

A Blueprint for Supply Chain Security

What You Need to Protect
Containerized Applications

The New Stack

A Blueprint for Supply Chain Security

What You Need to Protect Containerized Applications

Alex Williams, Founder and Publisher

Ebook Team:

Andrew Tillery, Digital Marketing Manager

B. Cameron Gain, Author

Ben Kubany, Project Manager

Colleen Coll, Sponsor Advocate

Danni White, SEO Editor

Diana Gonçalves Osterfeld, Designer

Heather Joslyn, Ebook Editor

Judy Williams, Copy Editor

Supporting Team:

Benjamin Ball, Director of Sales and Account Management

Joab Jackson, Editor-in-Chief

Michelle Maher, Assistant Editor

Vinay Shastry, Director of DevOps

© 2022 The New Stack. All rights reserved.

20221004.2

Table of Contents

Sponsor.....4

Introduction5

What Is Software Supply Chain Security?9

The Threat Landscape13

Getting Started With Supply Chain Security20

A Best Practices Checklist34

Conclusion: Building on the Security Blueprint.....38

About the Author 41

Disclosure42

Sponsor

We are grateful for the support of our ebook sponsor:



Red Hat Advanced Cluster Security for Kubernetes — the first Kubernetes-native security platform — enables organizations to securely build, deploy and run cloud native applications anywhere.

Introduction

In the cloud native era, properly securing the software supply chain starts at the very beginning of the development process and continues throughout production and the entire application life cycle. The tradition of implementing security tests at the end of the development and production process or patching running applications is outmoded.

Improper security implementation can seriously impact business by delaying important releases in order to address issues found later in the software life cycle or by losing security fixes that were only applied to running workloads.

Just as automation is key for Kubernetes, it's also critical for the software supply chain. Proper security is continuous, with security gates implemented throughout the build, deploy and runtime process. Proper security allows developers to do their work with guidance from security policies implemented with automated tools that analyze, monitor and intervene when things go awry.

And yet, this is all so much easier said than done.

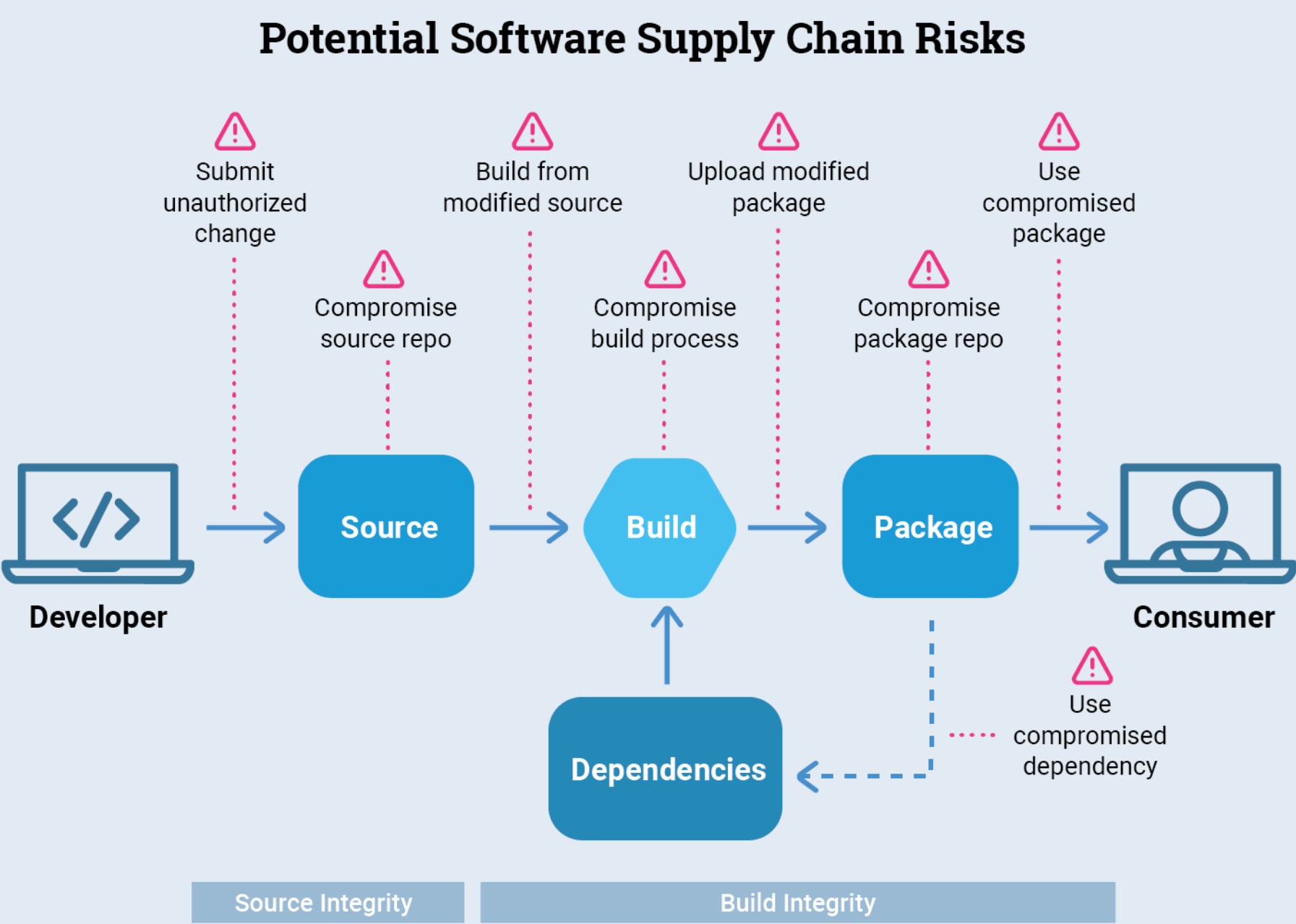
Most DevOps and SecOps teams have yet to implement current best practices to keep their software safe from build through production. Organizations are hampered by a proliferation of continuous integration/continuous delivery (CI/CD) tools throughout the organization, with many customizations, a frequent lack of automated processes, and the snowball effect of rampant software vulnerabilities coupled with ever more sophisticated attacks. Many organizations struggle to figure out where and how to start.

According to Red Hat’s “2022 Global Tech Outlook” report, [46% of respondents put IT security as their top funding priority](#). However, security is not a static state, especially in a distributed computing environment with containerized applications.

The needs in such a network shift constantly. The attack surface changes rapidly and vulnerabilities emerge across repositories where code is sourced and extends across the development and deployment cycle. Compliance demands vary.

Security teams are responsible for identifying security and hardening requirements, and for providing guidance to developers for securing code across the supply chain — including when and how source code, packages and binaries are sourced and stored throughout the CI/CD process.

Fig 0.1 Vulnerabilities can occur at all stages of the software life cycle.



“Many organizations struggle with figuring out where and how to start with software supply chain security.

