(https://www.gartner.com/home)

LICENSED FOR DISTRIBUTION

10 Things to Get Right for Successful DevSecOps

Published: 03 October 2017 **ID:** G00341371

Analyst(s): Neil MacDonald, Ian Head

Summary

Integrating security into DevOps to deliver "DevSecOps" requires changing mindsets, processes and technology. Security and risk management leaders must adhere to the collaborative, agile nature of DevOps to be seamless and transparent in the development process, making the Sec in DevSecOps silent.

Overview

Key Challenges

In the world of digital business, product development teams that deliver new IT-enabled capabilities to customers are king, rather than information security or IT operations.

Information security must adapt to development processes and tools, not the other way around.

Organizations producing new applications and services using DevOps have the same responsibility to produce secure and compliant code as required by any other application.

Perfect application security isn't possible and, in misguided pursuit of zero vulnerability applications, heavyweight security testing is an obstacle to the speed of digital business.

Recommendations

Security and risk management (SRM) tasked with ensuring application and data security should:

Integrate security and compliance testing seamlessly into DevSecOps so that developers never have to leave their continuous integration or continuous deployment toolchain environment.

Scan for known vulnerabilities and misconfigurations in all open-source and third-party components. Ideally, build out a complete bill of materials using software composition analysis.

Stop trying to remove all unknown vulnerabilities in custom code, which increases false positives. Instead, focus developers on those with the highest severity and confidence.

Be open to using new types of tools and approaches to minimize friction for developers (such as interactive application security testing [IAST]) to replace traditional static and dynamic testing.

Scale your information security team into DevOps by using a security champion model.

Treat all automation scripts, templates, images and blueprints with the same level of assurance that you would treat any source code.

Strategic Planning Assumptions

By 2019, more than 70% of enterprise DevSecOps initiatives will have incorporated automated security vulnerability and configuration scanning for open-source components and commercial packages, up from less than 10% in 2016.

By 2021, DevSecOps practices will be embedded in 80% of rapid development teams, up from 15% in 2017.

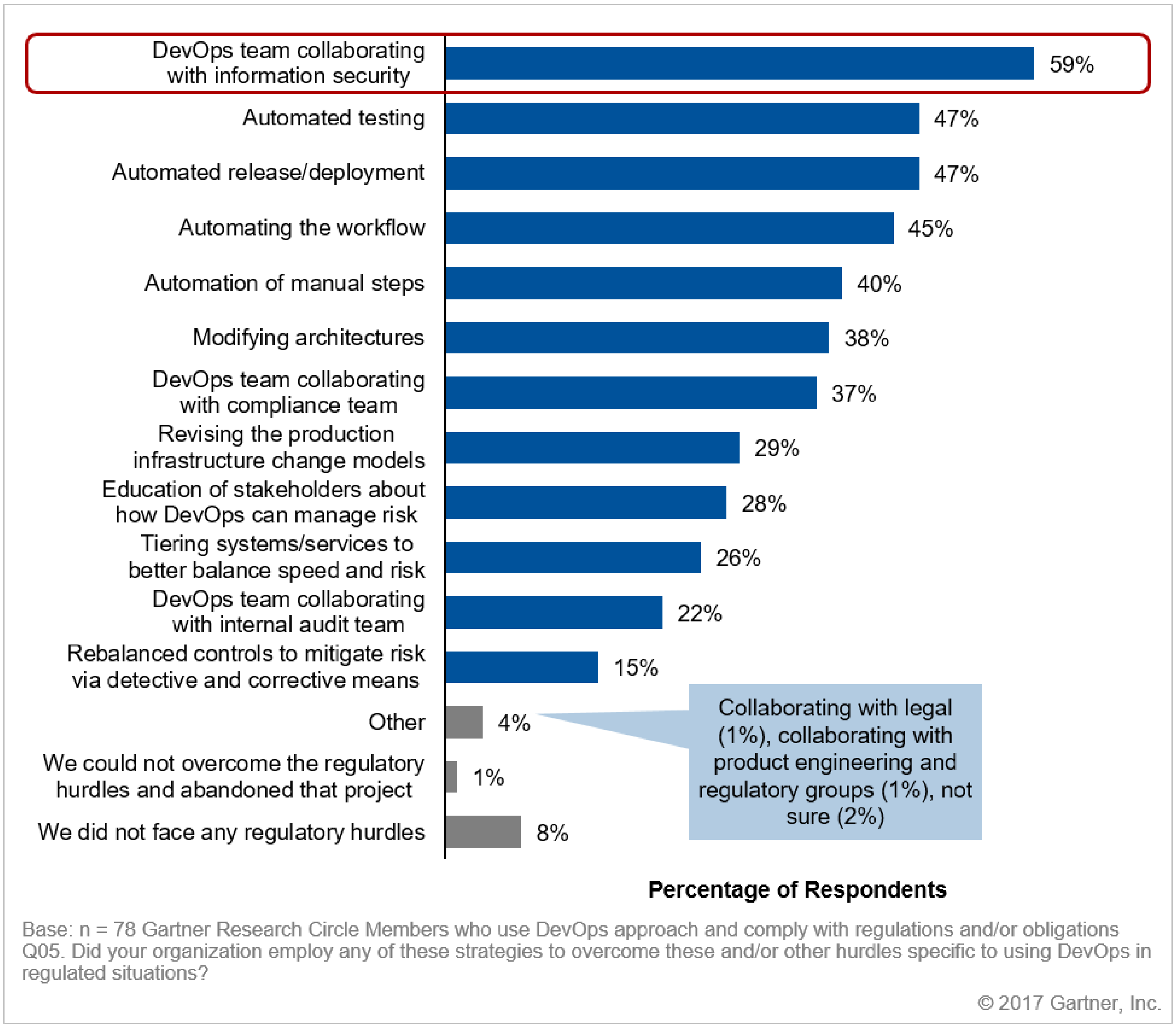
Introduction

Every business is a digital business. Every company is a software company. The key to gaining and sustaining competitive advantage in digital business, and a role in a digital society, will be in the development and continuous improvement of new IT-enabled capabilities and services for customers.

DevOps is becoming the preferred approach for the rapid development and continuous delivery of these new IT-enabled capabilities. Implemented correctly, DevOps offers IT organizations improved speed of development by embracing a collaborative philosophy that tears down traditional silos of development and operations. However, in most cases, security and compliance have been afterthoughts to DevOps.

The good news is that Gartner survey data shows that interest in collaboration with information security is the highest-ranked strategy for dealing with DevOps in regulated environments (see Figure 1).

Figure 1. Strategies to Overcome Hurdles to Using DevOps in Regulated Situations



Source: Gartner (October 2017)

In the past 12 months at Gartner, how to securely integrate security into DevOps — delivering DevSecOps — has been one of the fastest-growing areas of interest of clients, with more than 600 inquiries across multiple Gartner analysts in that time frame. From conversations with clients and by analyzing successful DevSecOps initiatives, we have seen what works, what doesn't and which approaches have the most success. Here, we provide specific guidance to address 10 areas that SRM leaders must get right to successfully enable DevSecOps.

Analysis

Adapt Your Security Testing Tools and Processes to the Developers, Not the Other Way Around

The DevOps philosophy is rooted in tearing down the traditional silos of IT, namely between development and operations. However, security represents yet another silo that needs to be removed and integrated. If implemented correctly, the "Sec" in DevSecOps should be silent. Our goal should be to make security as seamless and transparent as possible to the developers (in other words, secure DevOps).

Don't force information security's old processes to be adopted by DevOps developers. Instead, plan to integrate continuous security assurance seamlessly into the developer's continuous integration/continuous development (CI/CD) toolchain and processes. This is a significant mindset change for information security professionals accustomed to forcing developers to conform to our processes, so it will require several changes, such as:

- Never making developers leave their native toolchain environment. This means integrating your testing with their integrated development environment (IDE) and CI/CD toolchain tools (such as Jenkins, Bamboo, Microsoft Team Foundation Server, Chef, Puppet or Ansible).
- Designing the security scanning so that information security professionals are not needed to scan applications. The need for an information security assistance should be a rare exception. Where possible, everything should be automated and transparent to the developer.
- Architecting your security and compliance scanning to be performed automatically and programmatically via application programming interfaces (APIs), rather than through the native console of the security vendor. Testing and results should be natively presented in your development dashboard (for example, Sonar, Maven or Jenkins). As such, demand that all of your security scanning tools and services be fully API-enabled.
- Gathering simple automated security requirements for all new applications, using a threat modeling tool as part of the non-functional requirements.
- Addressing detected issues through the developer's existing bug tracking process (such as JIRA or BugTraq).
- Changing your mindset away from one-time gating using security scans to a continuous security assurance process, integrated from the beginning and assessed with each new iteration.

