DEVADA DZone

BROUGHT TO YOU IN PARTNERSHIP WITH puppet

# Introduction to DevSecOps

CONTENTS

- Introduction
- Getting Started With DevSecOps
- DevSecOps Core Practices
- DevSecOps Additional Practices
- Whats Next for DevSecOps?

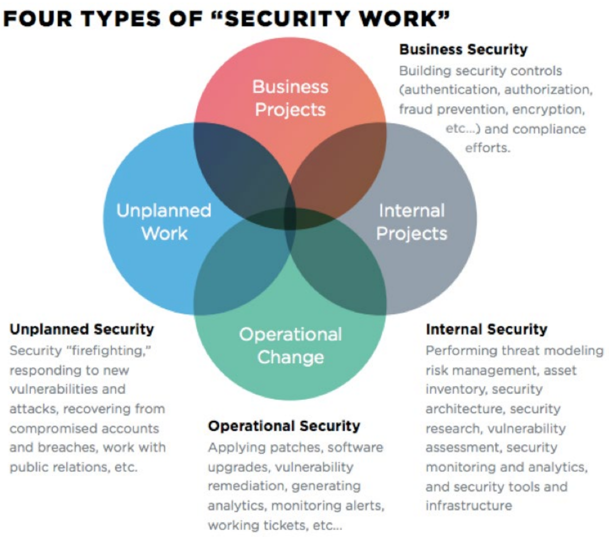
WRITTEN BY JEFF WILLIAMS
CO-FOUNDER AND CHIEF TECHNOLOGY OFFICER OF CONTRAST SECURITY

## What Is DevSecOps?

DevSecOps is an approach to IT security based on the principles of DevOps. The exact formulation is still emerging, but we think it's useful to capture emerging practices for achieving security while building applications and APIs without disrupting high-speed software pipelines.

- DevSecOps is full stack: DevSecOps spans the entire IT stack and includes network, host, container, server, cloud, mobile, and application security. Increasingly, all of these layers are turning into software, which makes application security a critical focus for DevSecOps.
- DevSecOps is full SLC: DevSecOps also spans the full software lifecycle, including development and operations. In development, the focus is on identifying and preventing vulnerabilities, while in operations, monitoring and defending applications are the goals.

Can you apply DevSecOps practices and tools to non-DevOps projects? Absolutely. The ideas in this document are applicable to almost any software project. If your goal is to produce highly secure software in the most cost-effective way possible, then DevSecOps is the path forward.


FOUR TYPES OF "SECURITY WORK"
Business Projects
Unplanned Work
Internal Projects
Operational Change
Business Security
Building security controls (authentication, authorization, fraud prevention, encryption, etc...) and compliance efforts.
Unplanned Security
Security "firefighting," responding to new vulnerabilities and attacks, recovering from compromised accounts and breaches, work with public relations, etc.
Operational Security
Applying patches, software upgrades, vulnerability remediation, generating analytics, monitoring alerts, working tickets, etc...
Internal Security
Performing threat modeling risk management, asset inventory, security architecture, security research, vulnerability assessment, security monitoring and analytics, and security tools and infrastructure

Gartner has named DevSecOps one of their fastest-growing areas of interest and predicts that DevSecOps will be embedded into 80 percent of rapid development teams by 2021.  Organizations practicing DevSecOps have shown impressive results. These early adopters are 2.6x more likely to have security testing keep up with frequent application updates and show a 2x reduction in time to fix vulnerabilities.

Understanding the different types of security work and their value to your organization is critical to successful DevSecOps initiatives. Until you truly understand the work, it's going to be difficult to deliver it effectively. You can learn more about this topic and DevOps in general by reading books like The Phoenix Project and The DevOps Handbook.

## Key DevSecOps Themes

Every DevSecOps program is a little bit different. It's best to view DevSecOps as a journey in time required to fix. As you progress, you may find that different teams are at different points along the path. The themes below aren't specific activities. Instead, they are guideposts that you can use to help make decisions along your journey.

puppet

# Say goodbye to soul-crushing manual tasks

**From automating simple tasks to securing global infrastructure, Puppet's products and open source projects can/will help you:**

- Enforce infrastructure configuration and compliance at scale
- Secure your infrastructure and fix vulnerabilities faster
- Deliver cloud-native infrastructure quickly and consistently

*Learn more at* **puppet.com**

| | |
|---|---|
| **Empowering Engineering Teams** | Development and operations are empowered to deliver secure applications into production themselves. Security experts provide support as coaches and toolsmiths, but do not have primary responsibility for security. Make sure that tools and processes are designed for developers and operations, not security experts. |
| **Making Security Visible** | In many organizations, security work is hidden, unknown, and untracked. In the end, the value of security is often not easy to understand. In DevSecOps, we make small security tasks that can be tracked, tasked, and measured like any other type of work. |
| **Shift Left** | Shifting security "left" means that security activities start during development and extend throughout the SLC, with continuous feedback at every stage from dev to prod. Shifting left does not mean that security is complete during development. |
| **Security as Code** | Like Continuous Integration and Continuous Deployment, Continuous Security means that you respond to continuous threats with security activities that are performed continuously, as part of the development and operations process, and integrated into the tools team members are already using. |
| **Continuous Security** | To address continuous threats, security activities are performed continuously as part of the development and operations process and integrated into the tools team members are already using. |
| **Prevent and Protect** | We will never produce perfect code. Nor will we ever detect or stop all attackers. Therefore, the best security strategies involve a balance of secure coding during development (DevSec) and runtime protection during operations (SecOps). |

## The "Three Ways" of Security

For decades, both software and security have struggled with poor quality results, cost overruns, and processes that require experts. While DevOps has shown promise on the software side, security is still being practiced in very traditional ways. DevSecOps is not just shoving traditional security practices and tools into DevOps.

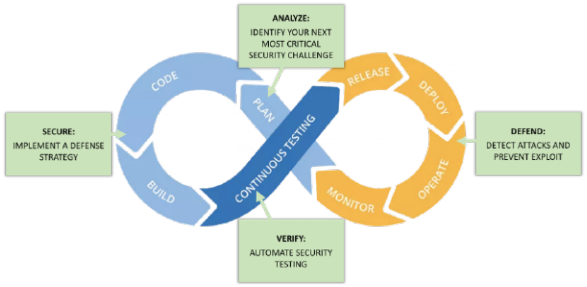
Instead, we must rethink the security work. We will need new practices and technologies to perform this work. We can give this transformation structure using the "Three Ways" from The Phoenix Project. By framing the problem this way, we can see that we need to get security work flowing, ensure instant security feedback, and create a security culture.

| | |
|---|---|
| **Get Your Security Work Flowing** | Most security work is monolithic and attempts to cover all risks in a single task, like a complete security architecture or security scan. How do we get security work flowing?<br>• Make the work visible.<br>• Work a single security challenge at a time.<br>• Limit work in process.<br>• Reduce handoffs.<br>• Automate everything. |
| **Ensure Instant Security Feedback** | Security is one of the most common causes of technical debt, and the cost of this work increases dramatically the farther it progresses across the SLC. How do we keep security work on track?<br>• Make problems instantly visible.<br>• Swarm on the problem.<br>• Seek the cause.<br>• Ensure security "findings" are designed for easy consumption. |
| **Encourage a Security Culture** | Many organizations have a security culture of blind trust, blame, and hiding that prevents developers and operations from working with security. How do we create a culture of security?<br>• Empower everyone to challenge security design and implementation.<br>• Take every opportunity to make security threats, policies, architecture, and vulnerabilities visible.<br>• Trust that engineering teams want to do the right thing.<br>• Celebrate the knowledge gleaned from security issues rather than blaming those involved.<br>• Spend more effort on upgrading practices and preventive measures than vulnerability remediation and incident response. |

If you follow these three ways, you will see security as a concrete output from your development process. It's a combination of security features and assurance, captured in a tangible way. By applying DevOps concepts, we can produce this concrete security continuously and effectively as a part of normal software development.

## Getting Started With DevSecOps

Traditionally, security has been performed as a series of massive tasks spanning all risks. For example, writing comprehensive security requirements, designing a comprehensive security architecture, performing a comprehensive security test, etc. But agility requires a risk-based approach. To accomplish security work in a DevOps organization, we can prioritize our security tasks and break them into small pieces for implementation.


ANALYZE:
IDENTIFY YOUR NEXT MOST CRITICAL SECURITY CHALLENGE
SECURE:
IMPLEMENT A DEFENSE STRATEGY
DEFEND:
DETECT ATTACKS AND PREVENT EXPLOIT
VERIFY:
AUTOMATE SECURITY TESTING
CODE
PLAN
BUILD
CONTINUOUS TESTING
RELEASE
DEPLOY
OPERATE
MONITOR

In this diagram, we show how security fits into the normal DevOps cycle at a very high level. Notice that these security augmentations are designed to fit naturally into the process. No extra steps, no gates, no delays. Instead, we will cycle quickly on small security tasks that are structured to be delivered by the development and operations teams using the tools they already use. We will explore these core practices in detail later.

### "HELLO, WORLD!" A FIRST STEP TOWARDS DEVSECOPS

Let's use a very simple example to demonstrate. Imagine that we have a web application being built by a DevOps project. All we know is that the application didn't do well on a recent security scan. There is no threat model or security architecture. How should we get started with DevSecOps?

| | |
|---|---|
| **1. Analyze**: Identify Your Next Most Critical Security Challenge | We're not going to try to secure everything at once. The most critical scan findings indicated that this application has obvious SQL injection problems. We want to make tangible progress quickly, so we decide to deal with this first. We're going to build our defenses and assurance over time in small pieces, not all at once. We create a JIRA ticket to address SQL injection. |
| **2. Secure**: Implement a Defense Strategy | Now we need to build our defense. Our strategy to prevent SQL injection is to use input validation and parameterized queries everywhere across our codebase. We may want to break this task up into even smaller pieces. We fill out the details of our defense strategy in our ticket(s) and start implementing them. |
| **3. Verify**: Automate Security Testing | We use simple tools to ensure that non-parameterized queries are eliminated from our codebase. We deployed a tool to warn developers in their IDE if they violate this rule. We also automatically re-verify during CI/CD as well as perform a final check prior to deployment. |
| **4. Defend**: Detect Attacks and Prevent Exploits | Finally, we want to be aware of any attackers that target us with SQL injection attacks. For visibility and protection in production, we implemented runtime application self-protection (RASP) and established a process for managing attacks. |

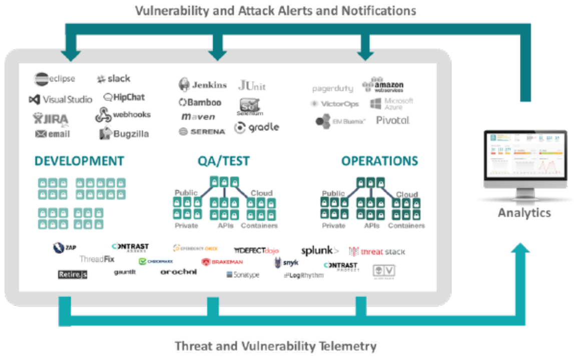
Once we are done with this security challenge, we can start in on the next one. Each time we complete a challenge, we leave behind the infrastructure to make sure it stays secure. Note that we don't want to create a fragmented set of individual defenses, so it's important to make good decisions over time about security architecture.

In DevSecOps, we don't let perfect be the enemy of good. We're looking to improve our security story on every iteration. Your first attempt might be a partial solution with sampling instead of a rigorous defense and complete testing. Our goal is to get this cycle running and make measurable progress over time.

### CREATING YOUR DEVSECOPS PIPELINE

A DevSecOps pipeline is the set of tools and processes that continuously performs security work as code is written, integrated, tested, deployed, and operated. While there's really just one delivery pipeline, having a security "view" of your pipeline may help you understand the security value stream separately, revealing bottlenecks and inspiring confidence in the results. The DevSecOps pipeline spans the full lifecycle, handling both development (vulnerabilities) and operations (attacks).

The goal of the security part of our pipeline is to provide vulnerability information to development team members in real time, through the tools they are already using. As we move into operations, our goal is to create visibility into who is attacking,what attack vectors they are attempting to exploit, what systems they are targeting, and whether the attacks are being prevented successfully.



The main cycle in the DevSecOps pipeline involves security tools, an analytics hub, and integrations with development and operations tools. The security tools at the bottom of the diagram identify vulnerabilities in applications and APIs across development, test, and production environments. In production, other tools monitor and prevent attacks. The telemetry from these tools feeds into an analytics system for historical tracking, analysis, and notifications. Common events include:

- Custom code vulnerabilities.
- Known vulnerabilities in libraries and frameworks.
- Attacks on custom code vulnerabilities.
- Attacks on libraries and frameworks.
- Application inventory, including all libraries and frameworks.
- Software architecture details.

In general, DevSecOps favors the use of notifications (real-time integrations into normal development and operations tools) over PDF reports. However, for some purposes, such as compliance, PDF reports may be generated. Notifications alert team members who need to know about security events immediately through the tools they are already using as part of their normal job. While a single analytics system would be ideal, today, you may need separate systems for vulnerabilities and threat events.

| | |
|---|---|
| **Sensors** | Think of all your security testing and attack monitoring as a set of sensors that is instrumenting your software development organization and systems. The best instrumentation runs continuously, has extremely high accuracy, and provides instant feedback. Sensors can be custom-built test cases, built-in rules in an analysis or protection technology, custom rules, etc. |
| **Security Analytics** | All the telemetry reported from sensors: inventory, libraries, vulnerabilities, attacks, etc. should be stored, measured, and tracked over time. You could start with a spreadsheet or a custom database, but it's better to use a tool designed to support a DevSecOps pipeline. The ideal analytics repository tracks issues (both vulnerabilities and attacks) over time, manages notification rules, and provides great reporting. |
| **Notifications** | DevSecOps is built on APIs and notifications. The goal is to get information about new CVEs, custom code vulnerabilities, misconfigurations, patches, probes, and attacks to the people that need them through the tools they are already using. This includes plugins and integrations for tools like Eclipse, IntelliJ, VisualStudio, JIRA, GitHub, Jenkins, Bamboo, Gradle, Maven, Splunk, ArcSight, PagerDuty, VictorOps, Docker, Kubernetes, AWS, Azure, and Pivotal. |

Implementing a notification infrastructure encourages downstream security stakeholders (developers, testers, operations, audit, executives, etc.) to work closely with upstream providers (like security testers) to ensure that the work is optimized for them. Your DevSecOps pipeline should be designed for very tight feedback loops — think seconds, not hours, weeks, or months. The faster you can get feedback to the people that need it, the more secure and cost-effective your DevSecOps pipeline will be.

When you start, your DevSecOps pipeline will only verify a few simple things about your software. But over time, as you address challenges, you will automate verification of more and more of your security strategies and defenses. Over time, the goal is to migrate from manual security testing to a fully automated pipeline capable of deploying secure code directly into production without gates.

### CHOOSING SECURITY TOOLS AND TECHNOLOGIES

Here are a few of the attributes to consider when choosing security tools and technologies to build your DevSecOps pipeline across the entire SLC. Please note that there is no one set of best tools for DevSecOps. The tools you choose should match the way that you build software, your goals, your culture, and the other technologies you use.

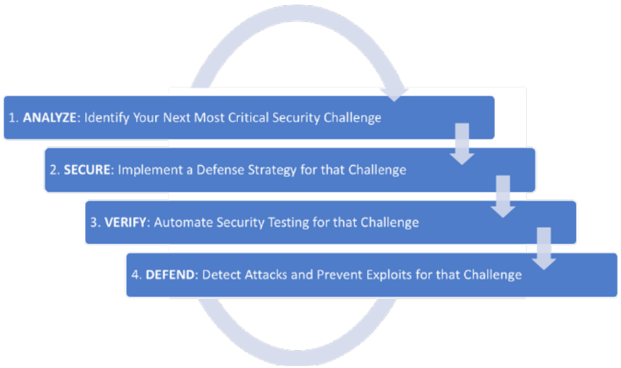
| | |
|---|---|
| **Policy Coverage** | First and foremost, you must confirm that the tool actually covers the risks you need it to cover. Many products have surprising shortcomings in this area. See the OWASP Benchmark Project for help. |
| **Accuracy** | Accuracy (eliminating both false positives and false negatives) is critical. Inaccuracy means humans have to fix results which will destroy your pipeline. You should carefully test your tools to be sure they accurately verify what you need. |
| **Speed** | You need to test whether tools are fast enough to work as a part of your DevSecOps pipeline. That may be microseconds, seconds, or minutes…but probably not hours and certainly not days. |
| **Scale** | Consider the size of your application portfolio and whether the tools you select are capable of operating continuously, in parallel, and across the entire portfolio. Be sure to factor in the number of people you will need to make that work. |
| **Process Fit** | You should verify that the tools are useful without a complex installation process. When well-meaning security folks buy tools for development and operations teams, it can cause friction if they aren't compatible with their technology stack or workflow. Engage the actual users in the evaluation process and conduct pilots to confirm they will be easy to install and use. |

| | |
|---|---|
| **Integrations** | Verify that the tool integrates with the tools that people in your DevOps toolchain are already using. Look for well-documented, supported REST APIs and SDKs in a variety of languages, IDE plugins, webhook support, ChatOps integrations like HipChat and Slack, notifications with PagerDuty and VictorOps, and SIEM integrations like Splunk. |

## DevSecOps Core Practices

DevSecOps takes a very agile approach to security, breaking down massive security tasks into incremental improvements that are performed as normal development tasks. These small batches of work include continuous verification so that security builds over time instead of repeatedly starting over from scratch.

Once we've identified the next security challenge, our normal engineering process can execute on the improvement. In this section, we explore four core practices to any DevSecOps initiative. Of course, your DevSecOps process might be considerably more complex. See the next section for more ideas or add your own practices to this basic cycle.


1. ANALYZE: Identify Your Next Most Critical Security Challenge
2. SECURE: Implement a Defense Strategy for that Challenge
3. VERIFY: Automate Security Testing for that Challenge
4. DEFEND: Detect Attacks and Prevent Exploits for that Challenge

Fundamentally, it's the constant tension between creating defenses and attempting to break them that actually makes organizations more secure. The faster you can repeat this DevSecOps cycle, the faster you can improve security. Over time, you'll build a complete security story that will provide assurance both internally and externally.

### 1. ANALYZE: IDENTIFY YOUR NEXT MOST CRITICAL SECURITY CHALLENGE

Why should you always focus on your most critical security challenge? Generally, working on anything else won't change your security posture very much. It doesn't help to close the attic window when the garage and front door are wide open. In DevSecOps, we get the work flowing by creating small batch sizes. So, in most cases, we want to work on the most critical security challenge to our enterprise first. Still, don't be afraid to choose a partial measure or a tiny improvement to your most critical challenge. Working in small increments makes sure we stay on track.

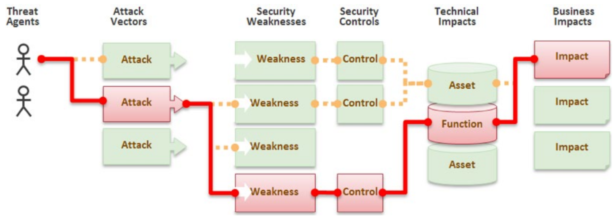
When deciding what to work on next, the team looks at all the potential security "work" available and makes it visible. The team might add new features, pay down some technical debt, make an architectural improvement, fix defects/vulnerabilities, or do something to improve the team's tools or practices to improve quality, security, or productivity.

It's important that the team use their threat model and security architecture to make an informed decision about the next most critical security challenge. What is the cost to the company of certain kinds of attacks? What is the cost of implementing preventive measures for those attacks? Try to use data from both internal and external sources to figure out the next thing to do that will most effectively reduce risk.

You may find yourself without a threat model or security architecture. Fortunately, in DevSecOps, these artifacts are created one step at a time. When you're starting out, it's easy to identify your top challenges: the problems that are the most likely to be found, be exploited, and cause serious damage like injection, authorization problems, known vulnerabilities, etc. Consider all the different layers of your application stack:

| | |
|---|---|
| **Applications and APIs** | Do you have proactive controls in place? Are you susceptible to common vulnerabilities? |
| **Libraries and Frameworks** | Are your libraries and frameworks up-to-date and properly configured? Do you have a complete up-to-date inventory (with exact version numbers) of all the software you are running across all your servers? |
| **Container and Cloud** | Have you hardened your platform configuration and kept it up-to-date? Are your cloud environments configured correctly? |
| **Network** | Do you have strong network security defenses in place and monitored for attacks? |

Use a risk-based approach to decide what to work on next. Be sure to consider whether there's a viable connection between a threat agent, attack vector, weakness, technical impact, and business impact in your enterprise. OWASP depicts this connection as follows:



Going forward, you should practice a combination of threat intelligence and security research to continuously zero in on your next most critical security challenge. Note that this process is significantly different from trying to assess all your threats. Many organizations get overwhelmed trying to protect against everything at once.

| DevSecOps Threat Intelligence, Security Research, and Security Architecture | |
|---|---|
| **Threat Intelligence** | **External sources**: ISACs (STIX/TAXII), OWASP, SANS, BlackHat, DefCon, LASCON, DevSecCon, CISO events, friendly peer companies, etc.<br>**Internal sources**: Monitor your systems for attacks and learn from the data. Understanding actual attacks is a major factor in prioritizing. |
| **Security Research** | Security research should focus on challenging security architectures and identifying new strategic ways to improve security. Where possible, work with development to enhance the DevSecOps pipeline with new testing methods. |
| **Security Architecture** | There is a dearth of great threat modeling and security architecture tools. But some interesting projects include OWASP Threat Dragon, IriusRisk, and Chaos Engineering (Chaos Monkey, ChaoSlingr). |

You'll have to work out your own cadence for re-examining your threat model. So, increasing your cycle speed will have a direct effect on the level of security you are able to achieve. And because DevSecOps adds to your security incrementally with continuous security verification (discussed below), you have protection against backsliding.

### 2. SECURE: IMPLEMENT A DEFENSE STRATEGY

Once you've decided on a security challenge to tackle, you'll need to choose a defense strategy. A defense strategy isn't a single security mechanism or product. A defense strategy can combine technical security mechanisms, secure coding practices, procedural controls, supporting processes, training, background checks, and more. We are using the term "defense strategy" broadly to include anything that you rely on to prevent a breach. Your defense strategy for a particular challenge can (and probably should) comprise one or more primary defenses and a set of supporting defenses as well.

You can capture each security strategy for implementation in a JIRA ticket that covers each security enhancement that you want to make, including:

| | |
|---|---|
| **Challenge Description** | The goal is to concisely justify why this is an important security challenge. This might take the form of a security story or misuse case. The description should cover the elements shown in the OWASP diagram above. |
| **Defense Story** | This story should detail exactly how the defense should work. For technical defense mechanisms, the story should clearly detail how the threat is countered and why this defense is effective. For other defenses, how they work to provide protection should be argued. |
| **Operational Guidelines** | As part of the defense strategy, you should also consider how to configure, operate, and use these defenses. This guidance should apply to end users and operations staff, and even indicate how developers should use the defense effectively. |
| **Security Testing Approach** | The final part of the defense strategy is to detail how you will continuously verify the correctness and effectiveness of the defense strategy. |

Your strategy is right when you can easily answer with confidence when anyone asks, "How do you protect against X?" Having a clear, concise, defensible answer to this kind of question can not only provide an easy path to compliance but can also provide business advantage over competitors.

Your defense strategy doesn't have to be perfect from the very start. It's far better to start with a basic defense and evolve it over time. In fact, after you implement a basic defense, you may choose to work on another, more pressing, threat. The key to DevSecOps is to continuously reprioritize based on the threat and your existing defenses. The ability to respond quickly is critical for a world of continuously changing threats.

The work of actually implementing the security defense shouldn't be any different than any other feature, and should, to the maximum extent possible, be delivered as code or configuration with everything in source control. Managing security in this way makes it possible to test and redeploy at any time, ensuring that defenses are in place, working correctly, and properly configured.

### 3. VERIFY: AUTOMATE SECURITY TESTING

A key part of DevSecOps is ensuring that the defense strategies have been properly implemented, configured, and operated. Security testing is the way to verify that your actual security controls match your intended defenses. In DevSecOps, we focus on automating those tests by "turning security into code" so that we can run them frequently without requiring humans, particularly security experts, in the critical path.

There are many ways to automatically verify the security of a system. There is no possible way to list them all, but we provide a few examples of popular tools that have proven themselves to be DevSecOps compatible.

| | |
|---|---|
| **Automate Everything** | It's not automated if you need humans in the loop. Don't fall into the trap of thinking that you've automated security when all you really did was automate the "scan" button. Think about the entire process. Does the tool require human expertise to configure or run? Does it require an expert to interpret and triage the results? We are looking to eliminate the involvement of humans in the critical path so that we can push code to production with both speed and assurance. |

| | |
|---|---|
| **Avoid "Tool Soup"** | Every tool that you adopt means additional process steps, a full set of integrations, and a team of people to configure, run, and interpret. Choose powerful platforms that will allow you to address many different types of security challenges using an integrated framework. |
| **Test Your Testing Tools** | Security testing tools vary greatly in their ability to test real applications for a broad range of issues. The only way to know how well a particular tool will work on your applications and APIs is to try it. Consider temporarily adding "tool canaries" in your applications to verify that real vulnerabilities are being discovered and false alarms are not being flagged. See the OWASP Benchmark project for details. |

Below is a DevSecOps security testing funnel to help you choose a security verification technique for the particular security challenge we are working on. This may seem obvious, but don't blindly rely on the wrong tool. Take a minute to select the simplest, fastest, most accurate way to check that your defense implementation is correct, complete, and configured.

For example, testing for proper clickjacking protection is easy if you simply examine HTTP responses for the proper security headers. But it would be very difficult to verify this by looking at the source code, as there are so many ways that this might be accomplished.

| **DevSecOps Security Testing Funnel** | |
|---|---|
| **What are we trying to test?** | Think carefully about exactly what you want to test and the results you want. Direct, complete verification of application behavior is always best, but you can use sampling, fuzzing, design analysis, and other techniques. |
| **Do we need to test it?** | Probably, yes. But you should clearly prioritize things that are the most critical to security and the most likely to be discovered and exploited by an attacker. |
| **Positive or negative testing?** | Are we able to verify that the application always follows a known good pattern of execution (positive security) or will we have to resort to trying to verify that the application never follows any of the known bad patterns of execution (negative security)? |
| **Do we already test for it?** | You may be using a security testing tool that covers this risk. It's critical to verify that your tool does a good job of accurately and efficiently verifying your security defense. |
| **Do we already have a platform that will allow us to test this easily?** | Perhaps you just need to enable a rule in a tool you're already using. Or maybe you can use an existing tool as a platform for creating a custom rule. |
| **Can we test it by writing custom tests?** | If we can't use a security testing platform, can we create a custom test case? |
| **Is there another tool on the market that can help test this?** | **Network:** nmap, sslyze, ssh_scan, Tenable, Qualys.<br>**Cloud/container:** Aqua, Twistlock, Redlock, ThreatStack.<br>**Libraries/frameworks:** OWASP Dependency Check, retire.js, Contrast Assess, Snyk, Sonatype, BlackDuck.<br>**Application:** OWASP ZAP, Arachni, sqlmap, Burp, Contrast Assess, Micro Focus, CA Veracode, Synopsys, Checkmarx. |
| **Do we need human experts to test it?** | While the goal of DevSecOps is to minimize the number of things that you need human experts to test, bug bounties bounties, red team exercises, and manual penetration testing can provide useful insight into defenses that are difficult to test automatically. Ensure that these efforts deliver rules, test cases, and other automation, not PDF reports.<br>**Examples**: BugCrowd, HackerOne |
| **Are we also testing for what we didn't think of?** | It's impossible to know everything, so certain kinds of testing rely on volume and randomness to try to force applications to behave badly. Fuzz testing and chaos engineering tools can help here. |

Any security issues discovered during testing should feed into the DevSecOps management infrastructure described above to notify all the people that need to know through the tools they are already using.

### 4. DEFEND: DETECT ATTACKS AND PREVENT EXPLOITS

DevSecOps organizations recognize that you can never test yourself secure, so, they adopt a balanced approach that focuses on minimizing vulnerabilities during development and on identifying and preventing vulnerabilities from being exploited in production. While these two activities have traditionally been separate, DevOps has brought them together and DevSecOps projects support the full software lifecycle.

| DevSecOps Attack Detection and Prevention Tools | |
|---|---|
| **Runtime Application Self Protection (RASP)** | RASP uses application instrumentation to add attack detection and prevention directly to applications regardless of where or how they are deployed. DevSecOps projects can use RASP to achieve accurate attack-blocking and the flexibility to easily deploy in cloud/container environments.<br>**Examples**: Contrast Protect, Prevoty, Immunio. |
| **Web Application Firewall (WAF)** | WAFs have been on the market since the early 2000s and have a history of complex configuration and spotty protection. Nevertheless, a DevSecOps project might be able to use a WAF for basic protection or as a platform for virtual patches.<br>**Examples:** ModSecurity, Imperva, F5, Signal Sciences. |
| **Network Intrusion Detection and Prevention (IDS/IPS)** | There are a variety of network-, container-, and host-level protections against attacks. Seek out products that can be deployed and managed as part of your standard technology stack.<br>**Examples:** Snort, Suricata, Bro, Kismet. |
| **Security Information and Event Management (SIEM)** | SIEM tools provide real-time analysis of security alerts generated by applications and network hardware and are important to handling attacks in DevSecOps.<br>**Examples:** Splunk, ELK, SumoLogic, LogRhythm, ArcSight. |

The threat and attack data gathered should feed directly into the next DevSecOps cycle to be used by security research to help choose the next most critical security challenge.

## DevSecOps Additional Practices

There are additional sets of security challenges that emerge when an enterprise has hundreds or thousands of applications in their portfolio. Doing security at this scale is far beyond what a small dedicated security team can accomplish. DevSecOps is a technique for distributing this work effectively across development and operations.

It's worth noting that in most organizations, only a small percentage of projects are very far along in their DevOps journey. So, managing the transition to DevSecOps across an entire application portfolio is a key part of the challenge.

### STANDARD DEFENSES AND ENTERPRISE SECURITY ARCHITECTURE

Generally, DevSecOps organizations — particularly those with large application portfolios — prefer standard security defenses that are heavily tested for correctness and effectiveness. Don't reinvent authentication, authorization, encryption, etc. for every application and API you build. This reduces the amount of security work that has to be done and simplifies the organization's security architecture. Achieving effective enterprise security architecture in a DevSecOps manner is beyond the scope of this document, but these guidelines are good first steps.

| | |
|---|---|
| **Security Libraries** | You should generally use popular, well-tested libraries for security features, as it is dangerously easy to make small but disastrous mistakes writing your own. Probably the best approach here is to assemble your own enterprise security API that implements, extends, wraps, or is based on existing security libraries like Spring Security, OWASP ESAPI, BouncyCastle, OWASP Encoder, AntiXSS, AntiSamy, Jasypt, etc. |
| **Standardize on a Standard Software Stack** | To the highest extent possible, you want a standard approach to security, and you should use the security features provided by your software stack. This makes security invisible or automatic and reduces the likelihood of mistakes. Don't assume they're working, though; you have to continuously test these defenses. |
| **Security Services** | Consider the extent to which you can turn security defenses into high assurance services that can be invoked by your applications. This creates the possibility of upgrading security across many applications without having to recode, retest, and redeploy. In the spirt of DevOps, make sure there is a self-service way for your empowered engineering teams to consume them without needing central approval or provisioning. |

### MANAGE THE SOFTWARE SUPPLY CHAIN

Libraries have serious vulnerabilities, most of which have not yet been discovered by the good guys. And attacks now start within a day or so of new vulnerabilities being disclosed. So, every DevSecOps project must pay attention to the security of their supply chain.

