in a CD organization than in a company that deploys between once per month and once every six months. And 47% agree those same managers are more able to accurately plan and scope features in a CD environment.

Challenges remain for an immature DevOps. Despite the adoption of DevOps methodologies, frustration remains. This May 2017 TechTarget article, summarizing a more in-depth Gartner report, says that despite $3.9 billion having been spent on DevOps software in the previous year, more than half of the time spent on DevOps was wasted on logistics and repeatable tasks. Although some aspects of the life cycle have been automated, this is not moving the dial. In fact, almost 90% of the organizations that have already adopted DevOps report being disappointed with the results.

Although improvements have been made for specific Key Performance Indicators (KPIs) like deployment frequency and change failure rate, the full transformational promise of DevOps has not yet been realized. To fill the void, even more single-purpose DevOps tools continue to race to provide solutions. The promise of improvement is significant enough to draw enterprises, vendors, and investors alike.

Current DevOps tools are inefficient and complex

Fragmentation of so many new tools, most built for a single function, only further complicates the DevOps situation. With so many tools, integrated like a patchwork of rags into a makeshift blanket, it's no wonder that enterprises have not been able to achieve process improvements and streamlined deployments.

A commissioned study conducted by Forrester Consulting on behalf of GitLab, "to evaluate current toolchain management practices at enterprise organizations," provides interesting insight into the challenges organizations face with regard to DevOps tools. The study, "Manage Your Toolchain Before It Manages You", was published in June 2019 and concluded that "Dev and Ops teams agree: visible, secure, and effective toolchains are difficult to come by." The survey polled "252 IT professionals across the US, UK, France, and Germany," finding that:

> [M]any software development teams are overrun with tools and toolchains and struggle to maintain discrete toolchains. Out-of-the-box toolchain management solutions are seen as a potential solution to managing this complexity.

The survey's key findings are as follows:

- Multiple toolchains and numerous tools are used across the SDLC. Most businesses have two or more toolchains powering their SDLC, and, for the majority, each toolchain comprises six or more tools.
- This complexity creates visibility, security, and productivity challenges for development and operations teams alike. Toolchain managers struggle to contain the tool sprawl while at the same time software delivery release cycles have not improved, creating more pressure for development teams to find ways to speed delivery.
- Out-of-the-box toolchain management solutions are seen as a solution to the sprawl. In fact, improved security, increased revenue, and improved quality were the top-seen benefits from firms that have already implemented out-of-the-box toolchain management systems.

In addition, "The key challenges are: visibility across toolchains; maintaining security across tools; and ensuring that IT resources are available to maintain toolchains."

In its recommendations, Forrester notes, "this expansion of tools has come at a cost in terms of complexity," and, "All of this complexity robs resources from the main purpose of DevOps, to help software teams accelerate the delivery of innovation to the marketplace in order to achieve business success."

One of the most striking results of this study is that, "67% of respondents agree that handoffs between teams using different tools slows down delivery." Without doubt, the challenges inherent in complex toolchains have led to the dissatisfaction with DevOps noted previously.

GitLab has highlighted this fragmentation of competitive DevOps tools (as shown in Figure 9) by showing the multitude of tools and the functional slice of the SDLC that each provides. Enterprises will often stitch together a variety of tools to meet their end-to-end needs, when the potential is there for using a single tool, and eliminating the complexity and integration requirements.

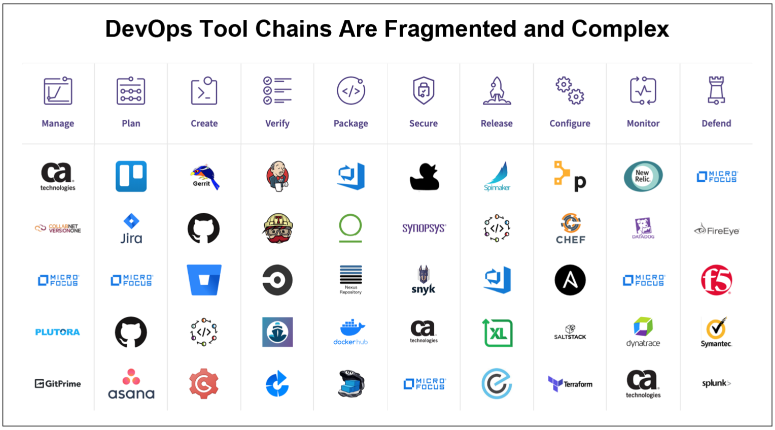


Figure 9. GitLab's chart showing competing tools for each function punctuates toolchain fragmentation, silos, and complexity. There is an abundance of point products for specific pieces of the SDLC.

When stitched together, these complex toolchains, like the example in Figure 10, reinforce process silos, consume self-perpetuating maintenance resources, and slow down software development and deployment. In addition to the tools themselves, resources are needed to design, build, and maintain the integrations, manage upgrades, and establish high availability (HA) and disaster recovery (DR) for each one. The integrations are brittle and can fail. Transparency and metrics become challenging across the silos. They can also affect the security of the code itself and the integrity of its delivery process.

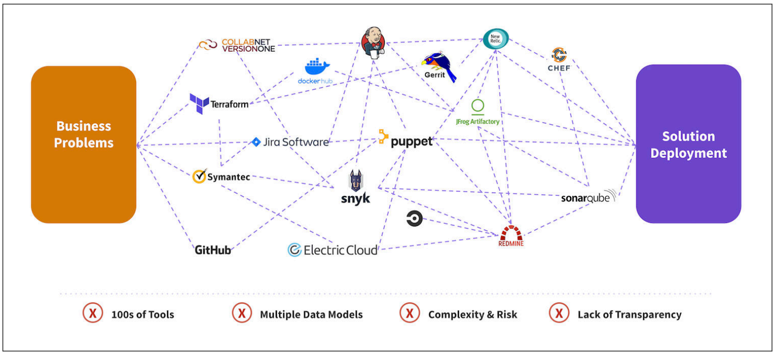


Figure 10. Current DevOps toolchains are a patchwork of siloed apps and functions that inhibit auditability and introduce more security risk.

Let's look at how this fragmentation and integration diminishes the returns of DevOps.

Complications across the DevOps toolchain. The commissioned study conducted by Forrester Consulting on behalf of GitLab, "Manage Your Toolchain Before It Manages You", also found that "Over a third of toolchains are integrated with a combination of plug-ins and scripts; one in five toolchains are integrated via manual, hard-coded custom integrations, a process that is not only time-consuming but rife with potential errors."

Furthermore, "Almost half of all respondents (45%) noted that maintaining security across the toolchain is a key challenge; each tool has its own diverse set of requirements which creates significant challenges for IT professionals to not only develop but also maintain. Forty-six percent of respondents agreed that they spend too much time and money integrating and maintaining this diverse security landscape for each tool."

Security tools complicate the already complicated DevOps toolchain. Application security companies have built their portfolios through acquisitions. At the same time, they have done a relatively poor job of integrating their acquisitions, choosing instead to sell each acquired scanning technology as an individual product and uniting them with an overall dashboard that they are also happy to sell to you as yet another product.

In addition to the multiple security products needed to complete static and dynamic security scans, each tool must be integrated into an already-complex DevOps toolchain (see Figure 11). Trying to connect multiple security tools into the byzantine and ever-changing DevOps toolchain is a bit like trying to run into a flying jump rope. It's constantly moving and you might have difficulty finding the optimal point of engagement. It can quickly become a nasty web of integrations that must be maintained and which consumes precious resources.

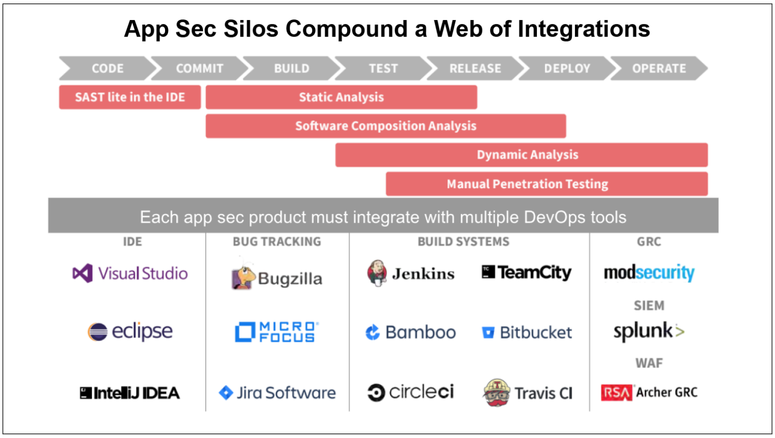


Figure 11. Individual application security testing tools must each be integrated with multiple DevOps tools.

While at Fortify, I designed and led a study using a third party to assess the intersection between development operations and security. I had a third-party company study what development thought about security, what operations thought about security, and all of the tools that both Dev and Ops use in their DevOps toolchain. We wanted to learn with which tools application security might need to interface. At the time, there were more than 50 potential DevOps tools identified with which we might need to interface! It became evident that a marketplace with APIs and plug-ins was going to be critical in the DevOps environment. But you need to ask whether the objective of reducing application risk is getting lost in the process. What portion of your resources are spent integrating tools? Maintaining their interfaces when one changes? Tracking vulnerabilities instead of remediating them?

It's not enough to simply integrate traditional application security with DevOps tools and expect that you're going to get a completely different, improved outcome with greater speed efficiency and business value. You need to marry the tools with the process.

Security and DevOps processes and tools go hand in hand. You cannot have a smooth application security process in a DevOps environment if the two processes and tools are incongruent. To achieve this integration, enterprises try to join traditional application security tools with DevOps tools and then mash this patchwork against a desire for an efficient, streamlined DevOps methodology. Needless to say, the approach falls short.

Security testing: A square peg in the round hole of DevOps

This speed of coding and deployment has challenged traditional security approaches where code is tested at the end of the development process. This newfound speed of Agile development was blocked by an essentially waterfall security process at the end. The industry has attempted to arm developers to find and fix the vulnerabilities they create, but without uniting the dev and sec workflows, the tools have been like fitting a square peg into a round hole.

Several classes of security tools have been introduced, each intending to "shift left" and help developers find and fix vulnerabilities earlier in the SDLC (when it is cheaper to do so); for example:

- Spellcheck-like scanning, included in the developer's IDE, was intended to help developers find and fix vulnerabilities at the point of creation, as the developer types. Yet these tools, often referred to as SAST-lite, are limited in scope and not capable of a robust SAST. Though they can remove some vulnerabilities, their use alone is insufficient for application security scanning.
- DAST requires a working application to test. Frustrated by waiting to dynamically scan until code was deployed to a test environment, demand was created for IAST. While there is a bit of a religious argument, as in this DarkReading article over the efficacy of IAST versus SAST, the fact is most enterprises continue to use multiple scanning methods to identify software security vulnerabilities.

To be truly impactful, shifting left requires a united workflow between developers and application security. In fact, those who have applied the benefits of a fully functioning review app in development have been able to apply DAST before the code leaves the developer's hands (an ability with the potential to obviate the need for IAST). This is possible only by embedding security into the developer's workflow.

The discipline of DevOps alone can contribute to better security practices. Those who have integrated security into DevOps have found it worthwhile. The GitLab Global Developer Report found that a mature DevOps model means teams are three times more likely to discover bugs before code is merged, and they are 90% more likely to test between 91% and 100% of code than in an organization with early stage DevOps. Not doing DevOps well leaves security teams 2.6 times more likely to need to deal with red tape before finding or fixing bugs.

At the same time, there is significant room for improvement.

Testing is in the way. Half of those surveyed in the GitLab Global Developer Report called out testing as the biggest source of delay in the development process, reflecting an industry-wide struggle to balance the benefits of manual testing with the need for automation.

In this same survey, respondents were asked about their opinions on their security capabilities:

- For developers, just 25% rated security at their organization as "good," whereas 30% said it was fair, and 23% said it was poor.
- Like developers, operations professionals have mixed views on their organization's security efforts. 34% rated security as fair, 29% said good, and 20% called it poor.
- Least satisfied with their area of responsibility is security, with only 20% of those surveyed rating their organization's security efforts as good, whereas 36% said they were fair, and 24% said they were poor.

There is good reason for the dissatisfaction. Application security testing, for the most part, continues to be applied without uniting incongruent workflows. As Figure 12 shows, security testing continues to be a sequential process.

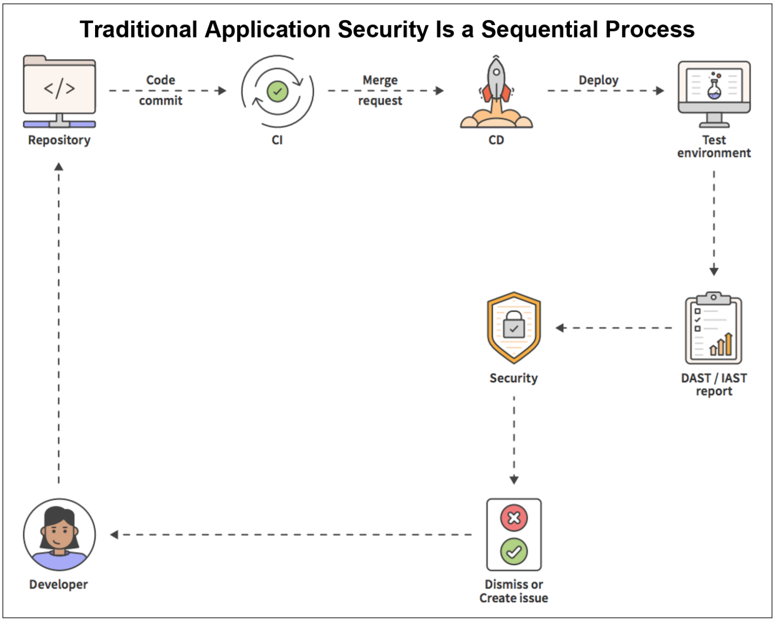


Figure 12. Traditional application security is a sequential process that reinforces silos and creates friction between security teams who run the scans and developers who must take action to remediate the code.

Software is developed to a point at which it can be tested by security professionals (code review for SAST, working app for DAST, manual penetration testing in test or production) with results delivered back to development in the form of vulnerability reports or work tickets. Challenges of traditional app sec workflows include the following:

- Finding the developer who can remediate the vulnerability
- Finding the developer accountable for creating the vulnerability for training and education
- Retesting to ensure the vulnerability was indeed remediated
- Prioritizing security flaws in developer backlog
- Visibility into vulnerability status (assigned for remediation, etc.)
- Separate views for dev and for sec with difficulty tying vulnerabilities found by security testing tools to tickets in dev backlog
- The entire app is scanned, making it unclear which vulnerabilities are new (for ticketing) and resulting in a heap of them to

> prioritize and remediate. A developer doesn't know whether they or maybe a colleague created a vulnerability, or whether it has been lurking in the code for years.