Quit Trying to Eliminate All Vulnerabilities During Development

There is a famous quote going back centuries and made famous by Voltaire:

> Il meglio è nemico del bene (Pescetti)

> Le mieux est l'ennemi du bien (Voltaire)

Translated, the saying means "perfect is the enemy of good." [1] This is as true now as it was centuries ago, especially in the world of digital business, where information security's drive for perfect security is at odds with the business and developer's need for speed and agility.

Perfect security is impossible. Zero risk is impossible. We must bring continuous risk- and trust-based assessment and prioritization of application vulnerabilities to DevSecOps. In a futile attempt to remove all possible vulnerabilities from applications, we are slowing developers down and wasting their time chasing issues that aren't real (false positives) or addressing lower-risk vulnerabilities that are real, but not directly or easily exploitable.

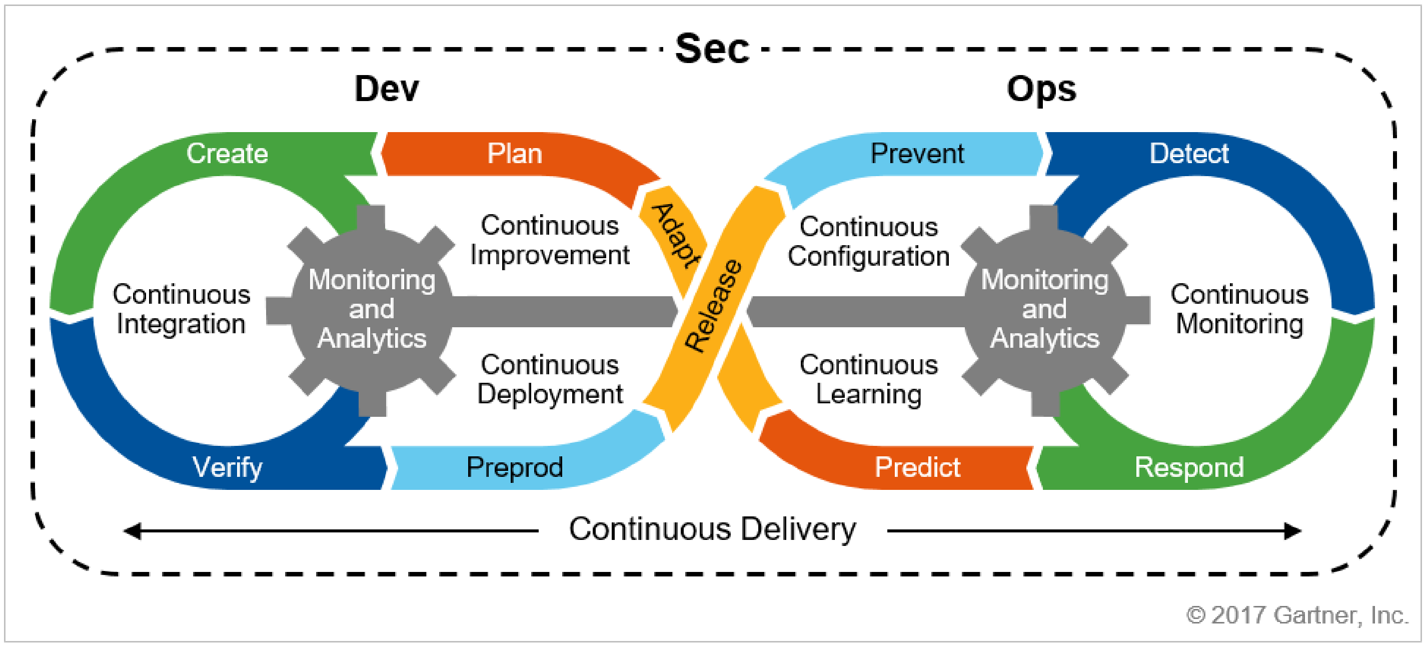
There is a trade-off between the speed of testing, wasting developer time with false positives and increasing risk from false negatives. With DevSecOps, information security must accept that having zero vulnerabilities is neither possible nor desirable if it significantly slows down the pace of new service delivery and innovation.