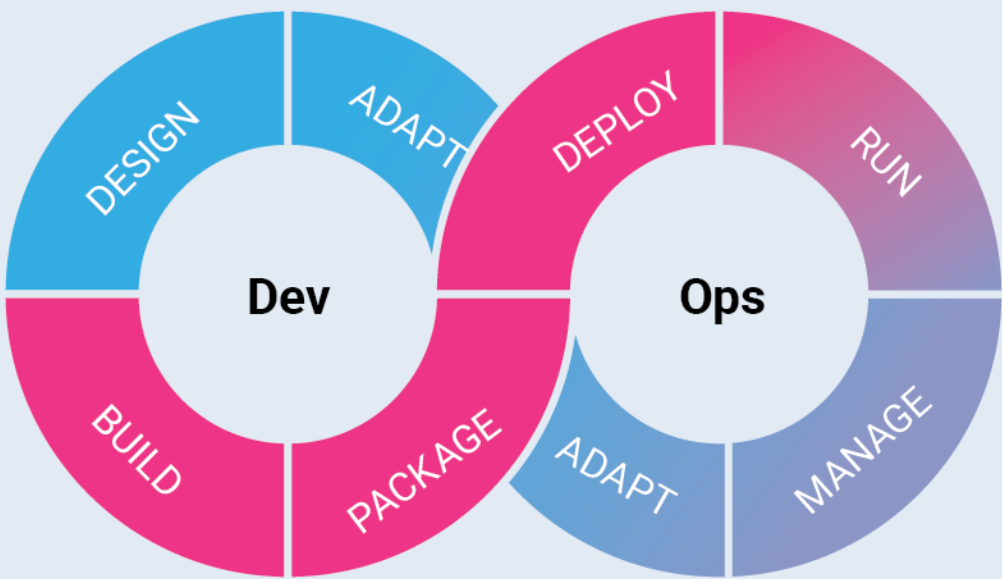
Security teams are also responsible for monitoring platforms and ensuring that running applications comply with security and hardening requirements. Meanwhile, the attackers and landscape are constantly shifting, and security teams must ensure their security tools and processes are able to meet the evolving threats.

Security teams, therefore, are mandated to lock down their code on all fronts as much as possible. But this is not necessarily the main goal of the developers or operations engineers.

While developers understand the importance of security, they are often rewarded for adhering to the fastest code delivery schedules and for creating business value by improving the customer experience. They’re not explicitly mandated to collaborate with the security team at the cost of short-term development velocity.

Fig. 0.2 *To remain secure, containerized applications running on Kubernetes demand a virtuous circle of building, shipping, gathering feedback, rebuilding and hardening.*

Containers and Kubernetes Need DevSecOps



Control Application Security



Protect the Platform



Detect & Respond to Runtime Threats

Security, meanwhile, can be seen as a hindrance, especially when a security test or container scan detects a vulnerability late in the production process and the application is sent back to the developer for rework, delaying production releases.

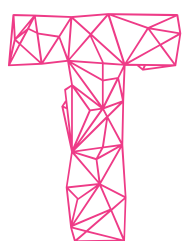
But when automated tools and best practices are properly implemented following DevSecOps principles, these competing agendas can disappear. Code is vetted, scanned and checked continuously, without slowing down the delivery of software created to improve the customer experience. The earlier an issue is discovered in the development process, the easier it is to fix.

When implemented properly, DevSecOps becomes a closed loop system. Proper implementation involves a “shift left” approach to security. This gives developers more responsibility for securing the applications they build by identifying issues and correcting them within their workflows, and automatically providing feedback on issues discovered in production back to the developer team to address.

In this ebook, we provide a framework of the threat landscape and what you need at every stage to create a trusted software supply chain. It is geared toward organizations developing, deploying and managing applications in cloud native environments.

CHAPTER 01

What Is Software Supply Chain Security?

 here are many definitions of software supply chain security. For the purposes of this ebook, supply chain security covers all aspects of the development and deployment of software, including all components and code libraries used to build or update an application. In addition, due to the immutable and declarative nature of containers and Kubernetes, runtime security may also fall within software supply chain security. Lastly, all related systems and tools for the entire process are critical to the security of the software supply chain.

Those tools include security functions that are largely used for testing, alerting and observability. They detect issues as well as platform and application hardening capabilities. Protection is provided through admission controllers, encryption, features that isolate clusters or containers if they've been compromised, and other features.

Software supply chain security needs to continue beyond the CI/CD stage. Security tools and best practices should cover the runtime environment once software is deployed.

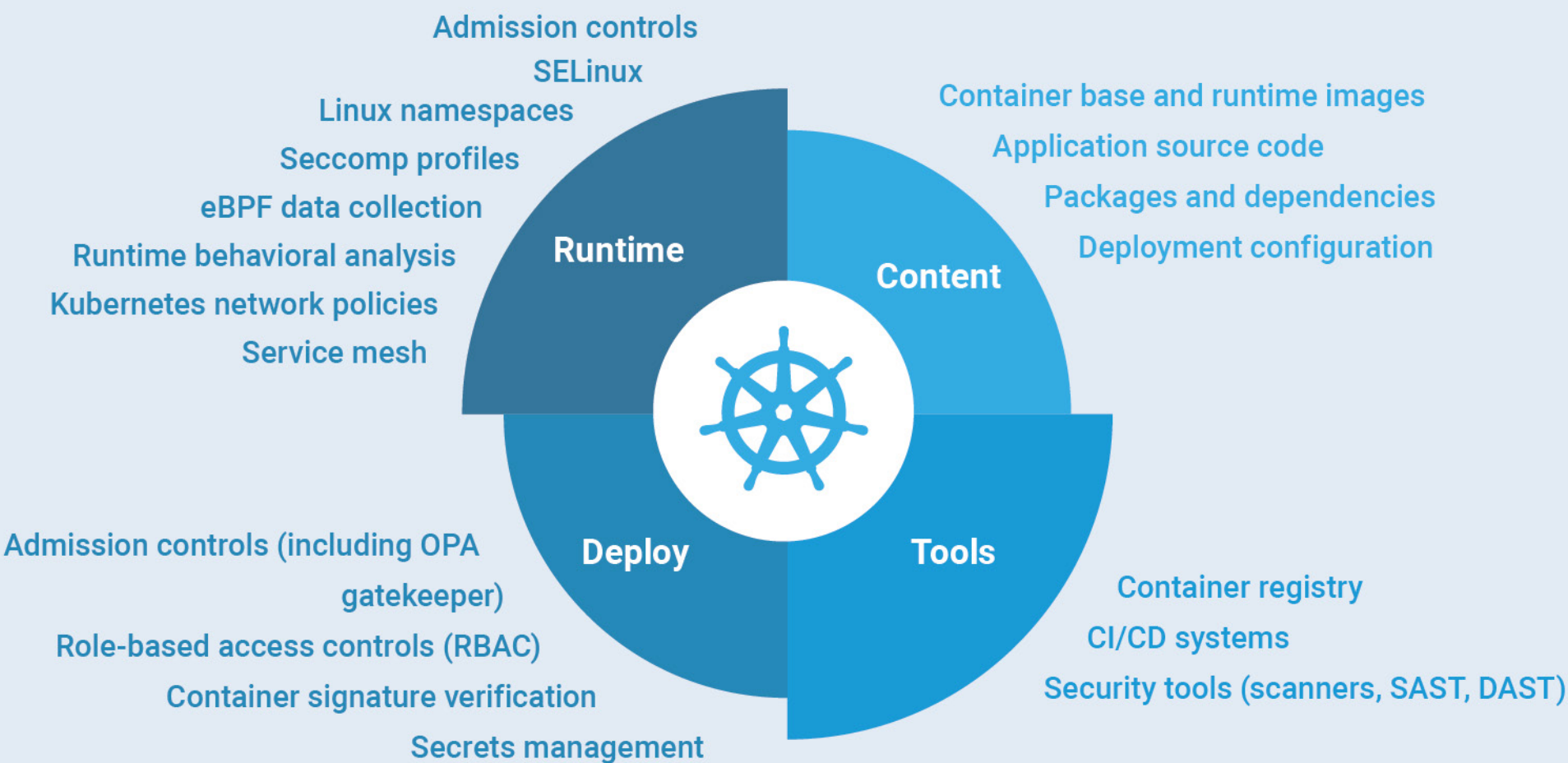
Automation of the container CI/CD process is especially important for runtime security. While it is possible to patch a running container to address newly discovered vulnerabilities, doing so in a Kubernetes environment is not best practice. Kubernetes continuously monitors the actual state of running applications against the declared state.

“The momentum on an industry-wide scale for supply chain security continues to build, due to the growing awareness that best practices are needed to help secure open source code.

For example, if you define for Kubernetes that three instances of an application should be running at all times and one instance goes down, Kubernetes will deploy a new instance from the container image. If you only patched the running instance of the app, the patch is then lost when it’s replaced by the new instance. Instead, the vulnerable code must be fixed at the build stage. When the container image is rebuilt, then the new, patched image can be deployed. This is one example of how the runtime environment is critical to the supply chain security.

Fig 1.1 What the software supply chain consists of in a Kubernetes-run environment.

Kubernetes Software Supply Chain at Glance



Now's a good time to define some other oft-misunderstood terms we'll use in this ebook:

Cloud Native: These are the technologies that let organizations develop and deploy applications on cloud environments, including private, public and hybrid clouds. For our purposes, they consist of containerized environments with microservices running on Kubernetes, usually with a declarative API approach.

Infrastructure as Code: Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

Zero Trust: [Zero trust](#) is a strategic approach to security designed for distributed networks. It differs from the old “moat and castle” approach in that it focuses on specific valuable assets within a system — rather than a static “perimeter” — that require protection, and grants the least possible access to them.

The momentum on an industry-wide scale for supply chain security continues to build, prompted by the growing awareness of the need for best practices to help secure open source code. Since 70%–90% of any software stack consists of open source software, the risk of shared vulnerabilities is balanced by the significant benefits open source offers the community, according to The Linux Foundation and the Open Source Security Foundation (OpenSSF).

The Linux Foundation and the OpenSSF [have established goals](#) to secure the open source supply chain. These emerging standards and best practices — which we'll cover later — include securing open source production, improving vulnerability discovery and remediation, and shortening the response times for patching..

Implementing a software supply chain security program is a momentous task. Even

once the right skills are in place, with the proper cultural support so that developers, operations and security teams are working in tandem to create and test code, it can typically take years for an organization’s supply chain security integration to mature. That’s why it’s so important to get started now.



SUMMARY

Red Hat Advanced Cluster Security for Kubernetes — the first Kubernetes-native security platform — enables organizations to securely build, deploy and run cloud native applications anywhere.

KEY FEATURES

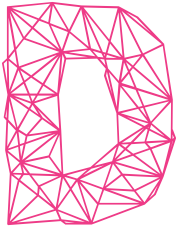
Supply chain security: Extend security throughout the software supply chain, starting with automated security scanning and compliance assurance in the development phase (DevSecOps).

Infrastructure security: Leverage built-in Kubernetes security posture management (KSPM) to identify and remediate risky configurations and vulnerabilities.

Workload security: Provide better security for workloads by maintaining and enforcing a zero trust execution approach to cloud workload protection (CWP).

CHAPTER 02

The Threat Landscape



DevOps teams are well aware of the ubiquity of code vulnerabilities that find their way into the development pipeline and in production — but how attackers are exploiting bad code is getting worse.

Supply chain attacks have proliferated in this young decade (notably the [SUNBURST malware attack on the networking tools vendor SolarWinds](#), in December 2020), with no signs of slowing down.

The rise in ransomware and other malicious attacks [prompted the Biden administration to issue an executive order in May 2021](#), stating that the U.S. federal government must adopt best practices (including zero trust architecture), accelerate securing the cloud services it uses, and centralize and streamline access to cybersecurity data to help agencies identify and manage cybersecurity risks.

The order applied not only to federal agencies but also to cloud providers and IT service providers that do business with the federal government. In March 2022, the National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA) [released a report offering guidance on hardening Kubernetes security](#).

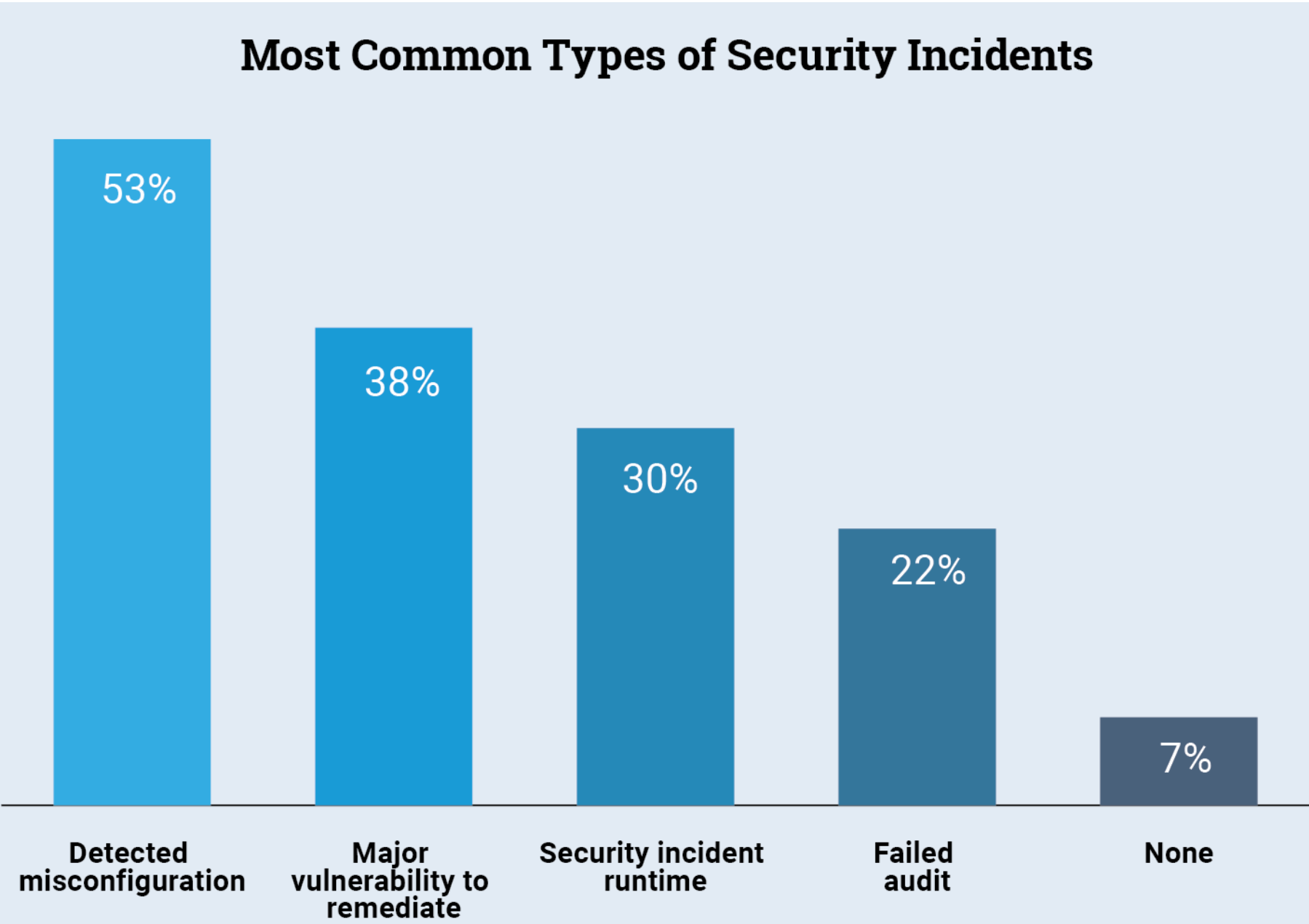
The NSA followed up its Kubernetes security recommendations in September 2022 by issuing [guidelines targeted specifically at supply chain security for developers](#). In its guide, the NSA detailed best practices for what it called “actionable

recommendations” for the development, production, distribution and management processes of supply chain security.

There is an abundance of data that demonstrates the threat landscape:

- By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021, according to [Gartner](#).
- [In a Red Hat survey](#) of 300 DevOps, engineering, and security professionals, [31% of respondents](#) reported their organization has experienced revenue or customer loss due to a security incident over the last 12 months.
- For Kubernetes environments, 93% of respondents in the Red Hat survey experienced at least one security incident in their Kubernetes environments in the last 12 months, sometimes leading to revenue or customer loss.

Fig 2.1 *Where the software supply chain is most vulnerable. (Survey participants could choose more than one type of incident that occurred in the previous 12 months.)*



- In the same report, Red Hat noted how Kubernetes and containers were designed for developer productivity. Meanwhile, default pod-to-pod network settings allow open communication for rapid cluster and node deployments, at the expense of security hardening.

Not surprisingly, nearly 53% of participants in the Red Hat survey reported a misconfiguration incident in their environments over the last 12 months. Thirty-eight percent said they have discovered a major vulnerability and 30% said they have suffered a runtime security incident. Lastly, 22% said they had failed an audit.
- Supply chain attacks can be devastating. In the event of a breach, the most common types of repercussions due to cloud misconfigurations are data compromises and the introduction of malware (38%), including crypto miners (37%) and ransomware (30%), according to a 2021 [Enterprise Research Group \(ESG\) study](#) based on the responses of 383 IT and cybersecurity professionals.

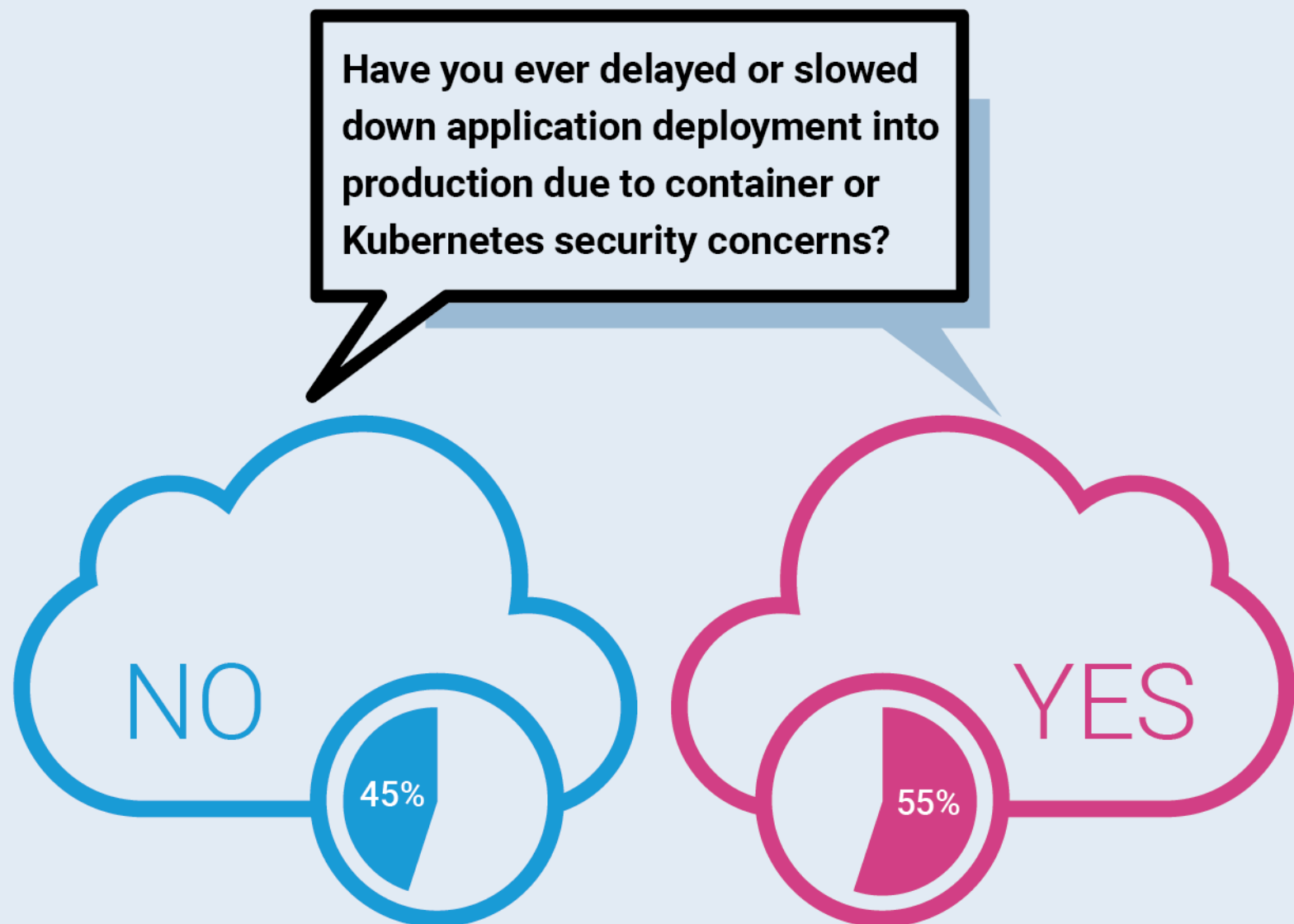
Fig 2.2 Misconfigurations can open to the door to security incidents. (Survey participants could choose more than one result.)

The Result of Misconfigurations During the Past 12 Months

Unauthorized access to application and data	40%
Remediation steps impacted service level agreements (SLAs)	39%
The introduction of malware	38%
The introduction of crypto-jacking malware to mine cryptocurrency	37%
The introduction of ransomware	30%
Fine incurred due to noncompliance with an industry regulation	30%
Data was lost	25%

Percent of respondents, N=350, multiple responses accepted

The Productivity Slowdowns That Security Concerns Bring



Source: "2022 State of Kubernetes Security Report," Red Hat.

© 2022 **THE NEW STACK**

Fig. 2.3 Security issues can slow down application deployment.

Guarding Against Human Error

Television and movies might lead us to believe that hackers are usually lone geniuses who crack an encrypted code or password (numerous automated tools exist on the Darknet for this alone). Think of all the scenes Hollywood has shown us, in which the hoodie-wearing Lone Genius suddenly announces their security breach with the words, "I'm in!"

But the truth is, the majority of attacks aren't the work of genius. Attackers simply [use the front door](#) that's left open by a system administrator.

In fact, human error continues to lead the causes of data breaches, accounting for 95% of all breaches, according to a [World Economic Forum report](#). Successful attacks

result from the access and often — if not most of the time — exploitation of vulnerabilities that remain unsecured in the software supply chain.

They involve, for example, a hacker grabbing inadvertently saved secrets in a GitHub repository — and especially, access gleaned through compromised passwords from phishing attacks.

Compounding the security challenge, [almost all \(99%\) of identity access management \(IAM\) policies are overly permissive](#), according to an April 2022 report of security threats to cloud infrastructure by Unit 42, the threat research arm of [Palo Alto Networks](#). Too many unused credentials present easy targets for hackers.

“Human error accounts for 95% of all security breaches, according to a World Economic Forum study.

The most common types of cloud misconfigurations are due to a lack of IAM basics, such as the use of default passwords (30%) and lack of multifactor authentication (22%), according to the 2021 ESG study mentioned previously.

“These join other misconfigurations reported by respondents, such as externally facing workloads subject to port scanning, overly permissive accounts targeted by bad actors and unauthorized access to services via open ports,” the report’s authors wrote.

