Engineering Continuous Security (a.k.a. DevSecOps)

This paper is derived from selections from my book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

May 31, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**How do you engineer Continuous Security (a.k.a. DevSecOps) ?**

IT security strategies and tools are at the top of the CIO priority list. Consequences of security attacks can include the following:

• Loss of sensitive or proprietary information

• Disruption to regular operations

• Financial losses relating to restoring systems

• Harm to an organization’s reputation

In the context of this white paper, Continuous Security encompasses both DevSecOps and Rugged DevOps. DevSecOps pertains to the culture aspects of “security as code,” while Rugged DevOps has to do with engineering security into design and deployment processes. Both have the tenet that security is everyone’s responsibility and that DevOps provides opportunities to improve security before deployment.

**Why Is Continuous Security Important to Engineering DevOps?**

DevOps compliance is a top concern of IT leaders, but information security is seen as an inhibitor to DevOps agility. Security infrastructure has lagged in its ability to become “software defined” and programmable, making it difficult to integrate security controls into DevOps-style workflows in an automated, transparent way. Modern applications are largely “assembled,” not developed, and developers often download, and use known vulnerable open-source components and frameworks.

**How Does Continuous Security Work with DevOps Engineering?**

As indicated in ***Figure 1—Continuous Security Engineering Blueprint, the DevOps Continuous Security Blueprint*** aims to move the organization to a better security posture. Each security flaw is carefully identified and is fixed one at a time to close the most urgent security gaps. DevSecOps identifies the most vulnerable concerns ahead of time and identifies how to avoid or move away from these bad positions. Without proper consideration given to security engineering practices, the continuous delivery of software changes facilitated by DevOps is risky. On the other hand, DevOps provides an opportunity to reduce security risks if security is integrated into the continuous delivery pipeline according to engineering practices.

The following are examples of the Nine Pillars of DevOps engineering practices for Continuous Security.

**Leadership:**

Leaders need to understand and sponsor a clear vision for security.

• Leadership demonstrates a vision for organizational direction, team direction, and three-year horizon, including security practices.

• Leaders intellectually stimulate the team status quo by encouraging new questions and questioning the basic assumptions about the work, including security practices.

• Leaders promote personal recognition by commending teams for better-than-average work, acknowledging improvements in the quality of work, and personally complimenting individuals’ outstanding work, including security practices.



**Figure 1—Continuous Security Engineering Blueprint**

**Collaborative Culture:**

Culture in organizations that work well with DevOps have a collaborative continuous security mindset.

• The culture encourages cross-functional collaboration and shared responsibilities and avoids silos between developers, operations, project management, quality assurance, and security.

• The DevOps system (toolchain) is created by an expert team and reviewed by a coalition of stakeholders, including security teams.

• Changes to end-to-end DevOps workflows are led by an expert team and reviewed by a coalition of stakeholders, including security.

• DevOps culture empowers and trains team members to take personal responsibility for security, compliance, and privacy obligations.

• Security engineers/architects are involved in the design for modular components and consulted when security patterns change within modules.

**Design for DevOps:**

Designing software for DevOps, at speed, requires application designers to master the engineering practices for continuous security.

• Software source code changes are pre-checked with static analysis tools prior to commit to the integration branch. This assures that the modified source code does not introduce critical software faults and security vulnerabilities such as memory leaks, uninitialized variables, array-boundary problems, and SQL injection.

• Software component analysis scans third-party components for known security vulnerabilities and identifies risk during the build process.

• Security frameworks for technology stacks that are used (such as Apache Shiro or Spring Security) are documented and shared with developers for their respective technology stacks.

• Software code changes peer code reviews include checks for defensive coding and security vulnerabilities prior to committing code to the integration/trunk branch.

• Common security components such as identity, authorization, key management, audit/log, cryptography, and protocols are maintained, published, readily available, and used within module development.

**Continuous Integration:**

Continuous integration (CI) within organizations that have multiple teams working concurrently on a project and different code bases is challenging. During the integration stage, it is critical to assess the application and understand the impact of code changes from a security point of view.

• A software version management system is used to manage versions of all changes to source code, executable images, and tools used to create and test the software.

• Incremental static analysis pre-commit and commit checks are wired into CI to catch common mistakes and anti-patterns quickly by only scanning the code that was changed. These checks identify security vulnerabilities through control flow and data flow analysis, pattern analysis and other techniques. These techniques find security-related issues such as mistakes in using crypto functions, configuration errors, and potential injection vulnerabilities.

• Binary artifacts are digitally signed and stored in secure repositories.

• Changes to security patterns used within software sources such as session management, authentication, authorization, and encryption code trigger a notification or pull request to security engineers.

**Continuous Testing:**

Continuous testing (CT) with DevOps has significant advantages for continuous security when engineering practices are followed.

• New security tests that are necessary to test a software change are created together with the code and integrated into the trunk branch at the same time the code is. The new tests are then used to test the code after integration.

• Security tests for each DevOps pipeline stage are automated and may be selected automatically according predefined criteria.

• Release regression tests include security tests. At least 85% of the security regression tests are fully automated, and the remaining are auto-assisted if portions must be performed manually.

