From DevOps to DevSecOps

Klara Eserstam and Pontus Broberg

April 2020

Abstract

This essay explains and compares DevOps and DevSecOps, suggests the cultural shift and extension of tools needed to go get started with DevSecOps, and presents DevSecOps tips from how Microsoft and PayPal. Overall, the conclusion is that DevSecOps is very similar to DevOps and the main difference is where the security testing is started. For a successful shift to DevSecOps, the developers need to be interested in security and given tools that can point out security mistakes while coding. In the future, more focus on automation and AI when reviewing security is likely.

Contents

1 Introduction

Two main goals of DevOps is to accelerate development and enable continuous delivery. In fear of slowing down business, parts that are less important for a successful delivery may be left out. On such part is security. Often security aspects are not regarded until the end of the development process, which makes vulnerabilities more likely to be either quick fixed or disregarded to promote fast delivery. DevOps professionals state that they want security tools that reduce the risk of vulnerabilities without slowing down work [5], and to achieve this security must begin earlier in the software development life cycle. This is known as shifting security "to the left", and it even has it's own term, DevSecOps. Another reason to take security measures earlier is due to the fact that nearly half of developers cannot code securely and malicious code can be injected in any stage of the development process [11].

This essay will explain how security is dealt with in traditional DevOps as well as DevSecOps, present some examples and approaches for shifting to DevSecOps, and run through how AI can be used in security. Based on this it will examine:

- Differences between DevOps and DevSecOps
- How to successfully implement security in DevOps
- The future of DevSecOps

2 Background

DevOps is the practice that looks to automate and integrate the different processes between Development and Operations, hence the word DevOps. The goal is to be able to more reliably test and release software while working in an agile structure and bridging the gap between software developers and IT operation teams.

In this section, we will explain traditional DevOps and DevSecOps to give a basic understanding of the two disciplines.

2.1 Traditional DevOps

According to Atlassian, a company that creates agile tools for software development management, the DevOps movement started around 2008 when software developers and IT operations communities started to point out some major flaws in the traditional software development model. [1]

In this old model, the two different teams worked separately with their own objectives and goals resulting in a lot of friction between the teams while new releases of software were slow and unreliable. Developers most often want to work as agile as possible, to release bug fixes and new features quickly and with low friction, while IT Operations are most often focused on stability and

uptime on the products that are released. These two different goals can seem contradictory and would with the old development method be the reason why neither the developers or operations teams happy. And so discussions began to emerge and soon the idea of traditional DevOps (just *DevOps* from now on) was born.

The DevOps model works to satisfy both the developers and operations by enabling the developers to make multiple releases while automating the processes of testing and building of the product so that operations can be sure that what is set to release won't break or is a bug-riddled mess. This DevOps model can be broken into different phases to create a pipeline, this to more easily be able to describe what kind of work is done in the different phases of a *software development life cycle* or SDLC. While the names and content of these phases may vary slightly from company to company, the basic ideas and structure remain the same when speaking about DevOps.

This pipeline, as described by Pennington, contains the eight phases plan, code, build, test, release, deploy, operate, and monitor, which together make up a sort of SDLC. He also notes that while breaking the DevOps pipeline into these phases make it easier to talk about and understand, it is important to know that this workflow is very much continuous and followed by the same team or a blend of teams, depending on the structure of the company and that there exist no hard barriers between the different phases. [14]

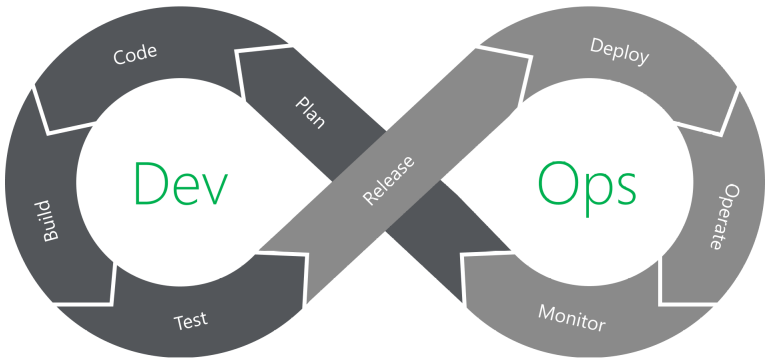

Figure 1: The eight phases of DevOps [14]

The planing and coding phases pretty much speak for themselves, this is where you make up some kind of project plan and then the developers get to actually coding the program according to specs. It is not actually until the build

phase where DevOps actually come in to play since this is where, after some light reviews of the code, automated processes start to build and the code and try to identify any fails or regressions in the building. When the build succeeds another automated process is started where more extensive tests are run. In this testing phase, manual tests are also done to see if an actual human user finds any issues or problems with the product. This is also the phase where we see the first sign of security come in to play as this is where penetration testing and other security tests are often done, but more on this later.

Going into the release phase is somewhat of a milestone since now all the tests are passed and the program should now be in a state where regression or failures and bugs are highly unlikely. In some organizations the builds that made it to this phase are automatically deployed with different kind of feature flags, meaning that the customer won't actually be able to use this new feature until it is manually activated, and on other organizations more controlled released are issued so this build is not deployed before manual approval is given. In the Deploy stage, the release is done and the product is released to the production stage. The new version is often deployed alongside the old version so that if any major flaws or bugs are found by a customer the company can easily rollback to the old and stable version without any real downtime. The operate phase is the IT operations home where they work hard to make sure that the product is running smoothly. This is also where feedback is gathered from the users which connect to the last phase, monitor, where this feedback is taken into consideration and discussed so that a new plan can be made for improvements or new features to the product, and with that planning the pipeline loop starts all over again.

2.2 DevSecOps

The idea behind DevSecOps s quite simple. In its core, it is not so different from DevOps and can be described with the same phases. The difference then becomes that instead of incorporating security tests in only one of the pipeline phases that was talked about in the section above. The difference being that security tests should be conducted throughout the entire process, this is sometimes called *Shifting left*. If we imagine the phases in a horizontal timeline, then saying that we shift something left means that it is done earlier in this timeline, the result being that any security flaws in the program are found earlier in the development process.

It is important to note that this should however not replace the traditional security testing that is done in regular DevOps or traditional software development, rather it should be a complement to them. The roles of these tests are somewhat changed with DevSecOps since instead of being all the testing that is done, it is now a final check, just like the other final tests that are done in that phase of the pipeline. [13]

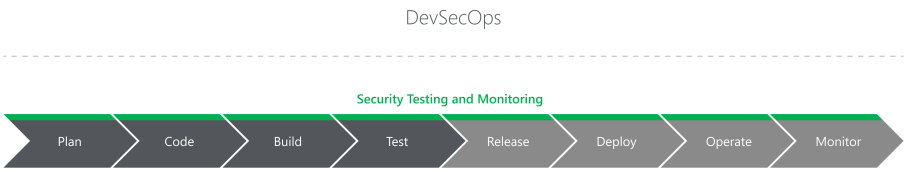

Figure 2: Security testing in relation to the phases of DevSecOps [13]

3 Making the shift

To make the shift from DevOps to DevSecOps, one needs to consider the existing security measures and how the can be improved, but most importantly how they can be shifted to the left. This section presents some of the steps to take.

3.1 Cultural shift

One of the steps towards DevSecOps in your organization is to make a cultural change. This change involves creating a mindset that everyone is responsible for security, and to create an environment of trust that everyone works together [6]. Trust is achieved through transparency, by sharing mistakes discovered as well as what works and what doesn't. It is also important with shared goals, with everyone striving towards improving DevSecOps measurements [3]. Things that can be measured are:

- Number of security issues discovered in production. The goal is to decrease these.
- Percentage of deployments not succeeding due to failed security tests, again the goal is to lower this percentage.
- The time it takes to fix security issues. Issues that are discovered early should be easier to fix, and less complex solutions required should be a reward from DevSecOps.

3.2 Tools and automation

As the CI/CD pipeline is an important part of DevOps, one of the first steps when shifting to DevSecOps should be to enhance this pipeline and automate security. There are many tools for static code analysis to manage vulnerabilities, which can be integrated early in the SDLC.

One such tool is SonarQube [16]. SonarQube is a tool for code quality and security, using Static Code Analysis rules to find bugs, detect security issues, and enforce clean code. It integrates well with existing tools such as Travis

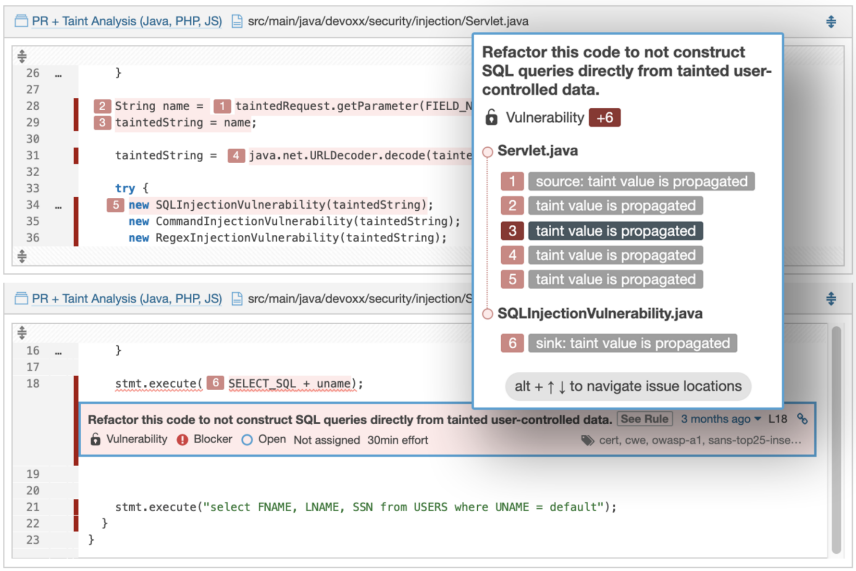

Figure 3: Example of the SonarQube UI [16]

CI, Jenkins, GitLab, and Azure DevOps to fit well into the DevOps pipeline. SonarQube can help developers tackle two types of security issues, Hotspots, and vulnerabilities. A Hotspot is a security-sensitive piece of code, that may not affect the application as a whole. It is then up to the developer to review the piece of code. Vulnerabilities impact the application's security and need to be fixed immediately. The UI for security issues is customized for developers, see figure 3, so they understand the behind the vulnerability and wherein the code the compromise occurs. SonarQube also provides security reports based on OWASP Top 10 [12], SANS Top 25 [15], and CWE [4]. For instance, SonarQube can detect SQL injection, Cross-site Scripting, and Code injection-based by analyzing the source code.

There are also tools to detect ongoing threats, where Logz.io [2] is one such tool. Their threat intelligence can detect threats and analyse them, by comparing data with records from public threat feeds such as AlienVault and blocklist.de. If something indicates a compromise, the threat information is recorded and displayed on logz.io, see figure 4.

3.3 Red vs Blue at Microsoft

At Microsoft, they do something called Red vs Blue team to get everyone engaged in security. [7] They do this in many of their teams but this specific example comes from the Visual Studio team services, presented by Buck Hodges. They had realized that the mindset that many in their team had was that they didn't think something was a threat or since they never had been breached they

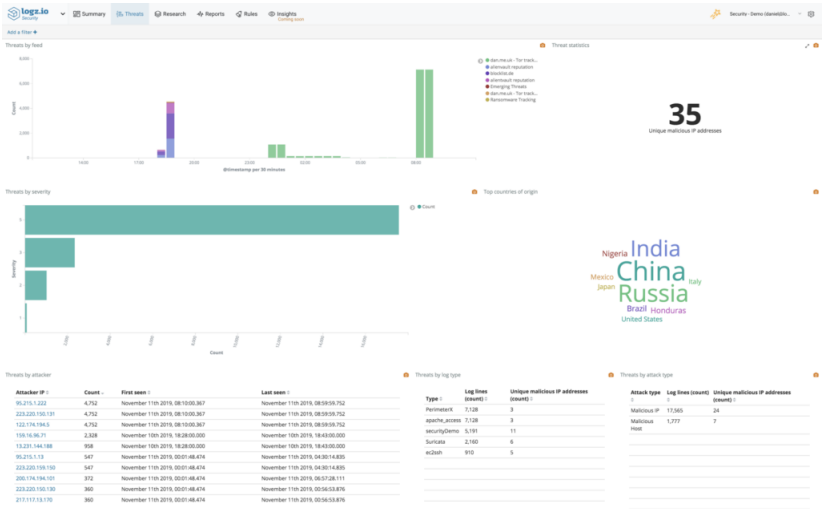

Figure 4: Example of threat recordings on logz.io [2]

never would be. The mindset shift that was needed was that everyone should assume that a breach can happen.

Red vs Blue is an event where a Red team attacks your service, while a Blue team tries to detect and respond to the attack. The Red team often consists of members both from within the company and external penetration testers, as the external tester know their way around scan tools and can find vulnerabilities that way, but the internal ones have a better understanding of the system and can combine multiple security issues. The Red vs Blue event usually lasts a couple of days to a week but could be longer. During this time, the Red team exploits vulnerabilities they find while the Blue team has to monitor to try to find out what is going on.