Your software stack is composed of thousands of libraries, frameworks, modules, and components written by unknown developers all over the world. Using this software can allow you to create software much more rapidly, but the cost is that you must take full responsibility for the security of all that software.

In practice, this translates to a few best practices:

| | |
|---|---|
| **Show Some Restraint** | Be cautious about the libraries and frameworks that you adopt. Stick to projects that demonstrate a solid security program, evidence of security testing, and effective response to new vulnerabilities. |
| **Self-Inventory** | DevSecOps organizations must know the exact version of every library, framework, application, and API that is running on every server in every environment. The best approach is to enable all your systems to self-inventory by reporting their "bill of materials" to a central database. This "always up to date" inventory will enable you to respond quickly to novel attacks. |
| **Establish Secure Coding Guardrails** | When you decide to take on a new library, standardize how you will use it safely (both positive and negative rules) and turn them into code so they can be continuously tested. |
| **Test for Latent Vulnerabilities** | Before you trust your business to someone else's code, you must verify that the defenses that are provided by the libraries and frameworks work as advertised and don't contain undiscovered vulnerabilities. Very few libraries receive adequate security testing. |
| **Continuously Monitor for New Vulnerabilities and Respond** | Modern applications are only fully composed at runtime, as their dynamic dependencies, plugins, and injections are fully realized. RASP tools (discussed above) can proactively prevent both known and unknown vulnerabilities from being exploited. |

### EXPLODE, OFFLOAD, RELOAD

While it's not strictly necessary, many DevOps projects use the cloud, containers, and APIs. Many organizations have already discovered that this is, in fact, the fastest path to securely achieving digital transformation. DevSecOps projects should strongly consider Ed Amoroso's advice to "explode, offload, reload."

| | |
|---|---|
| **Explode** | Perimeters are no longer effective at stopping attacks, so the first step is to break up your monolithic internal infrastructure into smaller distributed workloads. |
| **Offload** | Second, take advantage of security and cost advantages by moving these new, smaller workloads to virtual cloud and container infrastructure. |
| **Reload** | Finally, re-establish continuous security for each of your newly virtualized workloads by selecting and deploying modern protection technology at the application, library, container, and network layers. |

This strategy is very compatible with DevSecOps, as it allows for efficient centralized security management with distributed enforcement for speed and accuracy.

### CREATE A SECURITY CULTURE

Some companies simply seem to have the ability to take security seriously, focus, and do a great job. But others — even companies that seem to be doing all the same practices and using the same tools — are simply terrible at security. The difference is culture. And while culture is a difficult thing to change, there are a few key practices that have worked in organizations and should be part of a DevSecOps program.

| | |
|---|---|
| **Executive Sponsorship** | You must have support from the executive level. They need to make it clear that security is everyone's responsibility and that simply getting past the compliance audit is not the goal. |
| **Micro-Training** | Everyone needs to understand exactly what their security responsibilities actually are. The best way to achieve this is by providing instant feedback while they are doing their job. |
| **Accountability** | Development and operations team members should be responsible for the security of what they produce and operate. Security specialists shouldn't be in the critical delivery path and should act as coaches and toolsmiths instead. |
| **Security in Sunshine** | Make security as visible as possible. Be sure that vulnerability and attack data is never used to shame people. Instead, celebrate security risks as the fastest path to learning and improving. Only when security is visible can you achieve a culture of making informed risk decisions. |

## What's Next for DevSecOps?

DevSecOps is still in the formative stages. The best way for you to get involved is to try implementing DevSecOps in your own organization and publish your experiences. Here are some sources of additional information.

- amazon.com/Phoenix-Project-DevOps-Helping-Business/dp/0988262592
- https://www.youtube.com/watch?v=cIvOth0fxmI
- dzone.com/articles/a-devops-approach-to-building-security
- contrastsecurity.com/continuous-app-sec-cas
- linkedin.com/pulse/lose-security-wheel-edward-amoroso
- ruggedsoftware.org/
- ca.com/content/dam/ca/us/files/msf-hub-assets/research-assets/integrating-security-into-the-dna-of-your-software-lifecycle.pdf
- whitehatsec.com/news/12th-annual-application-security-statistics-report

Written by **Jeff Williams**

Jeff brings more than 20 years of security leadership experience as co-founder and Chief Technology Officer of Contrast Security. Previously, Jeff was co-founder and CEO of Aspect Security, a successful and innovative application security consulting company acquired by EY. Jeff is also a founder and major contributor to OWASP, where he served as Global Chairman for 8 years and created the OWASP Top 10, OWASP Enterprise Security API, OWASP Application Security Verification Standard, XSS Prevention Cheat Sheet, and many other widely adopted free and open projects. Jeff has a BA from Virginia, an MA from George Mason, and a JD from Georgetown.

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300