Attackers obviously seek to exploit vulnerabilities anywhere they can throughout the entire software supply chain. These attack vectors include repositories pulled or forked from resources such as GitHub and libraries that are often used for much of the base code in software development, the CI/CD production pipeline itself, and the runtime code distributed across the containerized environment. A comprehensive

supply chain software strategy must cover and apply to these three main factors.

Gartner describes the threat landscape in a [2021 report](#) as such:

- The increased threats of malicious code injection makes it critical to protect internal code and external dependencies (both open source and commercial).
- Leaked secrets or other sensitive data and code tampering prior to release are consequences of a compromised software build and delivery pipeline.
- Failure to enforce least privilege entitlements and flat network architectures enables attackers to move laterally against the operating environment, putting the enterprise at greater risk.

However, the good news is that there are immediate steps you can take to start locking down the supply chain and protect applications and code.

These best practices include the automation of:

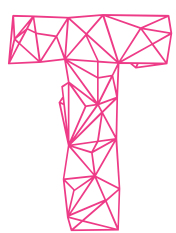
- Analysis of code from all sources before the development process even begins, with [a software bill of materials \(SBOM\)](#) for each code source. The SBOM should come with any code that is used. For code and application distribution, open source alternatives such as the [Software Package Data Exchange \(SPDX\)](#) serve to communicate SBOMs.
- Adherence to the [supply-chain levels for software artifacts, or SLSA](#) (pronounced “salsa”).
- The integration of policy checks through CI/CD and in runtime.
- Container image scanning before an image is pulled into the build and before deployment.

- Signing tools, such as [Sigstore](#), for signing images and configuration files.
- Continuous audit and logging for each step in the supply chain.
- Role-based access and authorization control for all tools used in the supply chain and in production, including namespace and other access to Kubernetes clusters.
- Collection and analysis of platform audit logs, and application logging and monitoring for potential security issues.
- Runtime behavioral security analysis for detection of anomalous behavior.
- Processes that ensure that information discovered during development informs deployment and production policies, and that information discovered in production is fed back into the development process.

Let's look at what you need to lay the foundation for a trusted software supply chain.

CHAPTER 03

Getting Started With Supply Chain Security



he “shift left” trend is generally thought to apply to the very beginning of the development cycle, at the point when developers first begin to create code and make pull requests. However, for a trusted software supply chain, the work needs to begin in the open source projects that are the building blocks for custom applications.

SBOMs and SLSA have emerged as key components in any holistic supply chain security blueprint. Both provide different yet complementary functionality, involving the application of shift-left security to an even wider degree. This is because SBOMs and SLSA, when used together, cover the history of all libraries, including external dependencies, used in the development of an application.

Here’s how each helps secure the software supply chain.

What Are SBOMs?

The information that an SBOM provides about an application, a code library or a third-party vendor’s software is often compared to the lists of ingredients and additives listed on a food label. An SBOM provides a library of all libraries used in an application, with a history of all code sources and timelines.

SBOMs are seeing rapid adoption among organizations that create software or use third-party software tools and platforms, regardless of where they are in their supply chain and DevSecOps journey.

The vast majority of software-composition analysis tools (90%) will be able to generate and verify SBOMs to help inform consumers of open source software by 2024, compared to 30% in 2022, [according to Gartner](#). By 2025, most organizations (60%) that create or procure “critical infrastructure software” will mandate and standardize SBOMs in their software engineering practice, compared to less than 20% in 2022, the analyst projected.

President Biden’s May 2021 Executive Order on Cybersecurity, which mandated that software provided to the federal government must include an SBOM, and [the discovery of the Apache Log4j’s Log4Shell vulnerability later that year](#), have helped to raise awareness about SBOMs.

After the Log4Shell discovery — which will likely pose problems for the software supply chain for years to come — DevOps teams became more aware of how SBOMs can help an organization build an inventory of software packages and libraries deployed in their environment, thereby simplifying the process of identifying the presence of vulnerabilities like Log4Shell.

SBOMs are still missing from many organizations’ supply chain security practices. Many readers of this ebook have downloaded or forked applications freely available on GitHub, GitLab, Bitbucket or other repositories. Some may have used an Infrastructure as Code (IaC) template without proper security checks, while assuming commercial repositories offer security checks by default.

But using code directly from any repository is very much like the risk of plugging a USB thumb drive that you find laying on the ground into your PC or server, as [Dan Lorenc](#), co-creator of SLSA as well as Sigstore security projects, has long noted.

“Everybody knows you don’t bring code inside that you grabbed from a random repository, but I saw it happening hundreds of times a day a few years ago,” Lorenc, co-founder and CEO of the software security company Chainguard, told The New Stack. “We started asking questions and hearing about SBOMs more often, but interest kind of stalled out. And then it started to pick up again with the SolarWinds attack, which was a big turning point.”

While SBOMS have been available for a few years and are being rapidly adopted, they are still a work in progress. For example, SBOMs can be limited in scope. There are also different, competing standards, such as [CycloneDx](#), [SPDX](#) and [SWID Tag](#).

Remember that SBOMs are but one part in a trusted software supply chain and should be used with the SLSA framework, which calls for the use of SBOMs and is designed to help organizations improve the integrity of their supply chains.

What Is SLSA?

The SLSA framework was designed for organizations to make use of its guidance gradually. It is designed to be adopted in incremental steps, all of which offer tangible improvement for software supply chain security along the way. SLSA includes four levels, or “milestones,” as they are called in the documentation, that organizations can attain.

The 4 Levels of the SLSA Framework

Level 0

No guarantees. SLSA level 0 represents the lack of any SLSA level.

Level 1

Documentation of the build process

The build process must be fully scripted/automated and generate provenance. Provenance is information that allows you to verify the origin of software. SLSA provenance includes metadata about how an artifact was built, including the build process, top-level source and dependencies.

Knowing the provenance allows software consumers to make risk-based security decisions based on information about the source of the software: e.g., does the software come from a known and trusted source? Provenance at SLSA level 1 does not protect against tampering, but it offers a basic level of code-source identification that can aid in vulnerability management.

Level 2

Tamper resistance of the build services

Requires using version control and a hosted build service that generates authenticated provenance. These additional requirements give the software consumer greater confidence in the origin of the software. At this level, the provenance prevents tampering to the extent that the build service is trusted. SLSA level 2 also provides an easy upgrade path to SLSA level 3.

Level 3

Extra resistance to specific threats

The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance, respectively. The creators of SLSA envision an accreditation process whereby auditors certify that platforms meet the requirements, which consumers can then rely on. SLSA level 3 provides much stronger protections against tampering than earlier levels by preventing specific classes of threats, such as cross-build contamination.

Level 4

Highest levels of confidence and trust

Requires a two-person review of all changes and a hermetic, reproducible build process. Two-person review is an industry best practice for catching mistakes and deterring bad behavior. Hermetic builds guarantee that the provenance’s list of dependencies is complete. Reproducible builds, though not strictly required, provide many auditability and reliability benefits. Overall, SLSA level 4 gives the consumer a high degree of confidence that the software has not been tampered with.

Sources: SLSA.dev and Red Hat

Fig 3.1 What SLSA requires at each level.

While SBOMs and SLSA are critical for a trusted software supply chain, they are two components among many. They are not a solution, but represent an ongoing process for the transparency of code provenance and associated libraries, regardless of where an organization is in their journey

“You can’t just buy something, plug it in and solve everything,” said Lorenc. “When you break the supply chain problem down, SLSA is focused on one area and SBOM another. There are a couple of dozen problems that you’ve got to solve and SBOM only targets one of them.”