Moreover, we can compensate for much of the residual risk of a known, lower-risk vulnerability or of an unknown vulnerability that gets into production by using runtime protection controls, such as:

- Network- and host-based intrusion prevention systems (IPS) (to protect against attacks on known OS and software packages) (see "Market Guide for Cloud Workload Protection Platforms" and "Magic Quadrant for Intrusion Detection and Prevention Systems")
- Web application firewalls (WAF) (see "Magic Quadrant for Web Application Firewalls"), especially if the WAF is configured for protection based on explicit knowledge of known vulnerabilities gained through dynamic application security testing (see "Magic Quadrant for Application Security Testing")
- Runtime application self-protection (see "Emerging Technology Analysis: Runtime Application Self-Protection")
- Application monitoring and protection (see "Application Performance Monitoring and Application Security Monitoring Are Converging")
- Botnet mitigation to protect against user interface abuse and information scraping
- In-depth defense at the application layer, including load balancing, denial of service and distributed denial of service protection

By thinking of application security via DevSecOps as a continuous improvement process spanning development and operations (see Figure 2), we don't have to eliminate all possible vulnerabilities in development, and we can view runtime protection as an integrated part of our DevSecOps strategy.

Figure 2. DevSecOps: Secure Development as a Continuous Improvement Process



Source: Gartner (October 2017)

Security doesn't stop in development (the left side of Figure 2). The entire DevOps life cycle needs to be secured, including when new services are deployed into runtime operation (the right side of Figure 2). Security, like development, is becoming a continuous delivery, learning and improvement process — a mindset shift captured in Gartner's continuous adaptive risk and trust assessment (CARTA) (see "Use a CARTA Strategic Approach to Embrace Digital Business Opportunities in an Era of Advanced Threats"). Perform risk-based scanning in development, accepting that some vulnerabilities will get through and compensating for this risk with continuous run time assessment. Finally, absent a regulatory deficiency, how much risk is acceptable is not up to information security. It is a business decision ultimately made by the product/service owner.

Focus First on Identifying and Removing the Known Critical Vulnerabilities

Modern software is assembled, not developed. Developers make heavy use of prebuilt components, libraries, containers and frameworks (such as Apache Struts and Swing) from publicly available sources and repositories (such as GitHub, SourceForge, Maven central and Docker Hub). Custom code represents a minority percentage of the code in a modern application. For example, one Gartner client in financial services has standardized on Java for development, uses a preapproved internal component library and combines this with heavy usage of Swing. Combining these practices with standardized, prebuilt components developed for their internal reuse (for example, for input sanitization) meant that, on average, the actual amount of custom code left for the developer to write represented less than 10% of the final application.

This leads to a shift in focus for security scanning, as the majority of risk can be addressed by identifying known vulnerabilities and known misconfiguration issues in these libraries, frameworks and components before they are put into production. From a security perspective, it is a much easier problem to identify known vulnerabilities in known code than unknown vulnerabilities in custom code (discussed in the next section). This can be done in a variety of ways, in its simplest form, by matching files against vulnerability libraries. Vulnerability assessment vendors are adapting their scanning capabilities to address the need to do this automatically in DevSecOps, and some toolchain element vendors like Docker (https://docs.docker.com/docker-cloud/builds/image-scan/) are integrating this capability. The importance of this best practice cannot be understated. The recent and highly publicized breach at Equifax may have had a root cause of a known vulnerability issue in Apache Struts, as stated by Equifax (https://www.equifaxsecurity2017.com/2017/09/15/equifax-releases-details-cybersecurity-incident-announces-personnel-changes/) . [2]

Open-source software (OSS) presents a unique challenge, as the developer may simply cut and paste source code rather than link in an entire library or framework. This can't be simply identified using hashes, and identification will require comprehensive scanning of the source code itself. Moreover, the use of OSS may represent legal risk to the organization from licensing, so a commercial software composition analysis (SCA) solution is recommended to build a detailed inventory (bill of materials) for the application (see Note 1). There is also an extended life cycle benefit to this inventory. If a component in use is later found to contain a critical security vulnerability, the security team can quickly answer "Which applications are affected?" with a query to the repository (for example, Apache Struts, as in the example above, or OpenSSL's Heartbleed attack).

When assessing applications for known vulnerabilities, SRM leaders should scan:

- The entire content of the developer's unit of work (including the virtual machine, container and machine image). Containers (see "Security Considerations and Best Practices for Securing Containers") should be fully supported (see Note 2).
- All source code for embedded OSS code with known vulnerabilities.
- All OSS components (including libraries and frameworks).
- All OS system files, executables and dynamic link libraries.
- All platform files (such as Apache and Microsoft IIS).
- Third-party commercial libraries.
- Third-party common applications (such as SQL Server and Oracle).
- Vulnerabilities in configuration (such as ports open/closed, services activated and processes running).

The notion of "scanning" should also extend to identify malware and sensitive data, rather than just coding vulnerabilities. SRM leaders should ensure that encryption keys and other secret data are not accidentally embedded in the code.

Don't Expect to Use Traditional DAST/SAST Without Changes

In the previous section, we stressed the importance of identifying known vulnerabilities in known components, libraries and frameworks, but what about the 10% to 40% of the application that is indeed custom code? In DevSecOps, you should scan for unknown vulnerabilities in custom code. However, don't expect to use traditional static and dynamic application security testing tools and services without changes (see "Magic Quadrant for Application Security Testing"). These traditional testing solutions will need to either be refactored, retuned or replaced. Follow these steps:

Start with simple testing integrated directly into the IDE (see Note 3). Some developers prefer that these quick IDE-based security scan metrics not be tracked and reported, and this should be considered as a way to build the partnership with DevOps.

All scans should be fully automated and kicked off automatically via APIs triggered by the developer's CI/CD toolchain and process events like code commit. Third-party offerings (such as those from Cybric and BMC) can help provide the automation and orchestration of scanning.

The scans should not require a security expert to run, configure or interpret the results.

The scans should be tuned to reduce false positives, even at the expense of false negatives.

The results should be directly delivered into the bug tracking system or development dashboard, depending on the CI/CD toolchain and process used by the developers.

No matter which security scans you perform, SRM leaders should prioritize the vulnerabilities discovered for the developer by risk. Focus on reducing false positives and directing the developers with the highest confidence and severity vulnerabilities first. Several AST vendors are using machine learning to trim scan results for precisely this reason. Accept that lower-risk vulnerabilities are OK, and may be addressed either with runtime mitigating protection controls or in future iterations of the software.

As an alternative to traditional SAST and DAST, we recommend an IAST approach where possible. IAST is an improved form of DAST where the application being tested is instrumented during the test, providing both outside-in and inside-out visibility. By combining both, IAST approaches can provide a better balance of efficacy — the reduced false positives of DAST with the precise line of code and code coverage visibility of SAST. However, this approach is limited in platform support and requires that the application be assembled in a running state. If the IAST tool uses the development scanning as the "inducer," then this requires a solid (ideally automated) testing practice with full regression testing.

Some AST vendors provide testing as a service. This can work within DevSecOps, but only with tight service-level agreements (SLAs) on the turnaround time, combined with the lightweight, near real-time scanning within the IDE. For example, 50% of scans returned within four hours, 80% of scans within eight hours and 90% within 24 hours, with monetary penalties for not meeting these SLAs. Note that even these "tight" SLAs may be unsuitable for very rapid DevOps cycles. The vendor's testing services should be fully accessible via APIs so that a developer should not have to use a different application to "submit" code for testing or to retrieve results.

Train All Developers on the Basics of Secure Coding, but Don't Expect Them to Become Security Experts

Every person on the DevOps product team should have security training and, depending on the role, that training will vary. Developers get the most, testers get nearly the same and product owners will get less. You cannot make these team members security experts, but you can train all developers in the basics of secure coding. For example, a majority of application layer vulnerabilities could be addressed by the whitelisting of input and the filtering out of extraneous characters. The root cause of SQL injection and cross-site scripting is the lack of input sanitization. The same principle applies to input from data and configuration files, as well as network traffic.

The OWASP Top 10 Project and similar publicly available guidelines are a great start. [3] The training should include:

- How to build and maintain simple threat modeling scenarios (thinking like a bad guy)
- Input whitelisting, filtering and sanitization for user input and files
  - SQL injection
  - Cross-site scripting
  - Cross-site request forgery
- Injection
- Broken authentication and session management
- Unsecure direct object references
- Security misconfiguration
- Foundational security hygiene
  - Why not to embed keys or credentials in the application code or scripts
  - The importance of patching
  - How and why hackers will target admins for credential theft and how to avoid this

Ideally, the first training session is conducted in person, with periodic yearly training refreshed and reinforced with computer-based training modules. Also, if security issues are identified in development, this can be used to identify the need for further specific training for a developer or team. For example, if a critical vulnerability is found in a developer's code, they can be referred for further training on the issue and how to avoid it. More detailed application security training may be given to the security champion discussed in the next section.

Adopt a Security Champion Model and Implement a Simple Security Requirements Gathering Tool

In "DevOps Security Champions Help Organizations Gain Leverage Without Training Everyone," we recommended the adoption of a security champion model to leverage limited information security team resources into the development organization. The use of a security champion grants organizations an individual who can act as an on-site advisor and expert who can anticipate potential design or implementation problems early in the development process.

Security champions can reduce the perceived complexity of secure coding by providing immediate, real-world examples in the team's code and focusing on immediate remediation rather than more abstract, less relatable issues. For example, the security champion can be the first point of contact for questions on security requirements gathering and threat modeling for new applications with low to medium business risk. We recommend a simple security requirements gathering and threat modeling tool to make it as easy as possible for the developer (see Note 4). The goal should be self-service wherever possible. For the highest risk applications, we recommend engagement directly with the information security team via the liaison for full threat modeling and security requirements gathering.

SRM leaders should identify developers who have an interest in security and provide them with an expanded role as these champions. Their purpose will be to scale limited security training more effectively and provide a degree of security oversight in agile and DevOps teams.

Eliminate the Use of Known Vulnerable Components at the Source

As previously stated, most risk in modern application assembly comes from the use of known vulnerable components, libraries and frameworks. Rather than wait until an application is assembled to scan and identify these known vulnerabilities, why not address this issue at its source by warning developers not to download and use these known vulnerable components (and in cases of serious vulnerabilities, block the download)?

This highlights a process issue that should be addressed jointly by security and development architects — whether or not developers should be allowed to go directly to the public internet to download code. For some organizations, this risk is considered to be too high and developers are blocked from downloading these directly from the internet. For other organizations, this risk is managed by restricting users to managed code repositories on the public internet (such as GitHub).

However, these repositories hold older, known vulnerable versions of software that developers are able to download. To address this issue, some providers offer an "OSS firewall" (see Note 5) to expose the security posture of libraries to developers to make educated decisions about which versions to use. Using this approach, the developer can explicitly block downloads of components and libraries with known severe vulnerabilities (for example, based on the severity of the CVE assigned).

Organizations that don't allow developers to download code directly from the public internet will need to consider a secure code repository model. Here, the development organization in cooperation with the information security team builds, vets and maintains a component repository hosted internally. This type of approach requires an ongoing commitment between development architects and information security to keep it up to date, and a process for developers to request new frameworks and libraries. In addition, some organizations will mandate the use of preapproved, standard libraries for common security critical operations (such as authentication, authorization, key management, encryption, clickjacking and input sanitization) to further reduce risk.

Secure and Apply Operational Discipline to Automation Scripts

Do not neglect the infrastructure and the runtime platform in your control disciplines. Implementing infrastructure-as-code demands that source code controls apply to the infrastructure too. That includes version control on all software-defined items (such as configuration scripts and executables). These controls provide assurance that the correct version of a script is being used, platform control and configuration scripts should not contain secrets (such as credentials, keys or other vulnerabilities). It may be necessary to use tools specifically designed to scan for such secrets. [4]

Treat automation code, scripts, recipes, formation scripts and other such infrastructure and platform artifacts as valuable source code with specific additional risk. Therefore, use source-code-type controls including audit, protection, digital signatures, change control and version control to protect all such infrastructure and platform artifacts.

Containers and container management systems must have their change control records retained and their logging capabilities used to provide auditable assurance that known vulnerabilities have been excluded.

In short, there should not be changes without records. Period.

Implement Strong Version Control on All Code and Components

Use good source code version control throughout the life cycle, supported by the appropriate tools (such as distributed version control systems [DVCS] and application life cycle management [ALM] tools).

In high-velocity environments, it's important to capture every detail of the change, including the provenance of the code, what was changed, when it was changed and who changed it, together with any authorizations that may have been given (for example, by other product managers whose products could be impacted by the change). In most cases, this isn't an explicit permission model, but the philosophy of "trust and verify" applies, as there are ways to tell who calls this API or who uses this code. Being able to tell the difference between what code was used (versus what code is actually exercised in production), becomes a way to either harden the application in future iterations by removing unused components or to lower the risk if these components are later found to be vulnerable but aren't actually used.

As code moves from preproduction to production — from the left to right of Figure 2 — measure code digital signatures and be sure that the final build is the version that was last scanned. Don't instantiate code at runtime without verifying that it hasn't been tampered with since approved out of the preproduction phase/repository. (The configuration and assurance of the code is part of the "Prevent" capability in the upper right side of Figure 2.) Note that the platform operations team will use traditional change control techniques, including ITIL approaches, for the underlying infrastructure and platform provided to the product team.

Adopt an Immutable Infrastructure Mindset

For modern, Mode 2 micro services-based applications that are updated frequently, there is an operationally more secure way to maintain these applications. Where possible, no person should be allowed to make changes directly on production systems. The infrastructure is considered "immutable." When updates are needed (including security patches and configuration changes), these should be made back in development (the left side of Figure 2) and deployed by automated tools.

The libraries, components and OS images that the production images are generated from are kept in secure repositories. If a file needs to be patched, do it there rather than on the production system. Then the newly patched application can be deployed using the existing DevOps processes. This has multiple security and operational benefits. Most security and operational incidents have their root cause in some type of misconfiguration, mismanagement or missing patches.

From an operational and security perspective, many of the modern container and workload orchestration systems include automated health monitoring and alert upon deviation from the expected state. If the workload is behaving oddly or drifts too much, it can be removed from the pool of active resources and replaced with a workload spun up from a known good state. With proper application architecture, the individual workloads that comprise the service are ephemeral and can come and go while the service delivery is consistent and uninterrupted to the end user.

Further, from a security perspective, this mindset can be advanced and has the potential to radically improve security by proactively "killing" workloads and replacing them with versions from a known good state — even when they appear to be "healthy." The assumption is that a clever hacker might somehow gain a foothold on a system that is not detectable via normal mechanisms. By periodically and randomly killing the workload, the attacker's foothold is lost and unable to persist. This vision was described more than five years ago in "Systematic Workload Reprovisioning as a Strategy to Counter Advanced Persistent Threats: Concepts" and "Systematic Workload Reprovisioning as a Strategy to Counter Advanced Persistent Threats: Considerations," but the technology is now making this approach possible.

Evidence

[1] "Perfect Is the Enemy of Good." (https://en.wikipedia.org/wiki/Perfect_is_the_enemy_of_good) Wikipedia.

[2] "Equifax Releases Details on Cybersecurity Incident, Announces Personnel Changes." (https://www.equifaxsecurity2017.com/2017/09/15/equifax-releases-details-cybersecurity-incident-announces-personnel-changes/) Equifax.

Regarding Apache Struts:

> The attack vector used in this incident occurred through a vulnerability in Apache Struts (CVE-2017-5638), an open-source application framework that supports the Equifax online dispute portal web application.
>
> Based on the company's investigation, Equifax believes the unauthorized access to certain files containing personal information occurred from 13 May 2017 through 30 July 2017.
>
> U.S. CERT identified and disclosed the particular vulnerability in Apache Struts in early March 2017.
>
> Equifax's security organization was aware of this vulnerability at that time, and took efforts to identify and patch any vulnerable systems in the company's IT infrastructure.
>
> While Equifax fully understands the intense focus on patching efforts, the company's review of the facts is still ongoing. The company will release additional information when available.

