THE CISO GUIDE TO DEVSECOPS TACTICS AND TOOLS

How to embed sustainable security tools and tactics in your organization

Konstantinas Jurgilas

DEVSECOPS: INTEGRATING SECURITY INTO DEVOPS

DevOps is both a mindset and a methodology that eliminates barriers between development and operations. Conventionally, development (Dev) and deployment operations (Ops) have been at the core of the software development life cycle for modern development teams to build, test, and deploy software faster with higher levels of quality and a minimum of manual intervention. DevSecOps (development, security, and operations) shifts security further upstream in the development pipeline and aims to integrate security into all aspects of the life cycle—including design, implementation, testing, and deployment.

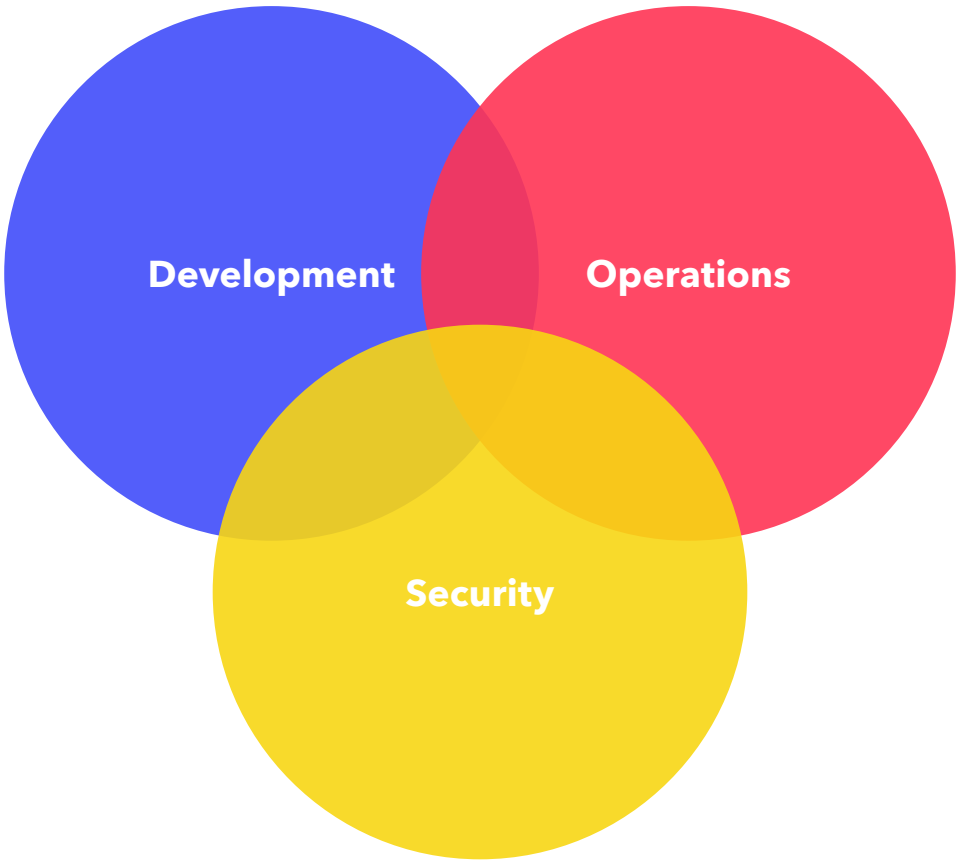
BUILD SECURE APPLICATIONS USING DEVSECOPS BEST PRACTICES

Embedding DevSecOps cultivates a mindset that the responsibility for security is to be shared among development, security, and IT operations teams. The aim is to build "software, safer, sooner," which is the DevSecOps motto. Delivery teams, customers, stakeholders all stand to substantially benefit from rapid, highly automated delivery of high-security software.

This paper presents an overview of implementing DevSecOps with tactical guidelines and tooling suggestions to embed the practice effectively in your organization.

WHAT IS DEVSECOPS?

The practice of DevOps includes integrated and multidisciplinary teams that work together to enable continuous software delivery. To improve velocity, developers constantly add open-source components to their projects. In most cases, there is potential to introduce security vulnerabilities and license compliance issues. DevSecOps (development, security, and operations) shifts security further upstream in the development pipeline, enhances other segments of the software development life cycle, and reinforces overall security. Indeed, DevSecOps aims to integrate security into all aspects of the life cycle, including design, implementation, testing, and deployment.