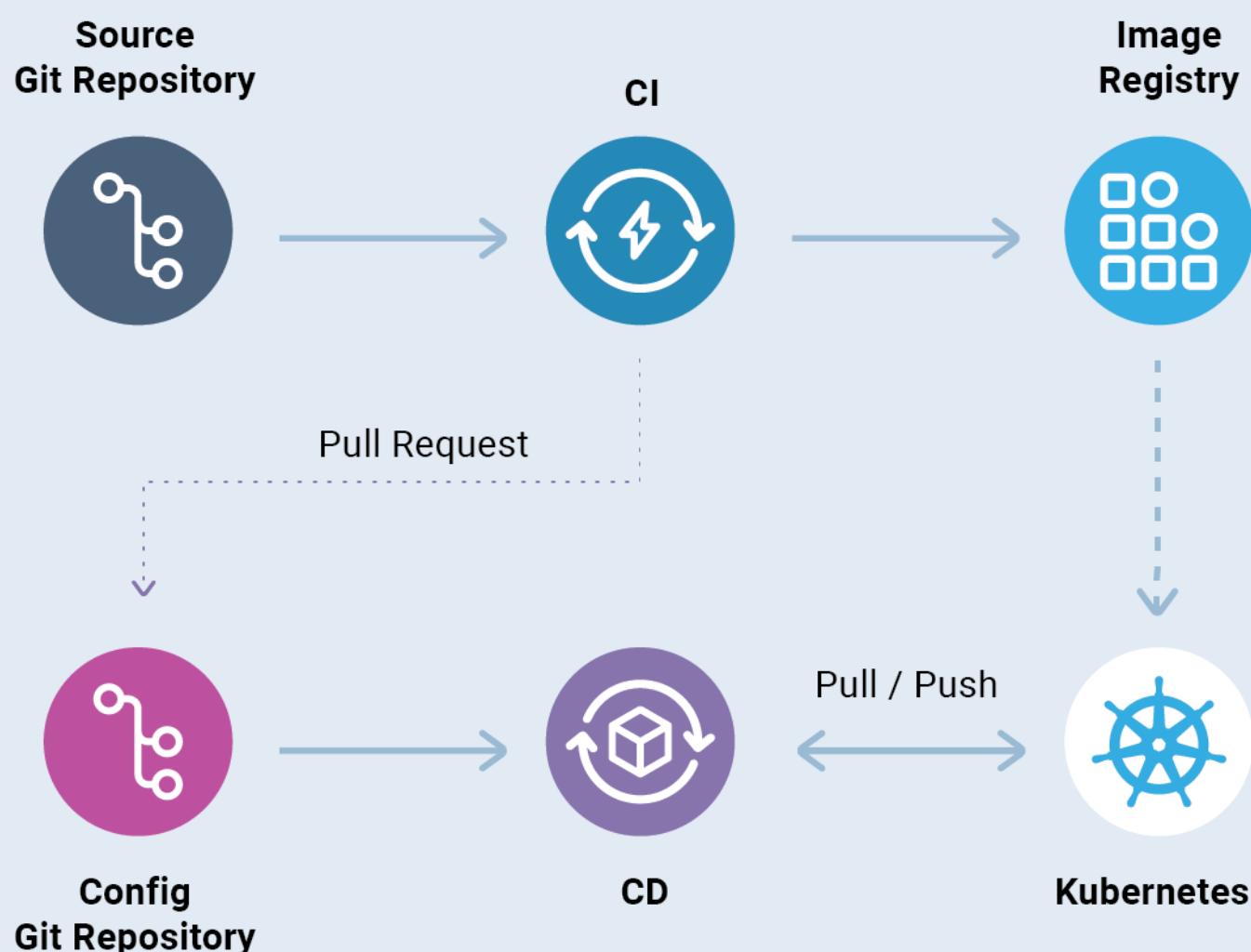
What Is GitOps?

Git is widely accepted as the standard version-control system for DevOps teams. It serves as the central repository for everything CI/CD, covering pull and merge requests, commits and forking. Git repositories are often located on [Bitbucket](#), [GitHub](#), [GitLab](#) and other Git providers.

[GitOps](#) — a set of practices that are complementary to and evolved from DevOps — has emerged as an essential component in supply chain security best practices for cloud native environments. It serves as what [Alexis Richardson](#), CEO and co-founder of Weaveworks — who is widely credited with coining the term “GitOps” — calls a “single source of truth” for declarative infrastructure and applications.

GitOps has proven useful in cloud native environments for managing Infrastructure as Code, for managing and applying cluster configuration, as well as for automating deployment of containerized applications and their configurations to distributed systems.

The GitOps Application Delivery Model



Source: Red Hat

© 2022 THE NEW STACK

Fig 3.2 *GitOps uses Git to make CI/CD more secure.*

GitOps is supported by leading open source tools such as [Argo CD](#) and [Flux](#). These tools help provide much needed automation, monitoring and security for this automated deployment process. In other words, GitOps is essential for the demands of supply chain security in cloud native environments.

Again, automation is key. What security team members should not be doing is manually configuring hundreds of YAML files to rollback a deployment on a cluster in Git. Nor should they ever manually apply changes during runtime directly in the nodes with `kubectl edit`. Instead, a GitOps platform should automate these supply chain related tasks.

GitOps processes also help to ensure that information discovered during development informs deployment and production policies, and that information discovered in production is fed back into the development process.

“In cloud native environments, GitOps can help manage Infrastructure as Code, manage and apply cluster configuration, and automate deployment of containerized applications and their configurations to distributed systems.

For example, security issues discovered in running applications will need to be remediated in the build process, and declaratively applied henceforth. This helps to ensure that an attacker or an admin cannot make changes directly on the cluster, since GitOps uses Git commits as the single source of truth, determining what is running in the cluster.

Conversely, if bad code was introduced during CD and that code has been deployed, the audit trail on Git allows for the deployed code to be rolled back using the familiar revert command in Git, replacing it with the previously deployed version without the vulnerability.

Here are some more examples of how GitOps helps protect and support supply chain security:

Consistency

As the “single source of truth,” the Git repository contains the application configuration definitions as code, reflecting the state of what applications or Kubernetes cluster configurations should be. The desired state of applications on Git are pulled automatically and are continuously monitored and reconciled with the running applications.

GitOps tools, such as Argo CD, continuously monitor application definitions and

configurations defined in a Git repository, and compare the specified state of these configurations with their live state on the cluster. Argo CD detects any configurations that deviate from their specified state. Administrators can choose to automatically resync configurations to the defined state or to be notified of the difference for manual remediation. Again, Git always serves as the single source of truth.

The Audit Trail

GitOps tools, such as Argo CD's continuous reconciliation of deployed configurations, are largely made possible by the audit trail of all previous versions of an application in the repository. The audit trail makes all deployed applications and code traceable. With all previously deployed configurations available on the Git repository, any version can be redeployed when changes in the cluster must be rolled back.

Role-Based Access Control (RBAC)

Declaratively defining access control for all tools used in the supply chain is critical. GitOps tools, such as Argo CD, include support for controlling access to their resources using RBAC. Configuring access control is instrumental in applying zero trust practices.

Managing Communication Between Services

Kubernetes' design enables connectivity between pods, including pods that provide microservices. By default, all Kubernetes pods can communicate with all other pods in a cluster. This runs counter to the concepts of least privilege and zero trust and

means that accessing secrets or exploiting a vulnerability in a single container has the potential to impact the entire cluster.

Kubernetes network policies are available to enable microsegmentation of the software defined network (SDN) that pods use to communicate with each other.

Kubernetes network policies provide a way to specify which pods can communicate with which other pods in a cluster, including what ports are allowed and the allowed direction of communication.