[3] Example frameworks for basic security training:

> The "Top 10 Project." (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project) OWASP.
>
> "Guidelines for Writing Secure Code." (https://msdn.microsoft.com/en-us/library/ms182020(v=vs.100).aspx) Microsoft.

[4] "Truffle Hog Finds Security Keys Hidden in GitHub Code." (http://windowsitpro.com/security/truffle-hog-finds-security-keys-hidden-github-code) Windows IT Pro Magazine.

"Hacker Publishes GitHub Secret Key Hunter." (https://www.theregister.co.uk/2017/01/09/hacker_publishes_github_secret_key_hunter/) The Register.

Note 1
Software Composition Analysis Offerings

- ANX (Positive Networks)
- Black Duck
- Flexera Software (acquired Palamida)
- Lexumo
- Sonatype
- SourceClear
- Synopsys (acquired Protecode)
- Veracode
- WhiteSource

Note 2
Container Support

Note that containers are not required for DevOps or vice versa, but they are often used together to slipstream new services from development into production and back again. Ensure all of your security scanning and testing tools fully support containers in development and at runtime.

Note 3
Integrated Development Testing

Vendors that offer simple testing directly integrated into the integrated development environment:

- Checkmarx
- MicroFocus DevInspect (acquired HP Fortify)
- Synopsys SecureAssist (acquired Cigital)
- Veracode Greenlight
- WhiteHat Security

Note 4
Example Simple Security Requirements Gathering and Threat Modeling Tools

- Continuum Security
- foreseeti
- Microsoft Threat Modeling Tool

OWASP

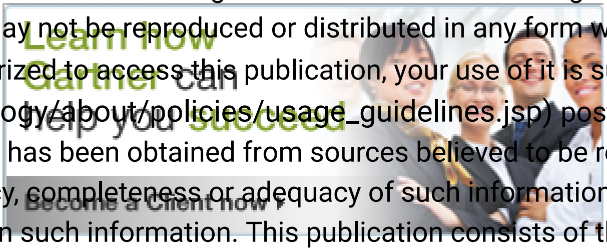
Security Compass

ThreatModeler

Note 5
OSS "Firewall" Example

Sonatype Nexus Firewall (https://www.sonatype.com/nexus-firewall)

© 2017 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. or its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. If you are authorized to access this publication, your use of it is subject to the Usage Guidelines for Gartner Services (/technology/about/policies/usage_guidelines.jsp) posted on gartner.com. The information contained in this publication has been obtained from sources believed to be reliable. Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information and shall have no liability for errors, omissions or inadequacies in such information. This publication consists of the opinions of Gartner's research organization and should not be construed as statements of fact. The opinions expressed herein are subject to change without notice. Gartner provides information technology research and advisory services to a wide range of technology consumers, manufacturers and sellers, and may have client relationships with, and derive revenues from, companies discussed herein. Although Gartner research may include a discussion of related legal issues, Gartner does not provide legal advice or services and its research should not be construed or used as such. Gartner is a public company, and its shareholders may include firms and funds that have financial interests in entities covered in Gartner research. Gartner's Board of Directors may include senior managers of these firms or funds. Gartner research is produced independently by its research organization without input or influence from these firms, funds or their managers. For further information on the independence and integrity of Gartner research, see "Guiding Principles on Independence and Objectivity. (/technology/about/ombudsman/omb_guide2.jsp)"



(https://www.gartner.com/technology/contact/become-a-client.jsp?cm_sp=bac-_-reprint-_-banner)

About (http://www.gartner.com/technology/about.jsp)

Careers (http://www.gartner.com/technology/careers/)

Newsroom (http://www.gartner.com/newsroom/)

Policies (http://www.gartner.com/technology/about/policies/guidelines_ov.jsp)

Privacy (https://www.gartner.com/privacy)

Site Index (http://www.gartner.com/technology/site-index.jsp)

IT Glossary (http://www.gartner.com/it-glossary/)

Contact Gartner (http://www.gartner.com/technology/contact/contact_gartner.jsp)