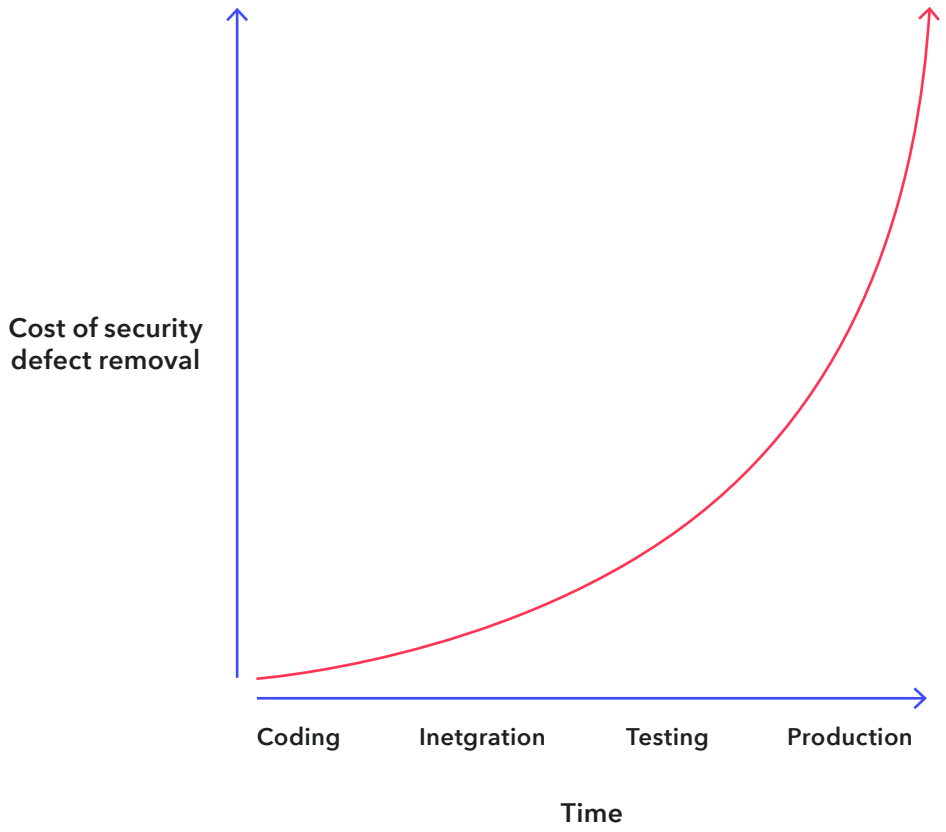


DevSecOps integrates development, operations, and security

WHAT ARE THE BENEFITS OF DEVSECOPS?

The DevSecOps model combines people, processes, and technologies to provide strong support for:

- An organization as it focuses on providing secure software and services.
- An organization as it seeks to reduce costs and effort by detecting and fixing security issues as early as possible in the life cycle.
- Teams to collaborate more efficiently and have maintain a clear, mutual understanding of how to ensure that the produced software is secure.
- Eliminating the practice of "throwing things over the wall" as each team member participates collaboratively in much of the software development life cycle.
- Sensible automation of many life cycle operations as automation prevents human error, reduces costs, and minimizes the number of manual, mundane tasks.



Remediation cost for defects increases sharply downstream in the pipeline

WHAT IS THE DIFFERENCE BETWEEN DEVOPS AND DEVSECOPS?

Because DevOps cultivates more efficient collaboration among development and operations teams, adopting the practices moves an organization toward rapid software development. Without DevOps, teams operate in silos, neglect mutual communication, and avoid participating in full ownership of the entire software development life cycle. Thoughtful professionals understand that such inefficiencies only hinder team morale and place limitations on business opportunities and potential value creation. These are the primary drivers for the rise of DevOps as a mature practice.

DevOps has become both a mindset and a methodology that eliminates barriers between development and operations. A modern development team to build, test, and deploy software faster—at higher levels of quality—without a minimum of manual intervention. DevOps helps unify developers, QA testers, system admins, support staff, and end users in sharing significant responsibility for the final product.

Conventionally, development (Dev) and deployment operations (Ops) have been at the core of the software development life cycle. However, security is becoming critically important as the multitude of threats increase, and severe breaches come at a high price to many companies. Enter DevSecOps.

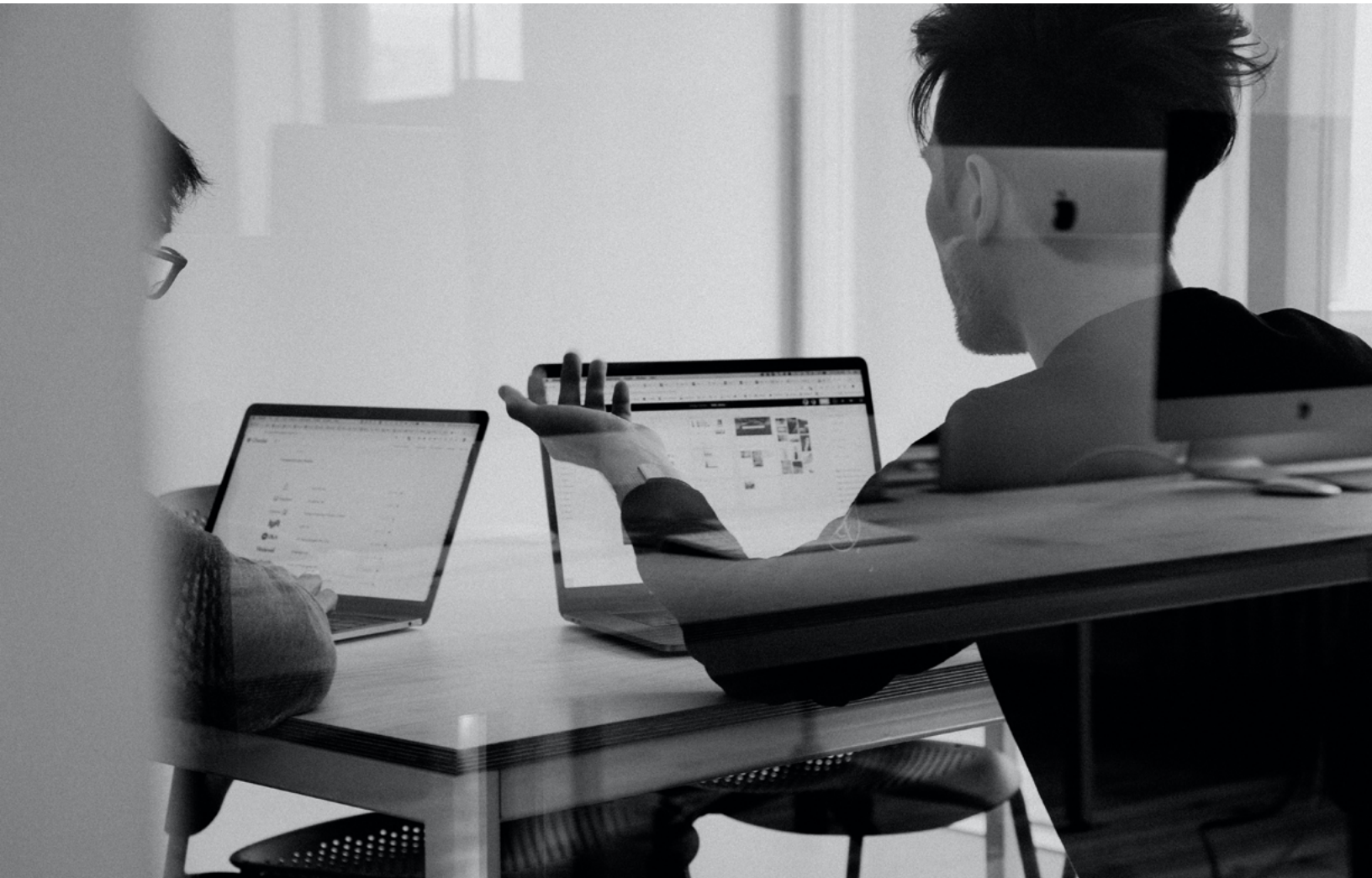
REALIZING THE NEED FOR SECURITY INTEGRATION

In the past, security testing was done at the end of the development cycle, with QA executing penetration tests and configuration checks just prior to a release. A security team tested a new version of the application or some infrastructure changes to ensure the app was secure and compliant. Development and operations delivered changes in bulk—with minimal documentation.

If a team has established DevOps practices, it is likely that the security team won't be able to maintain the rapid pace of a highly efficient DevOps team that deploys several releases a day. In such cases, security testing becomes a major bottleneck. Management will either accept this wasteful bottleneck or take the riskier path of dedicating less effort to security testing. Consequently, many teams have turned to DevSecOps.

DevSecOps integrates infrastructure and application security into DevOps processes and tools. The team manages security issues as they emerge—when it's easier, faster, and less costly to fix. Additionally, DevSecOps cultivates a mindset where the development, security, and IT operations teams share the responsibility for security. The collective aim is to build "software, safer, sooner," which is the DevSecOps motto. Delivery teams, customers, stakeholders all stand to substantially benefit from rapid, highly automated delivery of high-security software.

This paper presents an overview of implementing DevSecOps with tactical guidelines and tooling suggestions to embed the practices effectively in your organization.



HOW TO IMPLEMENT DEVSECOPS

DevSecOps enhances a software development life cycle by integrating security at every phase of the life cycle—design, implementation, testing, and deployment. The practice stands on two pillarsautomation and culture—both are necessary to implement DevSecOps and take advantage of all the benefits.

Augmenting DevOps with security practices requires tooling to automate and speed up software delivery. The same principle applies to security activities. Tools enable teams to automate repetitious tasks and make room for other meaningful activities (e.g., a security team chooses to automate penetration tests to allow more effort to be spent on threat-modeling activities).

For cases in which DevOps practices use infrastructure as code, DevSecOps analogously provides security as code. Employing security as code ensures:

- Security controls are versioned since each is stored within the version control system.
- Mutual acceptance of changes is done via peer reviews.
- Accountability is demonstrable since it is clear who made an adjustment and when a change is made.
- The risk of human error is reduced since the process of applying security controls has been automated
- Consistency is expected and configuration drift is eliminated because the same code is applied to different environments.
- A higher level of confidence is achieved since the security code can be tested to verify its correctness.

Tooling and automation alone are not enough to safeguard the efficient incorporation of security into the software development life cycle. The way people cooperate and work together is an equally critical component to ensure a smooth process.

To augment and integrate DevOps with security, it is vital to:

Treat security as a shared responsibility

Strongly encourage a high-security mindset in development, operations, and security.

Encourage continuous feedback and improvement

Everyone should be vocal about discoveries, insights, and recommending improvements.

Share knowledge regularly

Security experts should conduct frequent knowledge-sharing sessions, help to configure the tooling, and promote best practices.

Embrace transparency

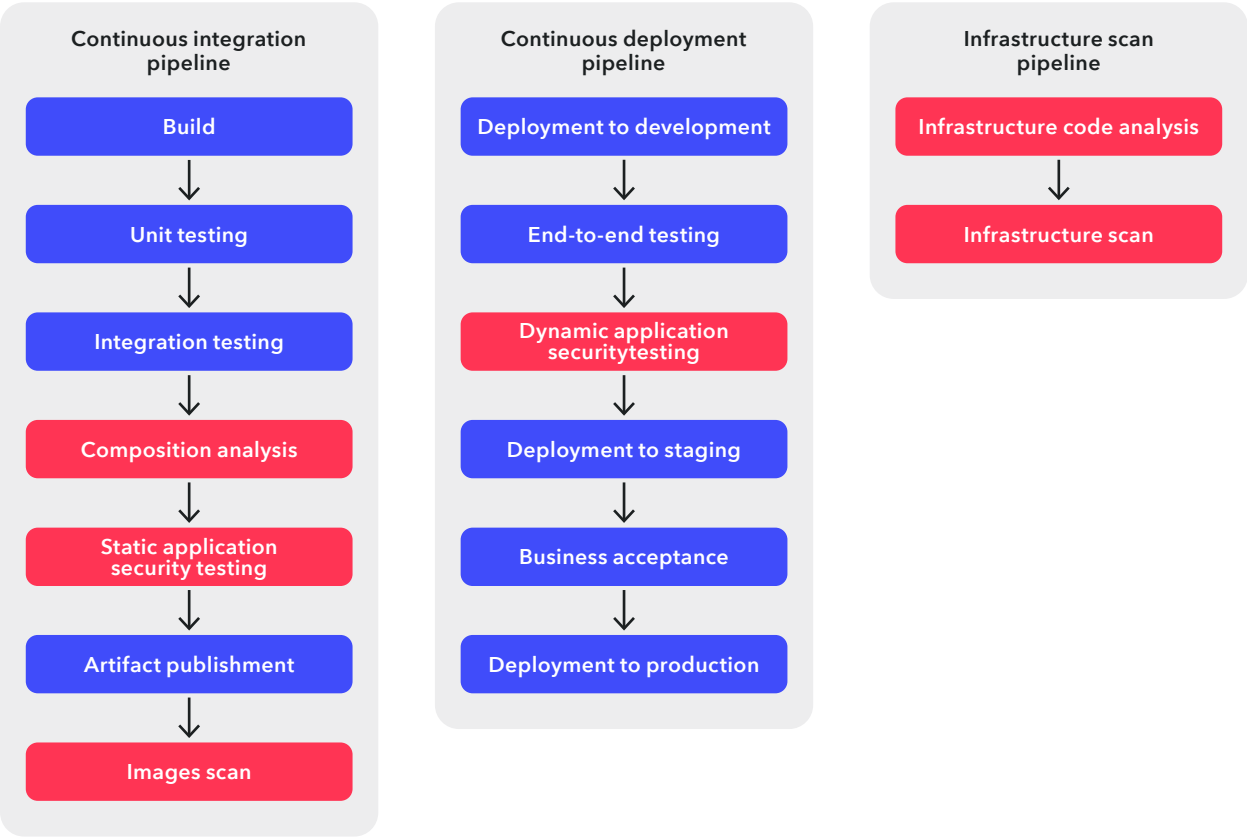
Open and timely communication is key to overcoming issues and blockers.

EMPLOYING DEVSECOPS SECURITY AUTOMATION TOOLS AND TACTICS

DevSecOps calls upon dedicated tooling to perform a variety of security checks. Supplementing DevOps with security introduces an additional analysis to scan the infrastructure on which the software runs. The continuous integration pipeline extends to include composition analysis, static application security testing, and scanning of images.

BUILDING A CLOUD INFRASTRUCTURE SECURITY PIPELINE

Explore cloud infrastructure issues in the article on Implementing perimeter security scans on cloud infrastructure. The contents focus on perimeter scans, available open-source tools, and the benefits of good tactics and successful use of the best tools. Use this knowledge to begin your journey toward a cloud infrastructure security scan pipeline.



CI/CD and Infrastructure scan pipelines

The following are recommendations for tactics and corresponding tooling for safeguarding applications with DevSecOps. Keep in mind that vulnerabilities found by detection tools usually include:

Severity
level

Links
to references

Explanation
of the issue

Recommendations
for remediation or repair

As you explore the tactics and tools below, keep in mind that DevSecOps analysis sequences do not cover all the activities and tests that are necessary to ensure the release product is secure. Other security-related activities, such as threat modeling and in-depth penetration testing must also be completed regularly.

TACTIC #1:
GIT SECRET SCANNING USING PRE-COMMIT HOOKS

A common software development practice is to store all the source code in a version control system that contains central repositories and verifiable assets for one or more development projects. A version control repository provides the ability to strictly control what changes individuals contribute to the source code and retrieve any changes made by the others.

DON'T:

Transfer sensitive data such as passwords, API keys, and cryptography secrets into a repository since such assets may be inadvertently accessible by unauthorized users. This is especially important for public repositories. Too often, for example, security breaches occur through leaked credentials in Github repositories when leaked AWS keys have been used to access protected resources.

DO:

Exclude sensitive assets when in any repository submission.

It's relatively easy to mistakenly commit sensitive data to the version control system. By contrast, removing committed data requires unwanted additional effort because the version control systems track all changes, including the removed code.

The best way to protect sensitive assets is to strictly avoid committing sensitive data. Version control systems provide pre-commit hooks that run custom scripts for specific types of actions. A pre-commit hook analyzes the change, rejects any commit that involves sensitive data, and warns the user.

THE TOOLING:
TALISMAN AND GITLEAKS

Talisman is an example of a pre-commit or pre-push hook for use in Git repositories that analyze any suspicious outgoing changes, such as authorization tokens and private keys.

```
$ git push

Talisman Report:

+----------------+-----------------------+
|    FILE        |                ERRORS                 |
+----------------+-----------------------+
| secret.pem     | The file name “danger.pem”                    |
|                | failed checks against the                     |
|                | pattern ^.+\.pem$                             |
+----------------+-----------------------+
| secret.pem     | Expected file to not to contain hex encoded texts such as:    |
|                | awsSecretKey=someSensitiveSecret              |
+----------------+-----------------------+
```

Alternately, repositories can be scanned for existing secrets using gitleaks.

TACTIC #2:
SOFTWARE COMPOSITION ANALYSIS

When contributing to a software project, a developer typically extends or modifies only a small fraction of the code base that would be executed at run-time. Even small applications typically employ third-party libraries which contain reusable functionality that developers do not have to write themselves. Composing software from reliable, reusable code improves development speed and enables quicker delivery of value to the end users.

Like any software asset, third-party libraries may contain security vulnerabilities. If a vulnerability is exploited, application functionality is disrupted and likely results in a security breach. Many software developers do not analyze and verify the third-party libraries in use on a project. When a new library is added, no review is done to examine any publicly disclosed vulnerabilities and available security improvements which are typically available for each version of the library.

THE CONSEQUENCES OF A DATA LEAK

A disastrous example of poor deployment of vulnerable components and libraries was the CVE-2017-5638 Apache Struts 2 framework vulnerability. This event led to the Equifax security breach that impacted 147 million consumers. A netmask npm module reported a server-side request forgery vulnerability. The library provided functionality to parse IPv4 CIDR blocks, and this vulnerability extended to over 278,000 other libraries that used the npm module.

The risk associated with third-party components is so pervasive that it is among the OWASP Top 10 Web Application Security Risks. Security experts suggest that the most efficient way to mitigate this risk is to perform continuous analysis of third-party libraries in parallel with the versions linked to the code base and check for publicly disclosed vulnerabilities. It is a vital best practice to analyze all project dependencies on every application build. If a high-severity vulnerability is detected within one or more third-party libraries, evaluate the impact, and take remediation measures.

The following are few notable issues and means to resolve each.

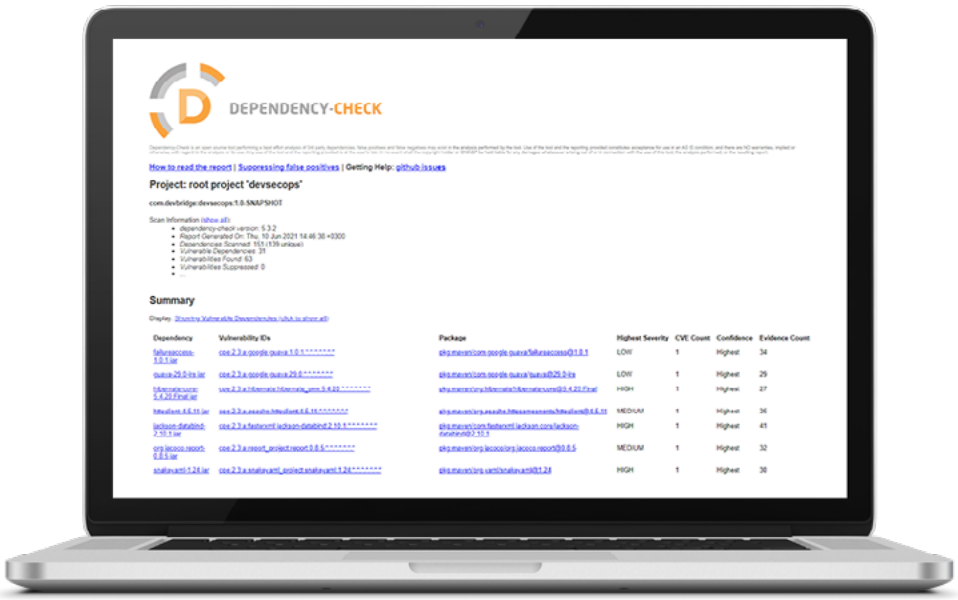
PROBLEM	SOLUTION
A vulnerable library has a security patch	Introduce a new patched version
A vulnerable library is used for the first time	Remove it from the code base
A vulnerable library does not have a security patch	Evaluate the risk and apply other remediations to reduce it

THE TOOLING:

OWASP, NPM, AND SNYK

There are a variety of tools to detect vulnerable third-party components. A few worthy options include:

- OWASP Dependency-Check detects publicly-disclosed vulnerabilities for third-party libraries that are part of a development project. The dependency check tool determines if there is a Common Platform Enumeration (CPE) identifier for each specific dependency. If the CPE identifier is found, the tool generates a report that links to the corresponding CVE entries. This tool primarily supports Java and .NET packaging.
- npm build tool is part of the npm ecosystem. The npm build tool includes built-in functionality to analyze dependencies. An npm audit command can be used to analyze dependencies against publicly-disclosed vulnerabilities.
- Snyk is a commercial tool that performs software composition analysis. The toolset supports a variety of dependency management tools, automatically creates pull requests to update vulnerable versions, and provides a web interface with powerful reporting capabilities.



TACTIC #3:

STATIC APPLICATION SECURITY TESTING (SAST)

Static application security testing (SAST) is a testing methodology that analyzes source code to find security vulnerabilities. The SAST approach analyzes the software implementation and the various execution paths to detect the most prevalent vulnerabilities, such as buffer overflows, SQL injections, cross-site scripting, and the usage of insecure cryptographic methods.

While it is feasible to manually perform SAST testing during formal code reviews, it's worth noting that any manual review involves the risks and inefficiencies of any human intervention. A best practice is to employ dedicated tooling that identifies the most common and obvious issues. In so doing, manual effort can be devoted moreso to the complex and composite issues.

THE TOOLING: SONARQUBE

SonarQube is a SAST tool that analyzes Java, PHP, C#, C, C++, Python, and JavaScript/TypeScript code against security vulnerabilities. It reports on security vulnerabilities and security hotspots. Keep in mind that manual review and approval are necessary to close a hotspot.

Credit: sonarqube

In SonarQube, quality gates are added to the continuous integration pipeline to enforce a quality policy by simply answering one question: Is the project ready for release? The quality gates encompass general metrics such as code coverage, code duplication, and security-specific density and severity of each vulnerability. If the density and severity exceed the quality gate threshold, the SAST step exits with an error code. Consequently, the entire continuous integration pipeline fails.

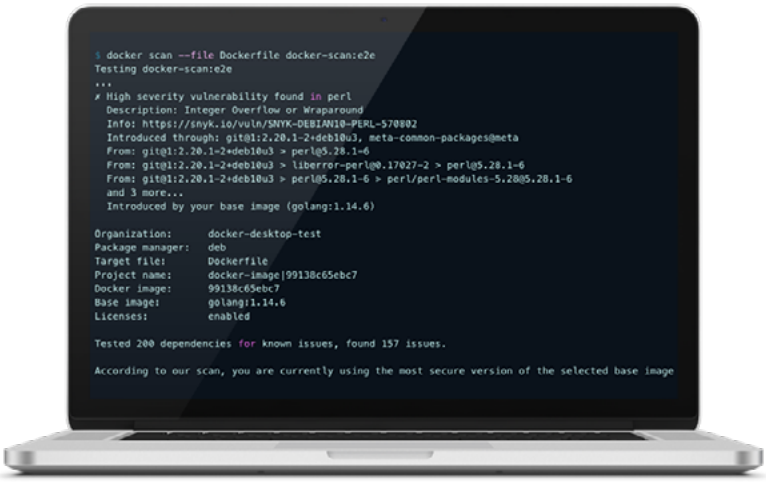
TACTIC #4:
SCANNING IMAGES

The popularity of containers continues to increase worldwide. Containerization is the type of virtualization that supports packaging software artifacts and dependencies into a single, standardized portable unit that runs on different cloud platforms, operating systems, and hardware. Containerized deployment is typically smoother since the container includes the run-time environment, including the application server and configuration. It is necessary to create an image that supports the template that will generate the container.

As with any software run-time environment, containers risk vulnerabilities. For example, consider a parent image containing an application server that harbors publicly disclosed vulnerabilities (CVEs). Like the risks of using third-party libraries, potential vulnerabilities arise from using insecure non-official parent images with similar or mistyped names in which dependent images also become vulnerable. As a result, it is important to analyze images to detect vulnerabilities.

THE TOOLING:
DOCKER, AMAZON ECR, AND CLAIR

- Docker is a popular containerization platform with built-in functionality to scan an image before pushing it to the image repository.
- Amazon Elastic Container Registry stores and scans images during a push.
- Clair is an open-source tool that scans OCI and Docker images against known vulnerabilities.



TACTIC #5:

DYNAMIC APPLICATION SECURITY TESTING (DAST)

While static application security testing uncovers many security vulnerabilities, it cannot detect run-time and environment issues. Dynamic application security testing (DAST) is available to analyze and identify such issues. DAST is based on a black box model in which analysis is performed on production software, and no knowledge of the underlying implementation is present.

Like penetration testing, DAST simulates a hacker approach for attacking the working application in an execution environment. DAST testing requires a solid understanding of how the application works and its primary use cases. The testing includes a variety of checks to verify that the software is resistant to vulnerabilities, such as fuzzing HTTP request parameters for SQL injections and analyzing HTTP response headers for misconfiguration.

THE TOOLING:

OWASP AND NIKTO

While there are many tools used to conduct dynamic application security testing, OWASP Zed Attack Proxy and Nikto are two popular toolsets.

OWASP Zed Attack Proxy (ZAP) is a web app scanner actively maintained by an international team of volunteers and one of the most popular OWASP tools. Penetration testers use ZAP as a proxy server, which permits full manipulation of all the traffic that passes through the proxy. ZAP provides automated scan functionality, which begins simply by entering the URL of a web application. The tooling proceeds to spider-crawl through the web application and passively scans each page it finds. The active scanner proceeds to attack each of the pages, including all functionality and parameters. ZAP also analyzes each HTTP response for vulnerabilities and misconfigurations—such as insecure Content Security Policy (CSP) or Cross-Origin Resource Sharing (CORS). The tool provides a report of all vulnerabilities, severities, and additional information.

Nikto is a command-line based web server scanner. Like OWASP ZAP, the tool accepts a web application URL and then runs the scan using different attack vectors, such as information disclosure, injections, and command execution. At completion, Nikto provides a report of all detected vulnerabilities and additional information.

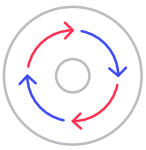
```
nikto -host devsecops.local -port 8080 -Plugins "@@DEFAULT;-sitefiles"

Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP: 127.0.0.1
+ Target Hostname: devsecops.local
+ Target Port: 8080
---------------------------------------------------------------------------
+ Start Time: 2020-12-05 18:13:20 (GMT2)
---------------------------------------------------------------------------
+ Server: nginx/1.17.10
+ X-XSS-Protection header has been set to disable XSS Protection. There is unlikely to
be a good reason for this.
+ Uncommon header 'x-dns-prefetch-control' found, with contents: off
+ Expect-CT is not enforced, upon receiving an invalid Certificate Transparency Log,
the connection will not be
dropped.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ The Content-Encoding header is set to "deflate" this may mean that the server is
vulnerable to the BREACH attack.
+ Retrieved x-powered-by header: Express
+ /config/: Configuration information may be available remotely.
+ OSVDB-3092: /home/: This might be interesting...
+ 7583 requests: 0 error(s) and 7 item(s) reported on remote host
+ End Time: 2020-12-05 18:16:32 (GMT2) (192 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

TACTIC #6:
INFRASTRUCTURE CODE ANALYSIS

Implementation of Infrastructure as Code (IaC) is an essential practice in DevOps, in which all logic assets and scripts for provisioning software run-time environments are stored in the version control system. Maintaining a repository of infrastructure code provides notable benefits, such as:

Cost reduction

The same code can be usedto swiftly provision many different environments.



Consistency

Environment drift is eliminated.

Accountability

All the changes are tracked, reviewed, and approved by peers.

As with any code, infrastructure code can be analyzed against different rules. Rules may be as simple as naming conventions. Some rules are advanced, such as those necessary to detect security issues and vulnerabilities, including:

Exposed server ports

Publicly accessible databases or files

And hard-coded secrets

THE TOOLING:
TERRAFORM AND TFSEC

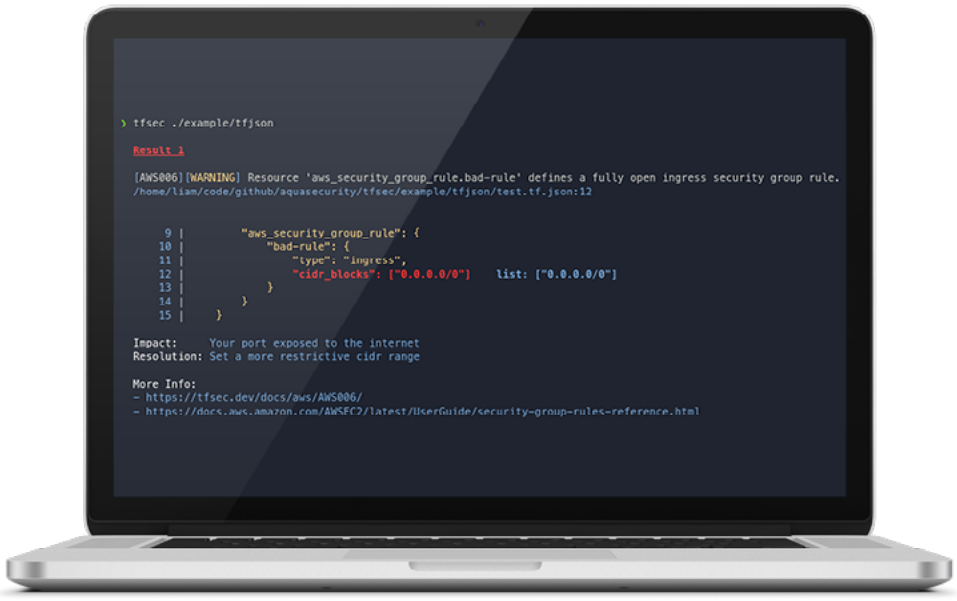
Terraform is a highly popular infrastructure as code tool that can declaratively specify any cloud resources, the parameters, and the relationships between different resources. After specifying all resources, the tool executes API requests to create, modify, and destroy the specified resources within several supported cloud providers, such as AWS, Azure, and GCP.

The tfsec tool can be used to check for sensitive data inclusion and violations of best practice recommendations across all major cloud providers to ensure Terraform scripts are secure. The tooling detects:

AWS S3 buckets with public ACLs

Publicly accessible AWS RDS databases

And usage of passwords authentication on Azure virtual machines



Credit: GitHub

TACTIC #7:
INFRASTRUCTURE SCANNING

Infrastructure as code analysis can be considered a static part of infrastructure testing because it analyzes only the code that creates the infrastructure—not the actual running infrastructure. Consequently, it is also necessary to perform dynamic infrastructure testing against the infrastructure in operation.

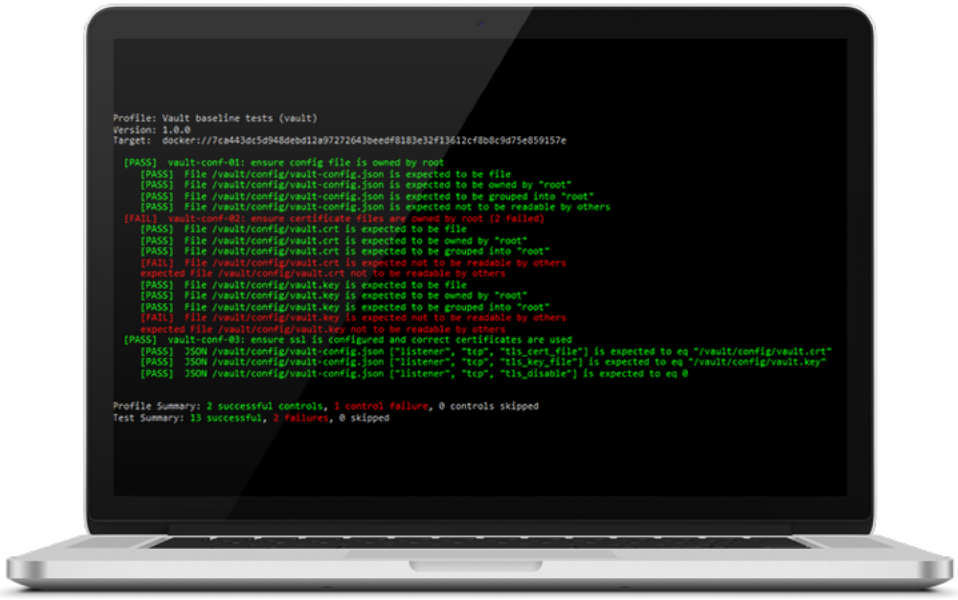
For certain sectors, the requirements for infrastructure compliance and auditing are quite stringent. The requirements can be internal (e.g., identified by the Chief Information Officer) or external (e.g., PCI DSS for financial organizations or HIPAA for health institutions). Typically, compliance testing and auditing are done with sustained, effort-intensive manual effort.

It is possible to automate testing to improve efficiency and minimize intervention errors. Many teams build a suite of automated infrastructure compliance and auditing tests that performs at any time: following changes to the infrastructure, upon auditor request, or on-demand.

THE TOOLING:

CHEFINSPEC AND DEVSEC HARDENING FRAMEWORK

Chef InSpec is an open-source framework for testing, auditing applications, and inspecting infrastructure. The tooling works by comparing the actual state of the system with the desired state that is expressed in easy-to-write Chef InSpec code. The tool verifies the desired state by using SSH or WinRM to connect to remote machines. The tool can integrate with cloud providers such as AWS, GCP, or Azure to audit any resources deployed in such environments. When the analysis is complete, Chef InSpec generates a report listing all violations. This tool supports the manual test creation that is useful when internal requirements need to be automated. Another option is Chef Supermarket, which contains various tests created by the development community.



The DevSec Hardening Framework provides a set of predefined controls called profiles for different applications and services.

Some examples include:

- **MySQL and PostgreSQL databases**
  An example is a profile that ensures any defaul passwords are changed
- **Apache and Nginx web servers**
  An example is ensuring that workers run as non-privileged users
- **SSH and Docker services**
  An example is ensuring correct file system permissions

The tooling provides both the profiles to detect violations and the resources for Infrastructure as code tools to resolve the violations. The framework currently provides:

Ansible Remediation Role

Chef Remediation Cookbook

Puppet Remediation Modul

TACTIC #8:

VULNERABILITY MANAGEMENT

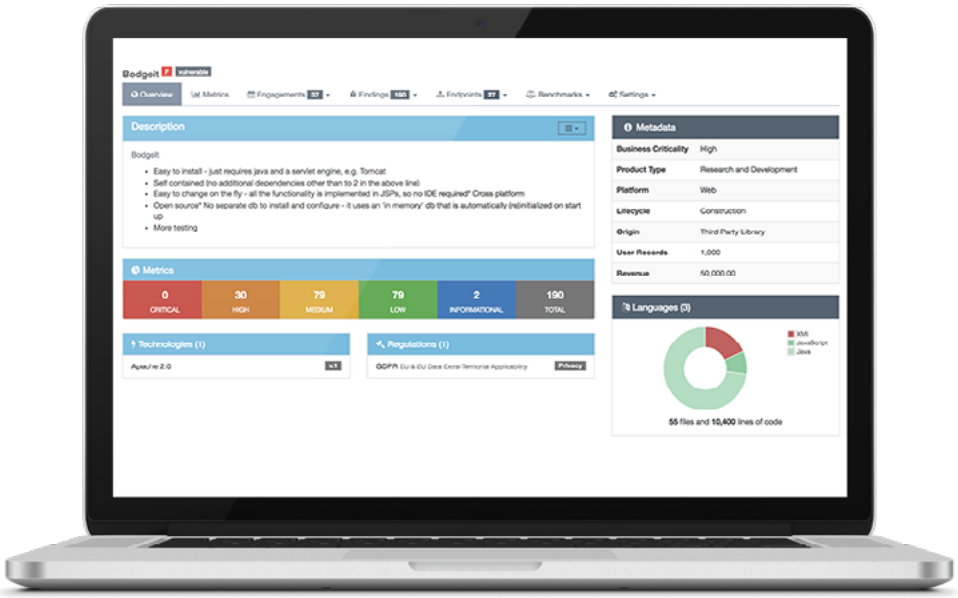
Vulnerability management is an integral part of DevSecOps, combining the results from all vulnerability checks in a central location. When tools report several vulnerabilities, it becomes easy to lose track of some issues. A single location for all detected vulnerabilities helps:

- Provide visibility into the overall status of security, such as how many vulnerabilities are present.
- Prioritize the order in which to remediate risks.
- Track historical data and tendencies.

THE TOOLING:

DEFECTDOJO, OPENVAS, NESSUS, AND FARADAY

DefectDojo manages application security program defects that maintain product and application information. This tool provides features for triaging vulnerabilities and pushes findings into defect trackers such as JIRA. DefectDojo consolidates findings from different security tools into a single repository. Over 85 scanner formats are supported.



Credit: GitHub DefectDojo

These are other notable vulnerability management systems.

- OpenVAS is a free software framework consisting of several services and tools that offer vulnerability scanning and vulnerability management.
- Nessus is a commercial tool for managing vulnerabilities and performing in-depth perimeter scans, penetration tests, and compliance checks.
- Faraday is a commercial tool that integrates vulnerability data from over 80 security tools, provides custom reporting templates, and integrates with ticketing systems, such as JIRA and ServiceNow.

10 SECURITY TOOL EVALUATION GUIDELINES

Extending the DevOps pipeline with security automation won't come easy. To avoid wasted effort and unnecessary frustration, proceed carefully as you integrate and automate security tooling. During tool evaluation, take care to follow these guidelines:

1. Tools used within the DevSecOps pipeline must have a command-line interface (CLI). If a tool doesn't have a CLI, then it cannot be added to automation servers that manage the DevSecOps pipeline.
2. Tools must provide a machine-parsable output, such as XML or JSON, so that other tools can parse and use the results in the vulnerability management system.
3. Mark and resolve false positives. Since invalid detections are likely to arise, staff should review and reject such findings to report only relevant information.
4. Include thresholds to specify the point at which specific pipeline step should fail. It is unrealistic to expect that the pipeline should fail when a single minor issue is reported.
5. Consider the requirements imposed on domains and set realistic thresholds. An example would be to permit no more than five vulnerabilities in total and not a single critical severity vulnerability.
6. The pipeline shouldn't run for an excessive amount of time. Aim to find an optimal set of tools and remove any duplications.
7. Pipelines should be parameterized to allow the option of specifying the level of detail in a scan.
8. Optimize the pipeline to skip steps. An example is to perform software composition analysis only when software dependencies change.
9. Occasionally, run the entire non-optimized pipeline to possibly capture potentially obscure vulnerabilities.
10. Be aware of licensing limitations for each tool you evaluate. There may be restrictions on parallel scans or threads that increase costs when scaling. Economize when sensible by using high-quality free and open-source tools.

BUILD SECURE APPLICATIONS WITH DEVSECOPS

DevSecOps, in practice, emphasizes the cultural change towards a shared responsibility for security and employs tooling to automate security testing. The implementation extends a conventional DevOps pipeline with security-specific steps and adds an infrastructure scan pipeline. It's important to realize that any security-related activities that are not included in the pipeline (e.g., threat modeling) must be performed separately on a regular basis.

As your organization embarks on its security automation journey, insist on abiding by these best practices:

- Employing pre-commit hooks reduces the risk of accidentally storing sensitive data in version control systems.
- Continuous software composition analysis reduce the risk of exposure to security vulnerabilities when using third-party libraries.
- Static application security testing often detects the most common security issues by performing code analysis.
- Dynamic application security testing mimics the hacker approach for attacking the running software in its execution environment.
- Analysis of infrastructure code detects infrastructure-specific security issues before they can arise in the actual environment.
- Infrastructure testing ensures that the running infrastructure is configured properly and meets the imposed compliance requirements.
- All the reported security issues are present within the vulnerability management system. Staff can use that information to learn the overall security status, prioritize the remediation of the issues, and track historical data.

Any movements to expand automation should consist of small, measurable projects. Success in these projects can be scaled and optimized to improve various other processes throughout the organization. As the drive for automation widens across business and IT operations, organizations that employ DevSecOps tools and practices can build a powerful foundation for digital transformation, modernizing applications, and maximizing efficiency in software development.