Contrast this to a DevOps developer workflow, which is very iterative. The developer selects a ticket from a Kanban board that will fit within the time available. They check out the necessary code and make updates, iteratively testing it until their change works satisfactorily. This iterative effort often includes a review application during which they can see the results of their changes and perform user testing. Performance testing can be automatically performed before the code is deployed.

Summary: How software is delivered and managed

Trying to marry security tools and processes with the newer DevOps tools and processes is a bit of a square-peg-round-hole problem. It won't be solved by simply inserting sec in between dev and ops and assigning security champions to Scrum teams. Security processes must be married to development processes and embedded within application CI/CD. Doing so not only solves process alignment challenges, but can help with regulatory compliance, as well.

Shift 3: How Software Complies with Regulatory Requirements

Compliance was once the low bar for security. You can certainly be compliant with a host of regulatory requirements and still be hacked. Retailing giant Target was merely one example of companies that are compliant with Payment Card Industry (PCI) standards, yet it was hacked. Application security compliance has been fairly simplistic, focused mostly on demonstrating app security scanning practices to catch the OWASP top 10 exploits and WAF to block obvious attacks.

So, why has compliance taken on more importance lately? The European Union's General Data Protection Regulation (GDPR) is likely a key driver. GDPR reflects strict privacy laws and, as CSO magazine explains, "places equal liability on data controllers (the organization that owns the data) and data processors (outside organizations that help manage that data). A third-party processor not in compliance means your organization is not in compliance. The new regulation also has strict rules for reporting breaches that everyone in the chain must be able to comply with." Protecting customer privacy means attention to data security—and the applications that manipulate data. The result has been a greater awareness of application security risks. Punctuating this fear, British Airways is facing a record GDPR fine ($230 million) from a 2018 breach that leaked 500,000 customer records.

With such high stakes, auditing becomes a way to identify compliance risks proactively. Auditors focus on common controls and love automation and the consistency it provides. Auditors can inspect the automation rules and do not need to see a large sample of results to prove efficacy like they do for manual controls. In addition, traceability and accountability become even more important capabilities to track who changed what and when.

Beyond application security testing

Application security testing is only one small part of regulatory compliance. With greater automation and cross-functional processes of DevOps, there is greater pressure on securing the software development process itself to protect consumers from insider threats, unfortunate mistakes, and fraud. Although each industry has their own regulations with which to comply, there are some common denominators among them. Following are some sample controls to address:

- Segregation of incompatible duties
- Identity and access approval controls
- Configuration management and change control
- Access restrictions for changes to configurations and pipelines
- Protections on branches and environments
- Audit logs
- Licensed code usage
- Security testing

Fundamental industry standards like ISO, PCI, and others focus on these common controls to secure access to the software itself throughout its life cycle. These controls focus on protecting access to the code itself and ensure that changes are logged, including who made the change and when.

Automation of the processes within the SDLC improves consistency, stops casual intervention, and provides greater transparency than manual processes.

Fragmentation creates real security challenges. The commissioned study conducted by Forrester Consulting on behalf of GitLab, "Manage Your Toolchain Before It Manages You", cited security as the number one process challenge faced by IT pros. The report points out that toolchains often rely on credentials stored in scripts in order to reduce process friction across functions; however, these integrations create vulnerabilities themselves. The report suggests, "Selecting a toolchain that is a complete solution allows identity and authentication to be managed in a uniform way across each process step, simplifying and strengthening toolchain security. It eliminates stored credentials and simplifies developer and IT interaction with the toolchain, making shortcuts and hacks no longer necessary."

With the growth of cloud computing, cloud native, and serverless apps, new attack surfaces arose, rendering many network controls irrelevant or out of bounds, and directing a greater focus toward application security. Security tools and compliance standards haven't yet caught up to next-gen software technologies.

Summary: How software complies with regulatory requirements

Beyond vulnerability scanning, prudent risk management requires that you secure the application, your SDLC platform, and the application's runtime environment. Partner with development to embrace its Git repository and CI/CD, which together enable greater automation of security policies. Encourage the use of one tool to provide greater visibility across the SDLC and eliminate the trade-off between easy access across tools and unsafe access methods.

Now that we've addressed these three major shifts that will affect how you secure your next-generation software, let's take a deeper look at what lies ahead for DevOps. It's important that CISOs really understand this movement in order to design an application security program that will meet the needs of this evolving landscape.

What Lies Ahead for DevOps

As we saw earlier in the discussion on DevOps adoption, the DevOps tools market will continue to explode in the near term. Given the fragmentation, we can expect consolidations. Microsoft's acquisition of GitHub threw down the gauntlet for other cloud providers by creating an on-ramp to its cloud services by cozying up to the developers. So, who do you bet on?

Given the shortcomings Forrester documented in the study referenced earlier, their report concluded, "Maintaining toolchains is necessary, but it shouldn't consume whole teams, instead find an out-of-the-box solution that offers a backbone of capability that your team can build upon, rather than building one from scratch." A single application for the entire SDLC can benefit the speed and efficiency of software development. In fact, several GitLab customers have demonstrated compelling results:

Customer	Problem	Result
Hammersbach	Multiple tools and communication inefficiencies	Increased build speed by 59 times; 14.4% improvement in cycle time
Jaguar Land Rover	Slow delivery and release cycles taking four to six weeks	Decreased feedback cycle down from three to six weeks to 30 minutes
Axway Software	Legacy SCM and complex toolchain limited worldwide collaboration	26-times faster release cycle; from annual releases down to every two weeks

At the same time, GitLab's revenue growth speaks volumes.

Why the Single-App Dev Platform Will Lead the Market

There are at least three compelling reasons why a single application development platform will continue to achieve momentum:

1. *Simplicity and integration always win in the long term*

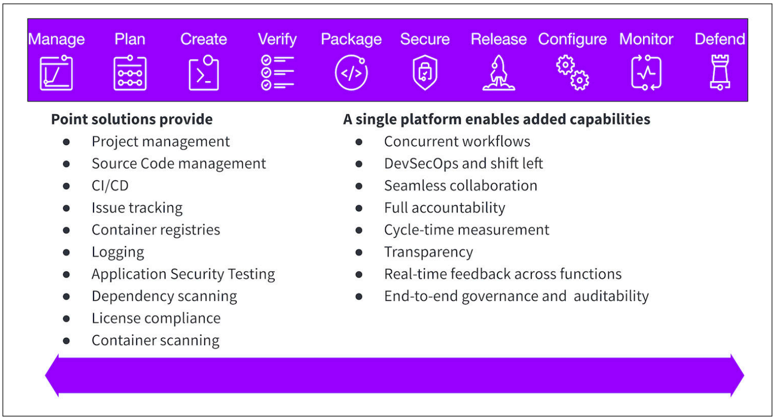
   Take the camera and the smartphone. If you always have your phone with you, you're much more likely to catch that cute pet photo or gorgeous sunset. If after you run and get your camera and adjust its settings, that gorgeous sunset might be gone. How many of us rarely use cameras any more for this very reason? Also, the camera doesn't inherently share. Some models may simplify sharing, but the cell phone has ubiquitous access to your favorite social media and backup storage. This could mean that you reserve a camera for special, prearranged occasions, making the smartphone your go-to device for a multitude of uses.

2. *Collaboration and transparency across functions embody DevOps*
   When various functional groups are united around a single application, they share a single source of truth and have ready access to the data needed to make decisions. Conversely, if they use different tools, the silos in which they work are reinforced. The very heart of the DevOps methodology encourages cross-functional work teams. Their tools must enable this.

3. *A single end-to-end SDLC platform enables new capabilities*
   Just as network security has begun using endpoint security as sentinels to identify threats to the greater network earlier, imagine how a single app to manage the SDLC could use a united view of threats from development through production—and back to development (Figure 13). When there is a single conversation and a single data store, there is no need to try to reconcile data across tools and take extra steps to communicate. A single permission model (user access), single interface, and shared governance ensures consistency of policies across functions and eliminates vulnerabilities created by gluing tools together. Everyone can be on the same page for metrics and life cycle analytics with easy-to-see contributions.

Security Considerations

Even though security embedded into the CI/CD process is ideal, the CISO cannot dictate which CI/CD tools should be used by the development teams. However, the CISO can reduce risk and reap the benefits of a united effort by partnering to integrate development and security processes for better mutual outcomes. A single application for the entire SDLC can help you build security into the way the enterprise develops and deploys applications.



Point solutions provide

- Project management
- Source Code management
- CI/CD
- Issue tracking
- Container registries
- Logging
- Application Security Testing
- Dependency scanning
- License compliance
- Container scanning

A single platform enables added capabilities

- Concurrent workflows
- DevSecOps and shift left
- Seamless collaboration
- Full accountability
- Cycle-time measurement
- Transparency
- Real-time feedback across functions
- End-to-end governance and auditability

Figure 13. A single platform that unites the code repository, CI, and security enables new benefits not possible with disparate tools.

What Does This Mean for Application Security Programs?

Security must be as iterative as your software development and deployment. You cannot expect to successfully marry an iterative development process with a security program rooted in a sequential process.

A single application for the entire DevOps life cycle can enable radically faster cycle times through not only automation but a single source of truth. Developers, security, and operations can all have a common understanding of the application, its vulnerabilities, and the configuration of its operating environment. Complex toolchains that knit together a variety of tools cannot achieve this. They are fragile and resource intense. Examples include Microsoft Office, which won out because it was one tool that could share data across spreadsheets and documents, or SAP, which won out over PeopleSoft and others that focused only on one function. Microsoft and SAP might not have been best in class, but integration won.

One example of new capabilities afforded by a single SDLC platform would be the case of DAST versus IAST. With traditional app security tools, DAST could be done only near the end of the SDLC because it requires a working application to test. The individual developer's work was already merged into the master branch and typically in a dedicated test environment where QA tests would be performed along with DAST. Because DAST occurs so late in the life cycle, IAST was born. By instrumenting the application, the app could be automatically tested for security vulnerabilities while QA tests are done.

Contrast this with a single application. A review app is automatically spun up upon code commit, prior to the merge with the master branch. This functioning app, that reflects the code updates, can be used for DAST analysis before the code ever leaves the developer's hands.

Another example of new capabilities afforded by a single SDLC platform is incremental scanning. When app security testing is done within the CI process, the differential code change is clear. Scanning can focus on only the code that changed and not the entire application. Such capabilities are possible when one application is used for repository, security, and CI/CD.

Security Considerations

By embedding application security into the CI process, you can achieve the following remarkable outcomes:

- Scan all code, every time
- Seamlessly for dev
- Using fewer tools
- With dev and security on the same page

Imagine what can also be achieved when development and production have a common point of view. Vulnerabilities found in development can be confidently blocked in production and exploits against production apps can alert the responsible developer.

The automated software factory promises new ways to develop and deploy more secure code with integrated feedback from apps instrumented from conception, providing feedback from production to development to alter the process for better outcomes.

Defining the Next-Generation Application Security Program

Before we get to pragmatic steps to refine your security program, let's take a 100,000-foot view of the evolution. This strategic context is critical guidance for your more tactical execution.

Application Security Is Difficult but No Longer Optional

If software rules the world, software vulnerabilities might bring it down. Exploits against applications remain common. A Brighttalk webcast, "The State of DevSecOps—Featuring Amy DeMartine of Forrester Research", reveals that 40% of firms suffered a breach as a result of an external attack. The top two methods were web app attacks and software vulnerability exploit.

Application security has traditionally plagued CISOs because it's difficult and presents many challenges:

- CISOs usually do not have a development background. Basic security focuses more on perimeters and endpoints, so application security is often a weak link in understanding.
- Developers don't want to become security experts, and security pros don't want to become developers. Individuals skilled in both are like unicorns. Security finds vulnerabilities but often doesn't know how to remediate them in the code. This sets up an adversarial relationship with developers.
- Traditional application security tools and/or services are expensive, whether manual penetration testing or automated scanners. Unique expertise is often required to translate the findings into action.
- Do I want to find vulnerabilities that I am then aware of (liability) and must convince developers to go fix? The easier path is to invest in a WAF that can offer some level of protection for all of my applications for much less cost and expertise. It checks the box for compliance despite providing only a thin veil of application security.
- It's more difficult than installing a firewall on the network with a set it and forget it approach. It's embedded into the development process—outside of the direct control of the CISO.

Despite all of these difficulties, application security is no longer optional, because threats are constant, widespread, and continually evolving. Countless breaches prove this point over and over again, and most usually come down to simply poor application security hygiene.

Software Innovation Requires Security Innovation, Too

Hackers are some of the most innovative people out there. If you are still relying on processes and technologies established 5, 10, or 15 years ago, you can be assured that your adversaries are not. So how do you keep up?

Traditional app security tools were built 10 or more years ago—before today's modern software methodologies like DevOps with daily deploys. Don't try to catch up using traditional app security. Traditional application security vendors, even the best of breed, are siloed in a single function and will be squeezed out between cloud providers and SDLC software with both sides taking on more of the application security functions. The industry needs to get beyond the simple shift left of giving dev a lite SAST in their IDE. Application security programs scale only when security is baked into the SDLC with a single application that is purpose-built for the modern software factory.

The next generation of software, along with the methodologies that develop, deploy, and manage it, need to be looked at with a fresh perspective. Sometimes, we get too close to the action or too emotionally attached to people, processes, and technology that we have tuned into a well-oiled machine to recognize the potential impact of changes around the bend. If you are truly managing risk, you must reevaluate processes and outcomes given evolving software development and cloud native/serverless landscapes. While the app security market has trusted, established tools, most were developed 10 to 15 years ago for a different world of software development. The tools must now transition to support new processes.

Innovation is a delicate transition. In his book *The Innovator's Dilemma* (HarperBusiness), Clayton Christensen talks about new products and technologies often running parallel to old ones until the new ones are proven and credible. Cybersecurity is no exception. Today's rapidly changing business environment requires that you innovate by not only looking for opportunities to improve on existing security processes, but, more important, to rethink what needs to be done and how. Have you considered what new risks are posed not only by the applications themselves, but your processes that create and deliver them, and composable infrastructure that surrounds them? Or have you jumped straight to specific solutions, promising to help developers identify vulnerabilities as they code, secure your containers, and integrate DevOps tools. Now you are stuck trying to make them all work together and realizing underwhelming returns. Step back and consider entirely new processes and supporting tools for next-generation software projects. New and old processes might both be used for a time as legacy apps are retired.

Jeff Williams, one of the OWASP founders, shared his thoughts on this challenge back in 2014. He said, "The goal is unprecedented real-time visibility into application security across an organization's entire application portfolio, allowing all the stakeholders in security to collaborate and finally become proactive." He further predicts that what's needed is application security "technology that will scale massively." Clearly, even in 2014, he saw that existing app security tools and approaches could not keep up.

The reality of application security is this:

- Applications are a prime target of cyberattacks.
- App security tools are expensive and require integration of both technology and processes.
- Traditional app security processes struggle to test all code when code changes faster and faster.

Practical Advice for Securing Next-Generation Software

Although securing next-generation software might sound like a quagmire, it's also an opportunity. You must look at the problem as one to be solved by automating policies and inspecting the exceptions rather than inspecting every project. Automation applies both standards and consistency to help compose secure code from the beginning. Focus more on helping developers create better code from here on, and less on retrospectively finding flaws. As software velocity improves, more code will be touched more often, so opportunities to improve its integrity will follow. According to the Brighttalk webcast, "The State of DevSecOps—Featuring Amy DeMartine of Forrester Research", "flaws persist 3.5x longer in apps only scanned 1–3 times per year compared to ones tested 7–12 times per year." It follows to reason that embedded, automated scans at the point of code commit will result in more frequent scans and more rapid vulnerability remediation.

Get in Front of the Change: "You Are Here"

With the rapid pace of change, the CISO needs to solve for the future because if they solve for today's legacy software using traditional tools, they will already be behind given the rapid pace of innovation with next-generation software.

As Wayne Gretzky put it, "Skate to where the puck is going." Invest your time to learn what *new* security challenges must be the focus in the next generation of software development. Start with a "you are here" approach and pick your path forward.

How fast does the business need to move to cloud, containers, or serverless? Consider legacy software shifts to containers to free up hardware lock-in. Is security an impediment? Can you take one path for new cloud native software while phasing out the old processes as old apps are retired? Invest in container and dependency scanning and cloud infrastructure security strategically to stay one step ahead of the business's moves.

If you've not already invested heavily in traditional SAST and DAST scanners, rely on security scanning embedded in a single platform for the SDLC, and focus on scanning every code commit starting from now forward. If you lack the resources to find and fix vulnerabilities already lurking in production, you can still prevent new ones from escaping development.

Application scanning is only one part of your risk mitigation. Risks can also come from the application's next-generation infrastructure (as we've seen), from the SDLC platform itself, and from the platform's ability (or inability) to enforce policies for security, compliance, and auditability.

Rethink Security as an Outcome, Not a Department

Security must become everyone's responsibility and must become a natural part of the development and operations workflows. In fact, at the 2019 RSA Conference, the CISO of VMware made this terrific point in his keynote: "Your most important security product won't be a security product."

A close partnership between developers, DevOps, and app security will be paramount. Finding a balance where developers don't own security, but they aren't absolved from responsibility either, requires innovation. Enablement will be possible only when application security is looked at within the context of how the applications are created, delivered, and managed.

Metrics can be incredibly powerful to drive behavior and align incentives. At a CISO roundtable on security, a couple of innovative marquis companies shared how they are using bug bounties to align dev and security. *Bug bounty programs* pay hackers to find and responsibly disclose software vulnerabilities. Typically, these programs are funded by the security or risk management department. However, these companies are requiring that when bugs are found, the bounties are funded by the application's owner. This ensures that the business function is incented to resolve vulnerabilities and will assess the potential business cost alongside the value of the desired software updates.

Although not as directly linked to costs as bug bounties, risk metrics such as number of releases to fix a vulnerability and number of critical vulnerabilities found and fixed in development versus number that escaped can also be helpful to ensure development places adequate importance on remediation. Metrics can help you manage the risks not the silos.

Start with the Process: Use Tools to Enable Change

Partner with development or DevOps to use value-stream mapping (a capability of some DevOps tools) to identify process bottlenecks in the SDLC. Although some might be intuitive, others might be more subtle and best revealed through this end-to-end mapping of actual critical paths and cycle times. Look for opportunities for security to contribute to developer productivity and faster releases.

Consider entirely new processes in which security scans are automated. Yes, I said automated. That doesn't mean pick a tool first. It means design the process around exception handling and assume the mundane work that can be automated is abstracted away. Don't forget audit and compliance processes. How much of that process can be automated, again leaving exception handling?

Solve for Obvious Cases: Breadth over Depth

Often, we become so focused on one aspect of security that we go very deep on those protections and leave many other, sometimes obvious, aspects completely exposed. Are you putting many locks on your door and leaving your window wide open, as our friend in Figure 14 demonstrates?

- Are you using a very powerful scanner for your mission critical apps but not scanning others?
- Maybe you're not scanning your third-party code because you expect it's in widespread use so it already checked out? (Think Apache Struts 2.)

Solve for the obvious cases. Given that most exploits tend to (as described in this Security Boulevard article) reuse historically successful vulnerabilities, it's better to scan everything for the most common vulnerabilities than to scan only your mission-critical apps looking for every little thing.

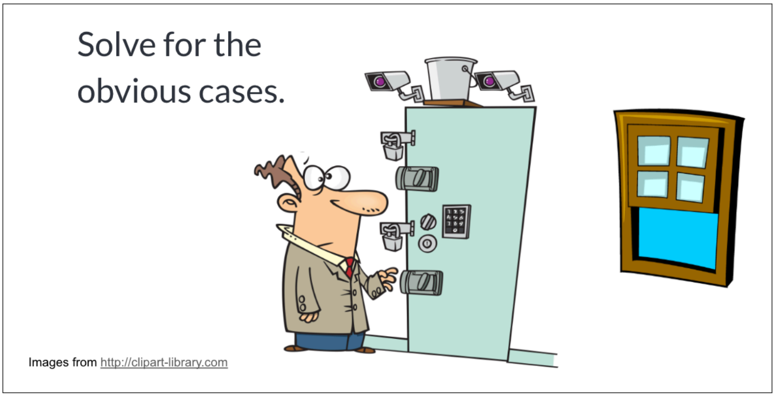

Solve for the obvious cases.
Images from http://clipart-library.com

Figure 14. Don't ignore the obvious. Test all code changes, automatically, every time.

Consider a process similar to a TSA screening at the airport. Everyone is scanned for the obvious things like knives, metal, and sharp objects. Frequent travelers are prescreened in advance to reduce risk and then enjoy a simplified screening effort at the airport. Only exceptions and random samples require a more rigorous assessment.

Analyst research supports this approach. From the Brighttalk webcast, "The State of DevSecOps—Featuring Amy DeMartine of Forrester Research":

> There are just too many vulnerabilities for organizations to tackle all at once, which means it requires smart prioritization to close the riskiest vulnerabilities first. For the first time, our report shows a very strong correlation between high rates of security scanning and lower long-term application risks, which we believe presents a significant piece of evidence for the efficacy of DevSecOps. In fact, the most active DevSecOps programs fix flaws more than 11.5 times faster than the typical organization, due to ongoing security checks during continuous delivery of software builds, largely the result of increased code scanning.

On top of broad-based scanning for common vulnerabilities, applying good hygiene is still the key. Use automation to enforce it.

Unite the Workflow of Development and Security

The process that unites the workflows of development and security needs to be automated, iterative, and exception-based to be congruent with DevOps. It must put the developer at the center, with enablement focused on removing straightforward vulnerabilities and on helping the developer find and fix what they have introduced. The key is making this enablement *before the code is merged*. Let's look at how this is achieved.

Automated and iterative

What does this mean, "before the code is merged"? The developer sees the results immediately of every change they made with very clear cause and effect. It means that as a developer, I made this change in code, and I immediately see the results of that change. I don't see results from my colleague three desks down, nor vulnerabilities that have been lurking in the code for several years. This very tight, iterative feedback loop makes the results far more actionable. It narrows down where the developer must look.

Vulnerabilities and license risk are shown within their pipeline as part of their natural flow (Figure 15). They can see that everything passed—and if it did not, they would see that, too, along with information to help them remediate the flaw.

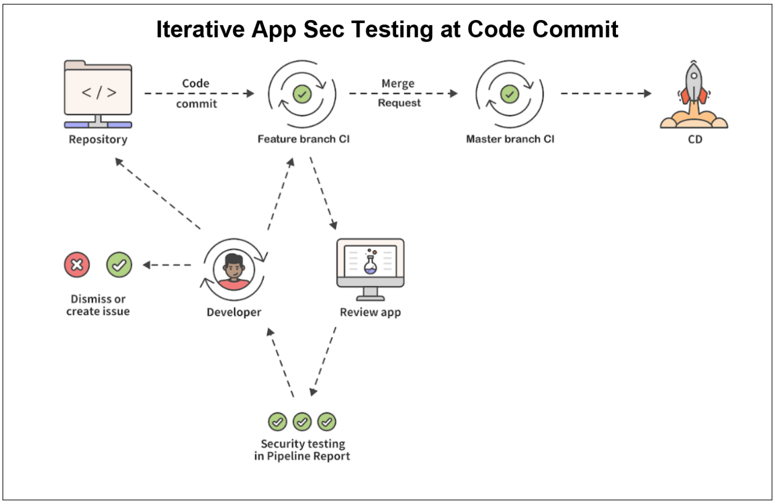


Figure 15. Continuous security = Iterative app sec testing at code commit. By testing the code change and showing resulting vulnerabilities to the developer before the code leaves their hands, the cause and effect is very clear.

In addition to greater code coverage in app security scans, such an approach improves the workflow of both dev and app security while reducing friction and frustration. Here's how:

- Continuous security scanning will empower dev to identify vulnerabilities that they just created, before it ever leaves their hands—or gets mixed up with vulnerabilities resulting from other engineers' changes. It provides clear accountability and tracking within the context of their existing workflow. The developer doesn't wonder where the vulnerability came from, and security doesn't struggle to find the responsible owner.
- Everyone sees the end-to-end flow, delivery, and results and resolves remaining vulnerabilities together (single source of truth).
- No hand-offs between tools and departments. No waiting for status updates concerning testing or remediation.

- The day-to-day risk tracking is automated and becomes a byproduct.
- Security teams can focus on exceptional security challenges, with mundane tracking and communicating across functions eliminated.

It also enables capabilities that couldn't otherwise happen, including autoremediation and DAST at the point of code commit: vulnerabilities that can be automatically remediated are also within the context of their existing workflow.

Autoremediation can be done well only when detection is embedded in CI. A complete autoremediation can do the following:

- Identify necessary fixes when vulnerabilities are detected in the app.
- Download patches.
- Set up a new branch to test the patch in a background pipeline to reduce noise.
- Verify changes for production readiness, and auto-revert if changes don't meet performance- or service-level objectives.
- Provide validated updates that can be pushed automatically to production.
- Provide audit trail with merge.

With a single app for the SDLC, a review application is spun up at code commit—before the individual developer's code is merged to master. The developer can see and test the working application and DAST can scan the review app. The developer can quickly iterate to resolve vulnerabilities reported in their pipeline report.

Exception-based

The developer can resolve the vulnerability with another code commit or can create an issue to resolve it later, having it documented in the work queue. Or they can dismiss the vulnerability, determining if it is a false positive or if there is a compensating control. The dismiss can trigger the exception process for review by security, as depicted in Figure 16.

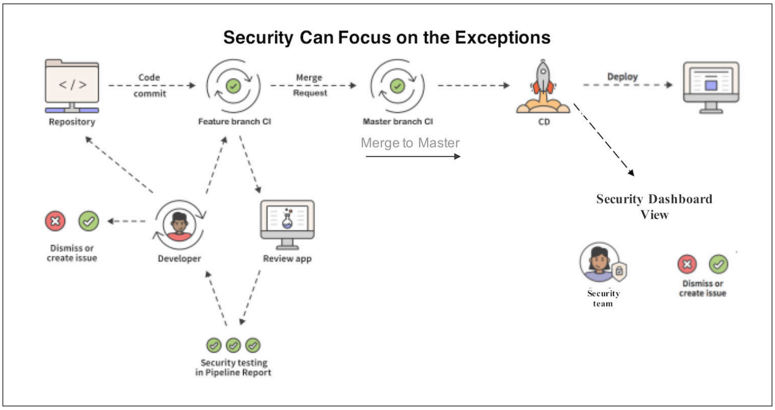


Figure 16. Automation can remove work for both developers and security alike. Developers can resolve what they can, and security can focus on the exceptions.

The app security team might not see many vulnerabilities because they can be identified and resolved before the code ever leaves the developer's desk. When even the resolved vulnerabilities are reported, the resulting metrics can demonstrate the value of the approach.

Iterative

How do you swallow an elephant? One bite at a time. Security too must be incremental and iterative. Enterprises that scan an entire application only to find 10,000 vulnerabilities will find it difficult to prioritize and digest them all. The traditional approach of a big monolithic security scan toward the end of a software's development cycle simply does not work in the iterative workflow of Agile development and DevOps. However, if application security scanning is iterative, congruent, and contextual within the developer's code commits (code changes) and CI/CD processes, vulnerabilities can be found as they are introduced and assessed and/or resolved at the point of origin, as demonstrated in Figure 17.

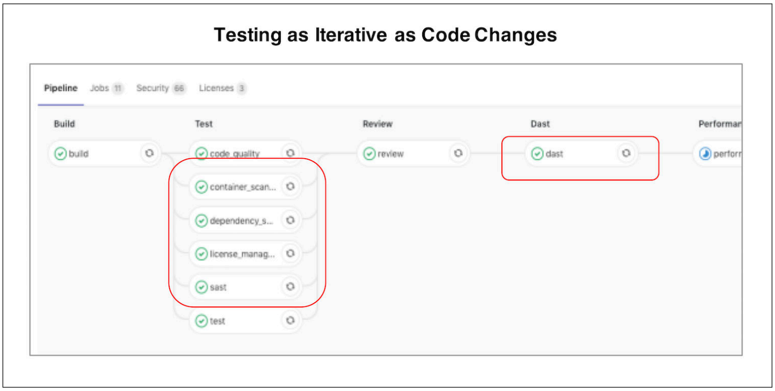


Figure 17. Embed security scanning into CI for seamless and automated workflow.

Security can be implemented in small, iterative steps when automated as part of CI.

Collaborative

Not only do developers and security need to collaborate, but they need to do so using common tools and views of the security scanning results. Today's teams struggle to communicate what the vulnerability and its risk is, where it is in the code, and how can it be remediated. Developers are frustrated that security only points out the flaw but cannot help resolve it. Security is frustrated that they are spectators to remediation efforts. For those enterprises working to engage security pros into Scrum teams, separate tools can be a hurdle. By having both disciplines use a united view, they can spend more time on value-added efforts rather than reporting and interpreting.

The cumulative effect of this new, united workflow brings the following benefits:

Contextual

- Within CI/CD dev workflow—accountable person
- MR pipeline for dev
- Shared view between Dev and Sec

Congruent with DevOps processes

- Iterative within dev, tests every code change

- Immediate cause/effect of code changes

Integrated with DevOps tools

- Create issues
- Autoremediation
- Production feedback

Efficient and automated

- Eliminate work wherever possible
- No context-switching
- Less tracking/triaging and more value-added security help

Monitor and Protect Applications in Production

As serverless brings new components into the framework, applying policies to the various components becomes more tedious and error prone. What's needed is a policy translator for which security policies are set in English and then automatically translated into individual component settings (such as AWS, Docker, Kubernetes, etc.). In short, composable infrastructure requires composable and automated security.

Applying Zero-Trust principles to protect applications

Composable application security should be thought of as similar to Zero-Trust principles, but for applications rather than the network. Traditionally, Zero Trust assumes that you do not trust the user nor the user's device. The user must prove that they are who they say they are and that they meet policy requirements to perform the actions they are wanting to perform. The user is authenticated and authorized. In addition, the device must prove that it is what it says it is, including patch levels. The device is authenticated and authorized. Furthermore, data is encrypted.

To apply Zero-Trust principles to applications, additional effort must be taken beyond traditional Zero Trust. Evaluation is needed to see whether the process is permitted and automation should be extended to include remediation whenever possible. System processes such as APIs that connect microservices need to prove themselves, in much the same way as individuals and devices. Similarly, container management (that determines how applications can communicate within containers) must be evaluated. The system-generated transactions should be valid and the processes allowed to perform the actions they are performing. Whenever exploits are encountered, protections should be automatically triggered. Doing so requires monitoring and protection during runtime.

Design for failure

To protect applications regardless of where they run entails data security (encryption is a traditional part of Zero Trust) but also security wrapped around the application logic that works on the data and the application infrastructure within which it runs. The ability to monitor applications in production, detect exploits, and take action to stop them becomes paramount. Attention must be paid to access at every point along the way; no longer just at the network or endpoints, but also the application and its containers, orchestrators, APIs, and dependencies. Often, teams stop at who can access the code (access controls), but this is not enough—more dimensions are needed to authenticate access, and access alone does not protect against malicious insiders nor viruses that can ride alongside the legitimate user. Applications represent the very logic that legitimately (or illegitimately) alters the data. It's becoming more important than ever that app security be included in a solid security and risk-management program.

One added consideration: this is not just the code in production any longer. In the old days, you could protect the production code by limiting who had access to the production servers. Now you must think about the application in the same way you think about the data—it moves, it has many different points of engagement, and so on. Its environment spins up and spins down. There is no longer a stable, physical box you can protect to secure the applications within. The traditional environments (dev, test, production) that separate access, code quality, and testing become less rigid. Application security programs that rely on traditional division of environments with a rigid flow from one to the other will find it incredibly challenging to secure next-generation applications.

Start with the basics of Zero Trust and build from there

Traditional Zero Trust requires endpoint security, data security, and access controls. A more recent addition to Zero Trust is the concept of secrets management, which we explored earlier in this report. Data security begins with classifying your data to understand what sensitive data you have and where it resides. Acceptable use policies are developed to determine which roles should have access to what data and then Role-Based Access Controls (RBAC) and Multifactor Authentication (MFA) are used to evaluate each data request. Let's look at authentication and encryption a little further.

Authentication. RBAC is a critical element in securing not only serverless applications, but any cloud-based applications. Unlike traditional Access Control Lists (ACLs) that would grant or deny write access to a particular system file, RBAC can control how that file could be changed. It checks roles and privileges of authorized users, validating that a user is who they say they are and whether they have authorization to access the requested resources. Enterprises have been moving away from simple password systems to multistep authentication. After the person is authenticated, they need to pass an authorization check and gain access to different types of information.

Encryption. Increasingly, businesses encrypt information from inception to deletion. Previously, data was encrypted at rest and then in motion, when moving from place to place on the network. With encryption, if the bad guy gets in, the data is useless to them. Given the new entry points, encryption must be more holistic. To effect this change typically requires changes to the code itself that are not easily nor quickly accomplished but rather worked in bit by bit as applications are modernized and updated. Scanners should check that encryption is used.

RBAC and end-to-end encryption are more a set of principles than a piece of software that you can buy. Software can augment the effort, but the principles should be applied to the way software is designed. These things are not as easy as slapping a tool like Auth0 or Micro Focus Voltage in place; they also must be taught and implemented over time. The challenge for the CISO is that most of the application of the principles is beyond the scope of the security team. The best way for the CISO to assist is to set policies for the development team to follow and provide tools to inspect the adherence.

Stepping beyond these more traditional Zero-Trust methods, there are additional elements to consider for cloud native applications. Monitoring, detection, and protection are needed for microservices' APIs/functions, administration of the cloud service, and containers and their orchestrators. Containers, functions, and APIs must be authenticated as automated system "users" in much the same way that a human user must be authenticated. Their credentials must be validated and privileges determined based upon roles or policies. Although only fledgling attempts have been made, eventually machine learning will be applied to security between apps, data, containers, and the cloud, similar to the newer cooperative intelligence between endpoints and the network in which each can provide feedback to the other.

Lastly, autoremediation, also an emerging area of capability, can automate a self-healing response to vulnerabilities and exploits. Automation such as this can greatly improve time to remediation, improve consistency, and provide educational feedback to development to improve coding practices.

These capabilities should be built upon a foundation of standardization, policy automation, control, and CI. Just as it's important to standardize endpoints to ensure consistent behavior, development pipelines can also benefit from standardization. With consistent SDLC tools and pipelines, policies can be consistently applied with more predictable outcomes. Similarly, end-to-end visibility is improved for consistent measurement and process improvement. And finally, consistency simplifies compliance and auditability. We dive deeper into auditability when we discuss the importance of having a secure SDLC.

Final thoughts on Zero Trust

To recap, there are several advantages of a holistic Zero-Trust approach:

- Lateral movement is much more difficult. Each service must be authenticated so the internal network is not permissive.

- Stolen credentials are less valuable. Strong authentication requirements increase the cost of credential theft and man-in-the-middle attacks.
- Known vulnerabilities that are easy to exploit will be rarer resulting in better ecosystem hygiene.
- Nontargeted attacks have less value. This forces targeted attacks to exact a higher cost from the attacker.

Data is the crown jewel and applications are their caretaker. You will never be right 100% of the time. The hacker needs to be right only once. So, expect that you will be hacked. The key to keeping your job and protecting your company is to minimize (or even eliminate) the impact and speed remediation. It's not enough to protect endpoints, data, and access. The applications that apply business logic to the data are critical, as well.

Align with Development Objectives

Software development reflects the business objectives. If the business decides that it needs to create a new business application or overhaul an existing one to better meet competitive threats or to capitalize on a revenue opportunity, security shouldn't be the roadblock. To be the hero, instead partner with development to find ways to take advantage of the broader manpower of development (versus security) and empower them.

No one wants to be "The One": the one who got the company hacked. The one who made a simple but careless mistake like including a password in the code. Developers are certainly no exception to this. Except for nefarious insiders, IT people don't wake up and say, "Hmm, I think I'll create a software vulnerability today, or misconfigure a cloud or container setting, leaving a gaping hole for attackers." Yet it happens every day.

A small focus group of developers validated this point. When asked what most motivated them to develop more secure code, most cited the fear of being "The One." This was a more powerful motivation than job performance objectives, bug bounty costs, or compliance. Yet at the same time, they all felt almost powerless to do anything to avoid this risk to their personal career. They feel very accountable but not empowered to take action. They are generally frustrated by the lack of security assistance available to the developer, whether automated or human expertise.

Security teams could improve this situation by partnering with development around security while placing the developer at the center of the action. After all, it's usually the developer who can remediate the application vulnerability, not the security pro. But how, you might ask? To secure the next generation of software, you will need to take risks by doing things differently (no longer protecting status quo). You must reimagine security in the next generation of software without being bound to methods from the past. Consider the following suggestions:

- As you move to embrace the new, try things in small, controllable environments, measuring carefully before and after.
- Align security tools and processes into the development workflow. Providing actionable insight about security vulnerabilities at the point of code commit is key. Cofunding solutions might be necessary, especially when security is embedded into CI.
- Align motivations. As just mentioned, if business leadership needs incentives to care about security, consider having them fund the bug bounties of vulnerabilities found in their applications.
- Develop security expertise within the development teams so that they have someone to turn to when having difficulty understanding how to remediate a security flaw.
- Developers and software engineers are naturally good at process design. Bring them into the redesign of your application security process.
- Embrace standardization, often disguised as reusable components. This may take the form of standard pipelines, standard containers, and standard code modules.

Embrace open source

From a security perspective, open source code has the advantage that more people use it, test it, and improve it to remove vulnerabilities. The downside is that because it is readily accessible, hackers can also evaluate it and look for exploits. In addition, open source code is an attractive target because of the number of users. The targets and blast radius can be extensive.

Yet repeatability is good. It brings consistency. With reuse, when one vulnerable library is remediated, the effort can fix many vulnerable applications. Having a Bill of Materials (or inventory of where dependencies reside) capability can help you to measure the risks and prioritize remediation efforts. Another related approach would be to have a pretested, preapproved list of third-party code.

Secure the SDLC

In most cases, a breach is usually a case of poor hygiene, like not applying a security patch or ignoring a critical software vulnerability. Sometimes, it's due to ignorance or misinformation like believing that applications run in containers or on cloud services are inherently more secure.

The best way to reduce application security risk from ignorance and neglect is through automation. By automating application security scanning, vulnerability remediation, and monitoring the application's infrastructure, the element of human error is removed and consistency applied. Auditors love the approach because automation enforces security policies while exceptions can be easily identified and documented.

In fact, compliance is essentially about ensuring the integrity of the software development process/pipeline. The best way to reduce risk of noncompliance with regulatory controls is by automating the controls and having traceability along the way to capture who changed what, when, and why. Building this automation into the SDLC can protect against not only careless mistakes and mistakes due to increased complexity, but also insider threats and external exploits alike. Securing the SDLC requires the following:

- Basic security hygiene
    - Scan every code change for at least the OWASP Top 10, completing all of the following types of scanning: static, dynamic, dependencies, container, and images
    - Scan code for license compliance
- Complying with compulsory industry regulations
- Monitoring, detecting, and automating response

- Building on standardization, policy automation, validation, common controls, and CI

Hygiene and compliance are foundational. Standardization ensures consistent application of policies. By following common controls like separation of duties and ensuring a clear audit trail of who changed what, risks of inside attacks and human error can be mitigated.

With a secure and consistent software factory, or SDLC, software can be developed with fewer security flaws and deployed in an environment that is monitored for security incidents. By using these prudent efforts, you can manage risk within your tolerance levels.

Now that we've looked at pragmatic steps that you can take to secure next-generation software, let's wrap up with 10 key principles that you can apply to your security program.

Conclusion

Next-generation software is different and so is the process to develop and deploy it. Securing it brings a new set of challenges and requires more focus on not only delivering secure applications but securing them in a new infrastructure and securing their SDLC.

Security Considerations

End-to-end application security needs to automate the following:

- Application security testing and remediation
- Production application protection
- Policy compliance and auditability
- SDLC platform security

DevOps and modern software development embrace iteration instead of programming an application to do an entire function all in one go. You bite off a much smaller piece of that elephant and do what can fit within a sprint, whether the sprint is two weeks, a month, or a day. You rely on microservices wherever possible to reduce investment and speed time-to-results. The guiding premise is this: what can I do today that will add business value? I can make that one small change and get it out the door (minimal viable change). How I test that small change is infinitely different than how I test a much larger change. It must be iterative and it needs to be in the hands of the one that can modify the code. Inspection for policies and requirements must become automated to be applied consistently and quickly as a natural part of the process.

10 Steps to Secure Next-Generation Software

For the CISO, this means:

1. Start with "you are here" approach and pick your path forward. Invest in dependency scanning, container, and cloud infrastructure security. Apply SAST and DAST to every new code.
2. Rethink security as an outcome, not a department. Align metrics. Manage the risks, not the silos. Finding more vulnerabilities might be less important than fixing the ones that are known.
3. The tools and the processes absolutely must go hand-in-hand. Start with the process and then apply the tools that will help you achieve the desired outcome.
4. Go broad, not deep, when testing software. Test every change, at least for most common security vulnerabilities, rather than narrowly focusing on "critical" apps. Everything is a weak link now.
5. Unite the workflow applying continuous security scanning with iterative development for Continuous Application Security-like bites of the elephant.
6. Test for security flaws *at the point of code commit*, to break down the work into smaller, actionable scanning and remediation cycles.
7. Automate, allowing security to focus on exceptions.
8. Align with development objectives. Embrace open source. Align security testing to the developer's workflow. Standardize pipelines, code, and more for consistent, predictable results.
9. Apply Zero-Trust principles to applications and their infrastructure. Work from the familiar perimeters and endpoints that protect the data, to the apps that apply business logic to the data, to the users that interface with the data.

10. Protect the integrity of the software development and delivery process by ensuring proper audit controls are in place and seamless across the SDLC.

Software development is changing rapidly, and security programs must also evolve if they are to be effective in this next generation of software. Security implications result from changes to how software is composed and executed, the methodologies by which it is developed, and the infrastructure surrounding its use.

My wish is that security leaders will apply this new, deeper understanding of the next-generation software evolution to become the change agent that helps their enterprise deliver software faster as well as more securely.

About the Author

Cindy Blake is the senior security evangelist at GitLab, the fastest-growing single application for the entire DevOps life cycle. Targeting rapidly evolving DevSecOps initiatives, Cindy Blake collaborates around best practices for integrated application security solutions with major enterprises. With nearly a decade of cybersecurity experience, Blake provides leadership and guidance to GitLab product teams, marketing, and sales to facilitate growth and bring maximum value to GitLab customers.