Zero trust for Kubernetes environments can include configuring network policies to deny all connections by default. However, blocking all communication between pods may mean applications running on the cluster will not function.

As part of the CI/CD process, application development teams need to work with the network security team to ensure that network policies are configured in a way that allows applications to function, but also meets security requirements. Kubernetes network policies are written in YAML, which means they can also be managed as code with a GitOps approach.

What Is Policy as Code?

Policy as Code takes the same approach described above for Infrastructure and Configuration as Code and applies that to security policies. Policy as Code can be applied to the CI/CD pipeline to help prevent compromised code from entering the supply chain during the CI/CD process.

Managing Policy as Code can help to ensure that information discovered during

development informs deployment and production policies, and that information discovered in production is fed back into the development process. Just as there are tools, such as ArgoCD, to implement GitOps practices for Application Deployment as Code, below are some tools that can be used to manage Policy as Code at build, deploy and runtime.

Pipeline Policies as Code

Kubernetes-native pipelines, such as [Tekton](#), make it easy to manage all the steps in your Pipeline as Code. Individual pipeline tasks, including security tasks, can be stored as code, as well as the full pipeline itself.

For example, security tasks that should be included in a pipeline are static source-code analysis tools that look for common coding flaws that can be leveraged by hackers; image scanning for known vulnerabilities or other misconfigurations; scans of application configuration code such as deployment YAML; and Helm charts for excess privileges or other misconfigurations.

“Managing Policy as Code can help to ensure that information discovered during development informs deployment and production policies, and that information discovered in production is fed back into the development process.”

Deployment Policy as Code and Policy Engines

[Open Policy Agent \(OPA\)](#) has emerged as a popular [Cloud Native Computing Foundation \(CNCF\)](#) project used for writing and applying Policy as Code. There are a variety of tools available that incorporate OPA for policy management throughout

the entire stack and/or the application life cycle.

For example, [Conftest](#) uses Rego, the language from OPA, to write rules for evaluating Kubernetes configurations, Tekton pipeline definitions, Terraform code, Serverless configs and more.

OPA Gatekeeper works as a Kubernetes admission controller to apply policies written in Rego at deploy time. OPA Gatekeeper can supplement the native Kubernetes PodSecurity Admission controller with more complex admission policies.

[Falco](#) and [StackRox](#) are other open source policy engines, tightly focused on implementing and managing security Policy as Code. Both Falco and StackRox have security rules designed for Kubernetes.

Falco is focused on detecting security violations at runtime. Falco provides several out-of-the-box checks for unusual behavior that may indicate an active security threat, such as unexpected privilege escalation, with the option for triggering an alert.

StackRox policies can be applied at build, deploy and runtime; the same policy can be applied to all three stages. For example, if you use a policy to flag images with known critical vulnerabilities during the CI/CD process, that policy can be used to warn or to break the build. The same policy can be used at deploy time to prevent admission of images with known critical vulnerabilities. And the same policy can be used at runtime to alert about deployed images with newly discovered critical vulnerabilities.

StackRox also provides CI/CD integration to automate build-time security checks, such as scanning images for vulnerabilities. StackRox's deploy-time checks can help

ensure compliance with configuration best practices and industry standard, while runtime threat-detection capabilities help to protect running applications.

What Are Private Artifact Repositories?

Artifact repositories or container registries are used to manage the storage and distribution of binaries and container images.

These artifact repositories and container registries help ensure the integrity and provenance of the software they contain by integrating with software composition analysis (SCA) and container image scanning tools, and also storing SBOMs and image signatures.

SCA complements the artifact repositories in that it provides the requisite analysis of the dependencies and the code, which is continuously updated and scanned.

Software composition analysis scans for and detects vulnerabilities in released software code that is typically contained in the [common vulnerabilities and exposures \(CVE\) list](#).

What Are Signing Tools?

Signing tools have been much needed in the developer community as a way to sign images and configuration files and to verify that the signatures match prior to use, thus helping to ensure that an attacker has not tampered with the code.

Called “the John Hancock and wax seal of the digital era” [by Wired](#), the open source

project [Sigstore](#), released in 2021, is a set of tools designed to ensure that the provenance of any open source software or code uploaded on Git or integrated into the CI/CD supply chain has been verified.

Security testing should start at the very beginning of the development cycle and continue through all stages of application development, including runtime testing.

Sigstore tools enable checking provenance by making it easier to sign software artifacts as they move through a pipeline, and easier to verify those signatures with the signature transparency log. Sigstore makes it easy to automate the process of digitally signing and checking the integrity of components by validating the signature. In May 2022, the release of Kubernetes 1.24 marked [the first Kubernetes version to use Sigstore](#).

What Are the Basics of Supply Chain Security Testing?

Teams validate the security of the software they are building through application security testing. There are many types of security testing tools which are designed for different stages of the software development life cycle.

“Security testing should start at the very beginning of the development cycle and continue through all stages of application development, including runtime testing.

Test-driven development is a software development process in which software requirements are converted to test cases before software is fully developed — rather

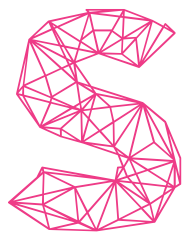
than afterward. Tracking of all software development is done by repeatedly testing the software against all test cases.

Wherever possible, security testing should be automated and integrated into CI/CD with policies applied, so that they cannot be overlooked or bypassed (which is a risk if they are initiated manually). The earlier security testing is included in the life cycle, the less likely the tests are to slow down release schedules. Issues found in early stages of software development are generally easier to fix than those found later.

Ongoing testing and container scanning throughout CI/CD should involve [static testing \(SAST\)](#) and fuzz testing. Dynamic application security testing (DAST) is used for tests at runtime.

CHAPTER 04

A Best Practices Checklist



o we have laid down a blueprint for supply chain security — the things your organization needs to keep its software safe at all stages of its life cycle.

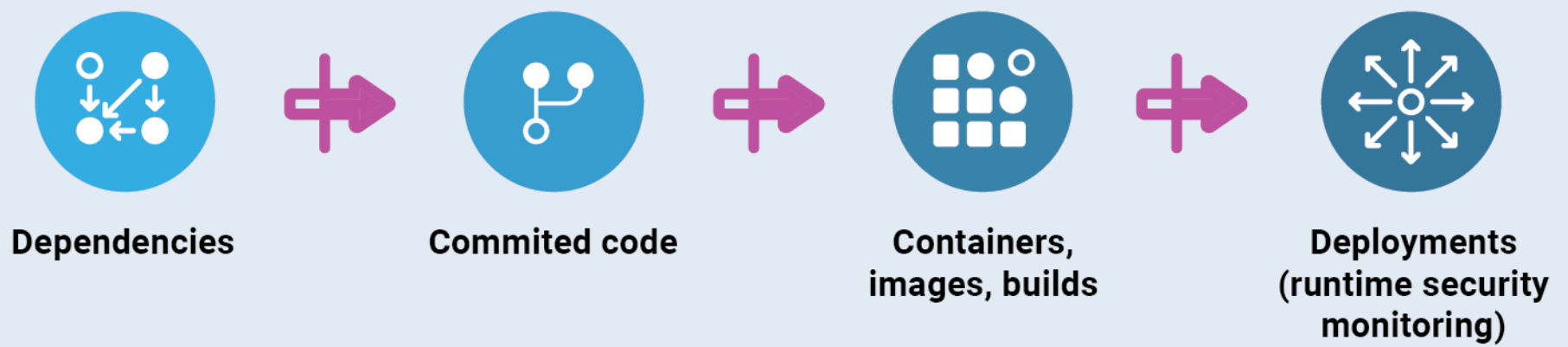
This includes software provenance with SBOMs and SLSA. We saw how GitOps represents a cornerstone for DevSecOps. We also reviewed the management and support of supply chain security best practices as code, thanks to its reliance on versioning and traceability, along with the audit trails it supports, and other benefits for cloud native environments.