• Test results that indicate possible security concerns are tagged for security analysis.

• Dynamic or interactive application security tests exercise the application for security vulnerabilities. Results of these tests are delivered to developers through tools and feedback loops native to their organization.

• If containers are used, image repositories are scanned for images with known vulnerabilities, hash checks for image drifts, and runtime checks for vulnerabilities during image deploys.

• Attack patterns, abuse cases, and tests are built for application module profiles.

• Unit, functional, and integration tests around—and especially outside—boundary conditions are run during CT. Tests include error handling, exception handling, and logic and negative tests.

• Automated security attack testing to include the OWASP Top 10RW75 integrated into automated testing.

**Continuous Monitoring:**

Continuous monitoring (CM) of security considers the dynamic nature of artifacts and infrastructure and a proliferation of objects and services to secure.

• Metrics and thresholds are automatically gathered, calculated, and made visible to anyone on the team that subscribes to them. Example security metrics include number of security defects identified in pre-production, percent of code coverage from security testing, number of failed builds due to security checks, mean time to detection, and mean-time-to-resolution

• Vulnerability information in consolidated to provide a comprehensive view into vulnerability risks and remediation across tools, pipelines, and apps—and over time.

• Metrics and events from production security controls such as WAF and RASP are used to improve security testing.

• Insight into security threats and events are shared and visible across DevOps teams to enable “attack-driven defense” methodologies.

**Continuous Security:**

Continuous security (CS) is itself a distinct pillar with standalone engineering practices that cross all the other pillars.

• All information security platforms that are in use expose full functionality via APIs.

• Immutable infrastructure mindsets are adopted to ensure production systems are locked down.

• Security controls are automated so as not to impede DevOps agility.

• Security tools are integrated into the CI/CD pipeline.

• Source code for key intellectual property on build or test machines is only accessible by trusted users with credentials. Build and test scripts do not contain credentials to any system that has intellectual property.

• External penetration tests (done out of band) scheduled either periodically or on a regular cadence are used to perform deep-dive analysis.

• Telemetry from production security controls such as WAF and RASP are delivered back to development teams to inform application updates.

• Accurate inventory of all software packages and version information is documented via infrastructure as code. Automated detection is used to identify whether any of the packages have known Common Vulnerabilities and Exposures (CVEs) associated and define specific remediation actions.

**Elastic Infrastructure:**

Elastic infrastructure environments offer advantages compared to legacy or traditional static infrastructure. However, elastic infrastructures need to follow engineering practices for continuous security because flexible infrastructures offer a broader range of attack surfaces.

• Configuration code includes automated checks, including ensuring unnecessary services are disabled and only ports that need to be open are open and permissions on files, audits, and logging policies are enforced. Development tools are not installed on production.

• Security-approved OS, software versions, and frameworks are used to compose required infrastructure. Security-related controls such as ACLs and FIM are defined as a part of infrastructure where applicable.

• A least-privilege model is enforced for processes running on shared infrastructure.

• Smaller clusters are used to reduce complexity between teams.

• Service provider partners security controls are validated to ensure they meet business requirements in their domains of the shared security model.

• IaaS or PaaS service provider security controls are validated to ensure that they meet business requirements in their domains of the shared security model.

**Continuous Delivery/Deployment:**

While DevOps enables new feature deliveries to users quickly, to minimize risk during deployment, the following engineering practices are important:

• Release-to-production decisions are determined according to predetermined metrics, which include security metrics.

• A whitelist policy for application segmentation is enforced during deployment for each environment, especially production.

• Continuous deployment processes trigger run-time security and compliance checks, including ensuring unnecessary services are disabled and only ports that need to be open are open and permissions on files, audits, and logging policies are enforced. Verify development tools are not installed on production.

• All secrets used for deployment are vaulted and retrieved programmatically during run-time or initialization of the continuous delivery process.

**Implementing Continuous Security**

The following are recommended steps for implementing continuous security:

• Empower and train developers to take personal responsibility for security.

• Embrace security assurance automation and security monitoring.

• Require all information security platforms to expose full functionality via APIs for automatability.

• Use proven version control practices and tools for all application software, scripts, templates, and blueprints used in DevOps environments.

• Adopt immutable infrastructure mindset where production systems are locked down.

• Security platform capabilities such as identity and access management, firewalling, vulnerability scanning, and application security testing need to be exposed programmatically.

• The integration and automation of these security controls are enabled throughout the DevOps life cycle in automated toolchains.

• Information security teams can then set policies, which can then be applied programmatically based on the type of workload.

**Summary**

In summary, the answer to the question: “**How do you engineer Continuous Security (a.k.a. DevSecOps) ?”** has been explained in this white paper.

Use a comprehensive Continuous Security Blueprint and engineering practices that integrate security throughout the Nine Pillars of DevOps (Leadership, Culture, Design, Continuous Integration, Continuous Testing, Elastic Infrastructure, Continuous Monitoring and Continuous Delivery).

Otherwise security problems can be amplified and accelerated as the release cadence increases.

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved