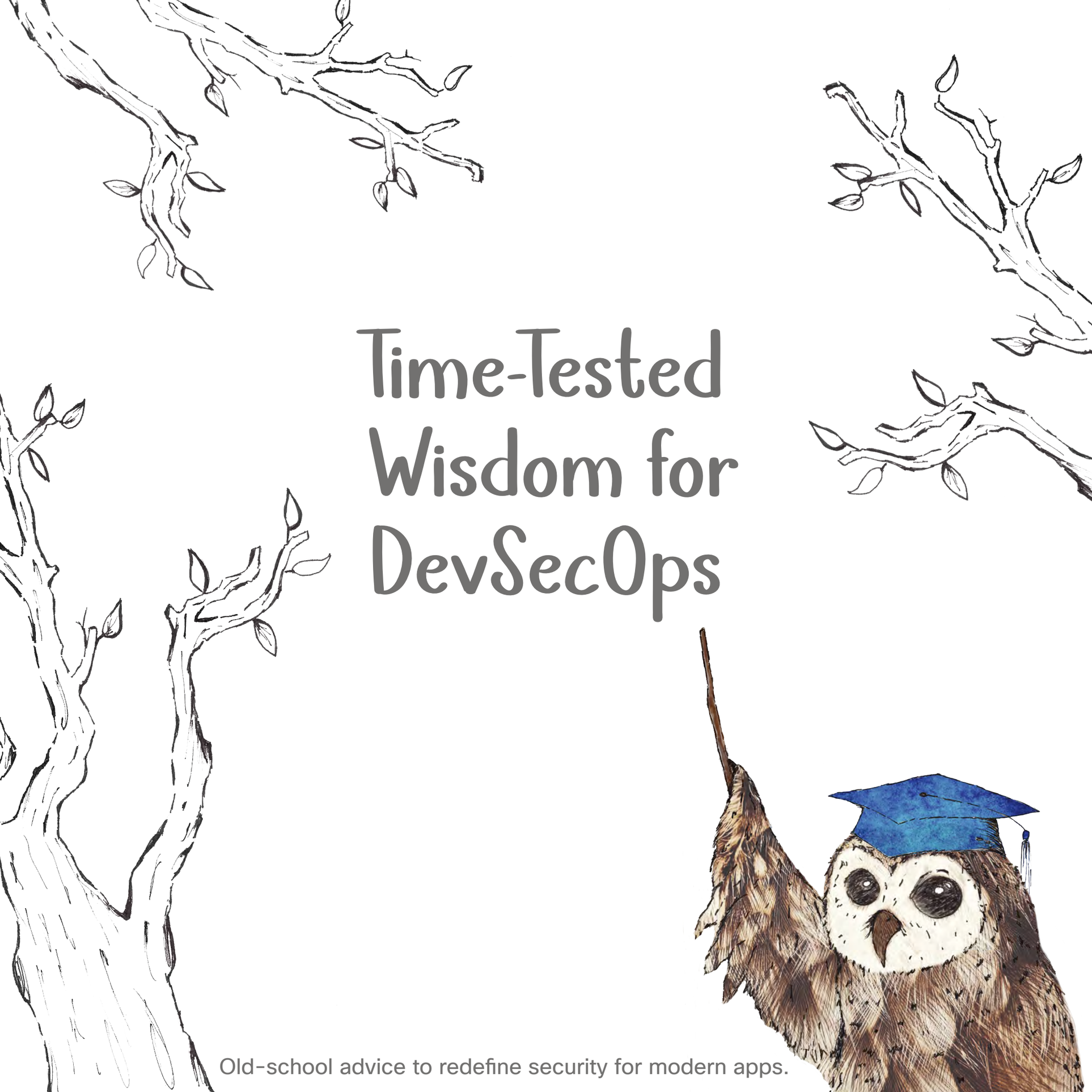
Time-Tested Wisdom for DevSecOps



Old-school advice to redefine security for modern apps.

Introduction

We've all grown up hearing certain sayings repeated so often that they've become clichés (such as "an apple a day keeps the doctor away" as encouragement us to eat our fruits and veggies).
Why do these sayings endure? Because they're universal truths!
In this eBook, we thought we'd have a little fun and put these sayings to the test in the context of "shift-left security," the practice of moving security earlier in the software development lifecycle (SLDC) —the genesis of DevSecOps.

Today's enterprise is under constant pressure to speed up delivery of new software and features to stay competitive in a crowded market. But this velocity comes with a cost. Coupled with the distributed nature of the cloud and an increased reliance on microservices and open source, security is more critical than ever before. "Old-school" SLDC, where the security team has little or no influence over software design or development, no longer has a place in our new "cloud-first" world. Time for a new paradigm, where developers and security practitioners team up and work together every step of the way.

Lesson 01

Many hands make light work.

The DevSecOps collaborative workflow isn't unprecedented. DevOps also emerged from a need to integrate functions for speed and efficiency. Gone are the days when developers and QA worked in lockstep while the operations teams and systems administrators toiled away down the hall and in the datacenter. This siloed approach not only prompted an "us-versus-them" mindset, it also bogged down the development cycle. DevOps brought the two together, introduced automation to help with testing and infrastructure provisioning, and added monitoring tools to alert engineers to potential failures.

DevSecOps is a natural evolution of this trend. By bringing DevOps and security together to work as a team, it empowers developers to iterate and release more quickly, without raising the risk of introducing security vulnerabilities. The approach is exceedingly helpful for both your CI/CD pipeline and your software supply chain.

You know that old saying, "teamwork makes the dream work." That's some wise advice!

UP NEXT
Lesson 02

Lesson 02

An ounce of prevention is worth a pound of cure.

Neglecting security can lead to disastrous consequences. When a vulnerability is exploited, it takes a toll on the entire company. Engineering teams will have to devote resources to incident response, damage control, and vulnerability patching. Security teams[may need to lock down developer access to cloud resources. Releases may grind to a halt. Your boss may be grumpier than usual. There will be system downtime, which will make customers even grumpier than your boss.

Breaches are also expensive. An IBM/Ponemon[1] report found that the average time to identify and contain a data breach was 287 days. The report pegged the average cost of a breach in hybrid cloud environments at $3.61 million. Ouch.

There might be compliance backlash, too. If your application mishandles personally identifiable information (PII) or protected health information (PHI), the result could be heavy financial penalties or legal consequences. GDPR fines[2] resulting from data protection and privacy violations can be particularly staggering.

What's more, the fallout from a breach continues long after mitigation. A company's brand is tarnished by stolen data or long system outages. If your applications are not secure, it's only a matter of time before those security threats come knocking at your door. And when they come, your customers will go.

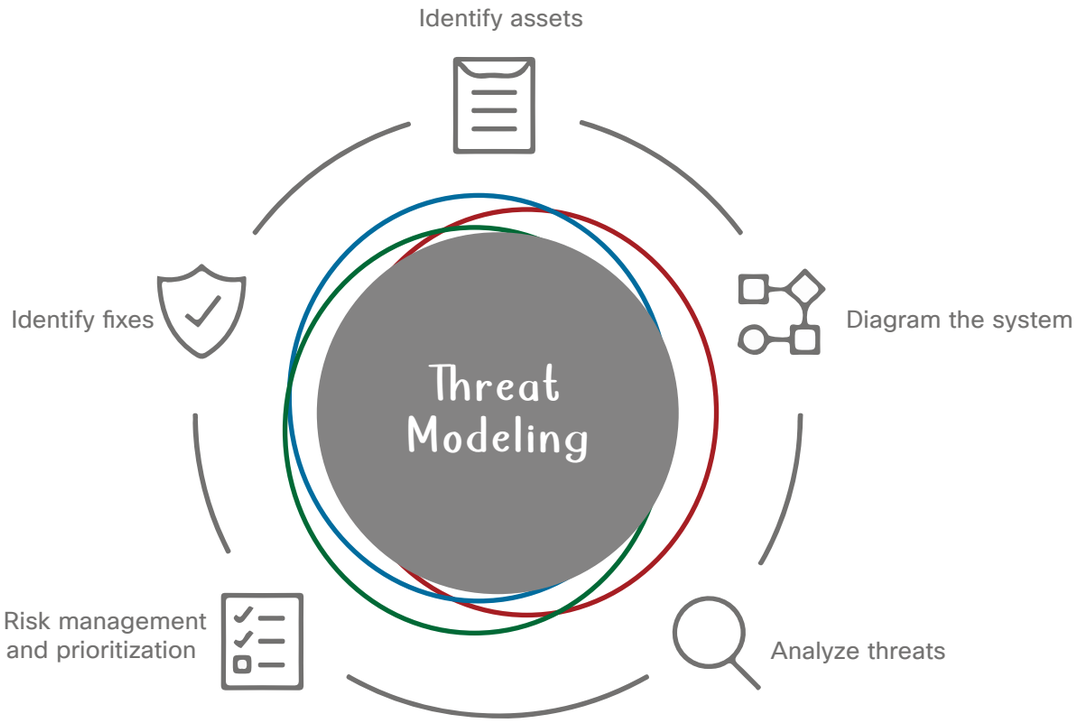
Given these risks, it makes sense to prevent vulnerabilities and breaches from happening in the first place.

Lesson 03

Don't put the cart before the horse.

The traditional approach to application security involves post-deployment manual checks, reviews, and patch updates. Security threats are handled reactively—after an exploit makes the news or when logs reveal suspicious activity. The DevSecOps approach reverses this workflow, conducting design, implementation, and testing during the development phase of the timeline. This is also the perfect point in the cycle to discover and mitigate security vulnerabilities.

Among the Open Web Application Security Project (OWASP) list of Top Ten Web Application Security Risks for 2021[3] is insecure design. OWASP notes: "If we genuinely want to 'move left' as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures." This security-aware planning helps to ensure that system designs incorporate security practices like access control, auditing, levels of isolation, and alerting.



Secure coding practices are key to the DevSecOps approach. These may include input validation, parameterized queries, and proper handling of sensitive data or credentials. The list of best practices continues to grow, and their application varies from language to language and from framework to framework. One solution is for developers to use IDEs with built-in tools that analyze code for potential risks as they work. Many IDEs also incorporate linters to ensure that code is properly formatted and conforms to coding standards.

Access control is another important practice. Most web applications have a login component and different levels of access (such as “basic user,” “privileged user,” or “administrator”). The OWASP list referenced above includes identification and authentication failures, as well as broken access control. It’s critical to make this a priority during application design and not treat it as an afterthought. If a malicious user gains access to your web application with administrator privileges, the impact could be devastating.

Lastly, unit testing at the early stages of application development should include security-related tests, along with the customary tests for bugs and business logic. These tests can verify that proper measures are in place for securing access control, preventing injection attacks, and validating inputs.

The bottom line is that security and vulnerability testing belong before deployment, not after.

Lesson 04

Knowledge is power.

We've established the importance of designing security into your products and running critical tests as part of your process. But it's equally important to make provisions for observability, monitoring, and alerting.

The DevSecOps approach includes instrumentation for exposing logs, metrics, and traces, making the components of an application observable. This observability data can help the DevOps team more easily discern the state and the health of an application. It can also use the data to gain insights into the root cause of incidents, including security threats.

While observability data helps you identify the reasons why an application is behaving a certain way, monitoring lets you set specific metrics—such as a certain number of bad password attempts, HTTP requests per minute, or database query latency—that indicate that something suspicious may be underway. You'll also want to set up alerts to notify team members when these conditions are met.

Bottom line: you can't fix something you don't know is broken.

UP NEXT
Lesson 05

Lesson 05

Loose lips sink ships.

The CI/CD pipeline deserves special attention. Engineering teams can use the automated pipeline to apply security measures when building and deploying an application. But they need to ensure that the pipeline itself is secure.

The CI/CD pipeline is responsible for provisioning infrastructure and deploying workloads. Your CI/CD pipeline will be configured to use your organization's credentials for accessing cloud provider accounts, service accounts, and other resources. Storage of these credentials must be secure. Managed pipeline services include secret-management tools for encrypted storage, decrypting secrets only when your pipeline uses them.

The lesson here is to keep tight control of your secrets and credentials.

UP NEXT
Lesson 06

Lesson 06

Don't reinvent the wheel.

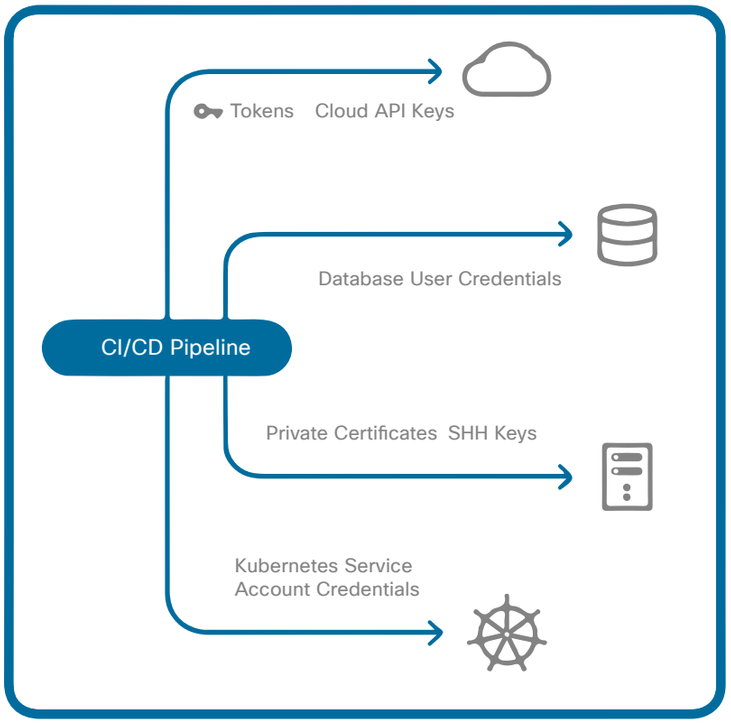
Restricting the task of infrastructure provisioning and workload deployment to your CI/CD pipeline enables you to ensure consistency and repeatability. If you allow these tasks to be performed manually, the margin of error increases, since a human may skip critical security or configuration steps—exactly what you're trying to avoid.

This type of automation is known as infrastructure as code (IAC). IAC lets you create declarative configurations of your system architecture. Without this, any variation inadvertently introduced by an engineer will create drift, which translates into security risks and is difficult to track or audit. In contrast, IAC tools (such as AWS CloudFormation or Terraform) allow you to record and review configurations, since they are stored in a version-control system.

Static code analysis

Earlier, we mentioned the use of IDEs for real-time analysis of code. The same goes for static code analysis tools, such as SonarQube[4], which you can integrate and run before building the application to ensure that all code conforms to security standards and conventions.

CONTINUE
Lesson 06



Vulnerability scanning

Just as static code analysis examines your application's code for security flaws, vulnerability scanning checks container images or third-party dependencies for security risks. Tools such as Trivy or Grype can be integrated with the CI/CD pipeline to perform scans. Later, we'll cover the threat of third-party dependencies in more detail.

Alerting on broken CI/CD

When all infrastructure configurations and changes—including updates or rollbacks or restarts—are handled by your CI/CD pipeline, there are several implications.

First, your CI/CD pipeline must be highly available. You depend on your CI/CD pipeline to make changes to your infrastructure. A security breach may necessitate shutting down or isolating parts of your application. If your CI/CD pipeline is unavailable, you'll be unable to make infrastructure changes. What's more, if the CI/CD pipeline itself is vulnerable to attack—making it unavailable or exposing its sensitive data for exfiltration—your entire system will be at risk.

For these reasons, it's important to create a plan of action for handling a broken CI/CD pipeline. This plan should include how you'll keep your systems running securely while you repair your pipeline. Alerts should be top priority.

Repeatable, automated processes are the key to success, whereas manual processes are akin to reinventing the wheel. So, just don't.

Lesson 07

Take care of the pennies and the pounds will take care of themselves.

In most modern enterprises, even proprietary applications utilize a set of third-party dependencies, including open-source libraries, packages, container images, or tools. In turn, those dependencies have their own sets of dependencies. The entire chain of dependencies—from your first library dependency to the smallest I/O package at the end—constitutes your software supply chain. Because security vulnerabilities in the supply chain expose your applications to risk, it's critical to apply heavy scrutiny to every single package.

Libraries continually undergo version updates, and each new version brings the potential for introducing a vulnerability. Some open-source projects may be trusted and innocuous for a long time in order to gain trust and adoption, only to be updated with intentionally malicious code in a later version.

Common supply-chain attack vectors include:

Dependency confusion. A dependency-confusion attack occurs when your package manager updates your application with code from an unsafe public package, rather than your vetted private package. For example, your application might depend on a trusted internal package called "auth-lib." Internally, your package is at version 2.9.3. A malicious actor might publish a package under the same name and release malicious code as version 2.9.4.

The intent of the attack is to confuse package managers that are not properly configured to prioritize private packages over public packages. The susceptible package manager will discover that a new version of the (public) package is available, then download and install it, overwriting the trusted package.

Secure your package manager against dependency-confusion attacks by prioritizing internal packages over public packages, even those with higher version numbers.

Vulnerabilities in container images.

When an application directly uses or depends on tools or libraries that use containerization, container images can introduce vulnerabilities[5]. Images are often more opaque than open-source libraries, where the source code is freely available for examination.

Compromised libraries/packages.

Dependency libraries undergo frequent updates for security patches, feature additions, and performance enhancements. Even if the version updates come from the authoritative maintainer of that software, a malicious actor can manipulate the update[6], allowing dissemination of malicious code.

Package corruption or sabotage.

Even package maintainers may intentionally sabotage their own packages or introduce malicious code. If an open-source library is present at a deep enough level in the software supply chain to be widely used, the results can be crippling.

It's critical to take measures to safeguard your supply chain, including the following:

Take a complete inventory of your software supply chain.

By starting with a software bill of materials (SBOM), you have a baseline from which to begin checking your dependencies. OWASP CycloneDX[7] is a lightweight SBOM standard that can help.

Assess your dependencies for security risks.

For example, Security Scorecards is a tool from the Open-Source Security Foundation that assigns scores to each open-source dependency to give you an understanding of your risk.

Use lockfiles.

Lock the dependencies you use to specific versions you've already vetted. Updates should be vetted, as well, not automatically updated.

Integrate vulnerability scanning tools in your CI/CD pipeline.

It's impossible to manually check all dependencies for security threats. Vulnerability scanning tools can examine container images and third-party libraries for security t hreats.

Use signed images and trusted registries.

This ensures that the images you incorporate are created by verified parties.

By paying careful attention to every dependency, you'll be securing the entire stack.

Lesson 08

Use Panoptica.

Panoptica

The Cisco Secure Application Cloud

Ready to integrate all these lessons into your development process? Panoptica is the only solution that provides robust security from CD/CD to runtime—for containers, Kubernetes, and service mesh. Its benefits include:

Securing the cloud-native app-dev lifecycle from code to cloud.

By driving security automation through the application-development process, developers can assess and build security policies from the IDE, Terraform, and GitOps tooling.

An agentless, Kubernetes-based deployment approach.

Panoptica provides deep, real-time visibility and enforcement. Its agentless approach—with its single Kubernetes admission controller—offers simplified deployment without trade-offs in security controls. It provides real-time automated enforcement in runtime, and deep visibility into application context and compute resources in Kubernetes and container deployments.

Deep vulnerability analysis, including code signing and SBOM.

Get a closer look at all workloads through vulnerability analysis down to the image layer, SBOM analysis, and code signing to ensure validity of your applications from build through runtime.

Comprehensive API security.

Panoptica provides deep insights into API inventory, immediate risk assessment, and remediation—including OWASP API Security Top 10. It also gives you holistic API security, covering infrastructure and application logic.

Ready to learn more?

See Panoptica in action — no signup or credit card required. Take it for a spin and try out Panoptica's features with a ready-for-action pre-configured cluster.

SHOW ME THE DEMO!

Try Panoptica for free with your own Kubernetes cluster, using our simple QuickStart instructions. If you don't have a Kubernetes cluster yet, you can go play in our sandbox or check out the demo.

TRY THE SANDBOX

See Panoptica in action — no signup or credit card required.

References:

[1] IBM
URL: https://www.ibm.com/security/data-breach

[2] GDPR
URL: https://gdpr.eu/fines/

[3] OWASP - Top Ten Web Application Security Risks for 2021
URL: https://owasp.org/www-project-top-ten/

[4] SonarQube
URL: https://www.sonarqube.org/

[5] “container images can introduce vulnerabilities.”
URL: https://blog.gitguardian.com/codecov-supply-chain-breach/

[6] “manipulate the update”
URL: https://fossa.com/blog/embedded-malware-npm-coa-rc-ua-parser/

[7] OWASP CycloneDX
URL: https://cyclonedx.org/

[8] Security Scorecards
URL: https://github.com/ossf/scorecard