Now, we are offering a checklist, including a sample CI/CD pipeline. Other checklists exist, such as the [CNCF's supply chain security assessment](#), and they are worth checking out. The idea is to create and follow a checklist that best suits your particular use case.

Pre-Development

The majority of software created today includes software from external sources; often that's open source software. Ideally, all software from external sources will have associated **SBOMs** and follow at least **SLSA level 1** guidelines. However, this is not yet the case for every open source project or for all code from vendors. Luckily, there are tools available, such as [Syft](#) or [Trivy](#), that you can use to generate an SBOM for code from external sources.

What Is Scanned and Tested?



Source: Red Hat

© 2022 THE NEW STACK

Fig 4.1 Scanning and testing takes place throughout the software life cycle to ensure supply chain security.

Meanwhile, trusted sources should be used whenever possible. This involves verifying signatures when pulling content from external sources and accessing code that has been properly vetted and stored in trusted private repositories. Additionally, use Policy as Code and other processes help to prevent code from being committed that does not adhere to policy.

The Development Cycle

Applications and code should be **scanned and tested continuously, through all stages of the supply chain**. Automated testing should begin as soon as the developer accesses libraries and commits code to Git. Testing should continue through the container builds, all along the CI/CD track, and extend through runtime, covering all clusters and nodes in cloud native environments.

SCA and other tests can scan and test dependency libraries in a trusted repository (which serve as a trusted proxy between the developers and public repositories, such as GitHub, GitLab, etc.).

The importance of testing across CI/CD cannot be overstated when building custom

software. It's key for developers to make pull requests and build their code in containers while also being able to vet the steps across the pipeline to ensure the code has not been compromised.

Committed code on Git:

- ☐ **Code tests are completed.**
- ☐ **Signatures are checked for artifacts**, with tools such as Ansible or Sigstore content signing technology.
- ☐ **Policies are stored as code** and automatically verified as part of the process to help ensure code meets security, compliance and other policies that apply to the very beginning of the CI/CD process.

CI/CD

- ☐ **Security tests and container image scanning** continues so that only trusted container images are deployed. For example, security SAST tests should be run on source code, and all dependencies should be scanned for known vulnerabilities. Container images, and all the packages contained within them, should also be scanned as part of the CI build process. Linters should also be run on application configuration once the deployment YAML and Helm charts are available.
- ☐ **Container image signature is verified** with tools such as Sigstore or Ansible content signing technology.
- ☐ **GitOps in place:** Tools, such as ArgoCD for GitOps, help to ensure the Git repository remains the single source of truth for application configuration files.

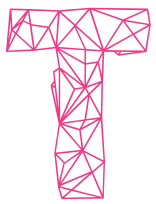
Argo CD will continuously monitor deployed applications to ensure the configuration on the cluster matches what's stored in Git.

Deployment and Production

- ☐ **Security monitoring continues.** Ensure that platform and application monitoring solutions are in place, including runtime behavioral analysis tools.
- ☐ **Testing includes dynamic application security testing (DAST),** which should be done in a staging environment.

CONCLUSION

Building on the Security Blueprint



he blueprint presented in this ebook can't provide absolute protection for your software supply chain, but it's a way to reduce harm and risk to an acceptable level.

Detecting and removing any and all vulnerabilities and potential exploits across the supply chain is the Holy Grail in security — and reaching the ultimate goal of a perfectly secure software supply will likely remain elusive. The best we can do is continue mitigating vulnerabilities as best as we can.

Major attacks, such as the massive exploitation of the [Log4Shell vulnerability](#) and increasing ransomware threats, have served to provide a wake-up call for many, if not most organizations to take supply chain security more seriously than in the past. Meanwhile, human error remains the main culprit behind attacks, as virtually all IAM policies in use today are not as effective as they should be, and phishing attacks remain a favorite way for attackers to gain access to networks and data.

The long-term solution consists of the automation of best practices and tool management for supply chain security that minimizes human intervention and preventable misconfigurations.

In the coming years, major supply chain attack vectors will likely change — signatures might make it so difficult for attackers to exploit pipelines that they will look for new ways to spread bad code.

Scanning tools will continue to improve, but they will likely only become better at helping security teams with threat assessment and triage than ever being able to

detect and remove every single security bug or bad code from the supply chain.

While software composition analysis (SCA) and other scanning processes and tests across CI/CD and through production are critical, DevOps and security teams will continue to struggle with putting detected vulnerabilities into context to determine their potential as exploits.

Adoption of SBOMs and SLSA guidelines will increase. DevOps teams will further integrate tools that examine code added to the pipeline to ensure it has the right signatures, as a way to check provenance and build further trust. But while they are very powerful tools, they cannot protect supply chains from unknown vulnerabilities.

So exploitable vulnerabilities will keep showing up in the otherwise closed loops of supply chains indefinitely. And security teams will increasingly respond by applying patches and fixes on a rolling basis, in addition to implementing policy to support automated responses to attacks when they happen.

It is thus key to be ready to react and fix critical security bugs and potential exploits as they become known. In doing so, organizations need to ask: How often are patches implemented: every week, month or even quarter? Are updates automated and consistently scheduled?

Better Communication Needed

On an industry-wide scale, communication of major exploits and vulnerabilities should be more properly channeled when they occur.

The supply chain threat that the Log4j vulnerability posed only became widely known when a blog post was written about how it was being used to compromise [Minecraft servers running Java](#), and how the contents of the video game's chat

messages between players and other data were exposed. This was after a member of the security team at Alibaba, the online retailer, had already discovered the exploitable bug in late November 2021, and reported it to the [Apache Software Foundation](#).

“How well is your security team prepared to apply the necessary patches and fixes as a best practice, when the next dangerous vulnerability is discovered?”

That’s a very roundabout way for the world to discover a global software supply chain threat.

When the next major highly exploitable vulnerability occurs, on par with Log4j’s, how well will it be communicated? Just as importantly, how well is your security team prepared to apply the necessary patches and fixes as a best practice, when that dangerous vulnerability is discovered? These are all critical issues the industry will need to address.

Community support and collaboration will remain essential to cloud native security practices in the future. This will include drawing from such community resources as The Linux Foundation’s [Trust and Security Initiative \(TSI\)](#), which was created in part to communicate security projects that deserve interest.

Organizations should also foster and support their developers’ contributions to open source security tools and projects.

There’s no way to totally eliminate risk from the software supply chain. But following — and building on — the blueprint we’ve shared in this ebook can help your organization defend its code, and its business, wisely and well.

About the Author



[B. Cameron Gain](#) is the founder of ReveCom Media Inc., which offers analysis and testing services for software tools used by developer, operations and security teams. His obsession with computers began when he hacked a Space Invaders console to play all day for 25 cents

at the local video arcade in the early 1980s. He has since become a long-time and steadfast Linux advocate and loves to write about open source software. In addition to its frequent appearances in The New Stack, his byline has appeared in Wired, PC World, CIO, Technology Review, Popular Science and Automotive News.

Disclosure

The following companies mentioned in this ebook are sponsors of The New Stack: Cloud Native Computing Foundation (CNCF), GitLab, Palo Alto Networks and Red Hat OpenShift.

TNS owner Insight Partners is an investor in Ambassador, ApolloGraphQL, Aqua Security, Armory, Docker, Gravitational, Honeycomb.io, Jit, Kasten, Kubeshop, LaunchDarkly, Lightrun, Mirantis, Spacelift, StormForge, Tigera, Torq and Tricentis, all of whom are also sponsors of The New Stack.