Apart from finding vulnerabilities through these events, Hodges mentions that the keys to these events are the reports that come from the Red team. With the report, they present to the development team what parts of their code they broke and how they did it. This creates a change of attitude among the team members, as they realize that breaches are possible and that they are responsible. This worked as motivation as no one wanted to be the reason behind a breach from the Red team.

3.4 DevSecOps at PayPal

Wes Hummel, VP of SRE at PayPal, explains that they treat security as a "strategic business priority and a fundamental part of how [they] develop, release, and maintain [their] product code" [9]. He says that it is integrated with every layer and regarded through from beginning to end of the SDLC. For him, DevSecOps means to provide the developers with great tools to help them create

secure quality software and also create a culture that build securely by default. They have worked to create a culture understand that successful products is dependant on both development, operations and security, and this makes it possible for them to trust their developers with freedom. To empower their developers, they give them the choice to use the recommended tools, including security penetration testing, security controls, threat modeling, automated scanning, and more. But they do not believe developers should be forced into a specific tool suite, so they also give the choice of picking their own stack.

4 AI in security

While Static Code Analysis is already used in tools such as SonarQube to detect vulnerabilities in the code, it can not detect or halt ongoing attacks. To do this, one can take advantage of AI and Machine Learning to identify harmful actions based on historical data, as opposed to based on recognition of threat signatures. The difference between these two security systems is also that the former one is managed by humans, while the latter is automated.

One way that AI can help predict attacks from users, is with User Behavior Analysis. Based on previous data, the machine learning model can analyze user behavior to predict an attack in real-time.

AI systems can also be used to identify malware, using heuristic algorithms. The system identifies harmful code and isolates it from the system before it can enter [8].

Apart from identifying ongoing threats, AI can also be used to perform penetration testing [10]. One existing, open-source tool that does this is Deep Exploit. It works by scanning for ports and determining their status and then runs machine learning-based exploits. If it successfully enters it continues to exploit internal servers.

Machine Learning and Artificial Intelligence reduce the time security experts need to spend identifying threats [17]. The number of security alerts that appear can also be very overwhelming for a security system based on people, where an autonomous system can filter these alerts based on data.

5 Discussion

This sections presents the three main points from the essay and discuss them more in depth.

5.1 Similarities and differences

Overall the differences between DevOps and DevSecOps are not that many. The difference mostly comes down to when the testing is done, and in the case of DevOps, this is only done in only one of the phases described in section 2.1, or at most in two phases with the second one being in the build phase where only minor security checks would be conducted. Extensive security testing is done at

the end of development either way and if any kind of security flaws are found at this stage, that would mean that patches would need to be added to the code or in the worst case, fundamental rewrites and refactoring.

Since shifting left and going with a DevSecOps approach means that testing is done through all of the development phases, this comes with the immediate advantage that security issues are found earlier in the process and can be addressed as soon as possible. If these problems are found early in the development pipeline, then these issues are probably easier to solve and not as costly in both time and resources as they would have been if they were discovered at the end of the development.

DevSecOps also remains very similar to DevOps since the idea is not the change the already existing security testing that DevOps already has incorporated, but instead adding more security checks before and after. In traditional security testing, you simply rely on, that a single agent, be it a single person or a third party company, will find all security flaws and notify you about them. Since the DevSecOps approach does not skip this phase, chances are that when you eventually get to this stage of development and let a third-party testing company check for vulnerabilities, none will be found.

5.2 Successful DevSecOps

One important aspect of successfully shifting to DevSecOps seems to be to get your developers interested in security and motivate them to write secure code. Initiatives like Red vs Blue at Microsoft can bring out the competitive side in people, as well as making people realize that their code might not be as secure as they think.

To keep the speed of DevOps, it's also important to relieve the developers of as much security work as possible, while still expecting secure code. Here, tools like SonarQube can find the most common vulnerabilities and educate the developer about the issue in the process. For this, the developers need to feel comfortable with the tools used, which PayPal solves by both having a well-selected, recommended tool suite, while still giving the developers the possibility to use their stack of choice.

5.3 Future trends

A future trend for DevSecOps seem to be further automation. While tools for static code analysis helps the developers bring security to their code-writing in the beginning of the SDLC, there are still things to be done in the end of it. Detecting and quickly dealing with breaches is an area where AI can be of great help. Tools like logz.io may become more advanced to not only detect threats based on known breaches but also make threat predictions based on breach data. We might also see events such as Red vs Blue or just penetration testing in general with AI tools similar to Deep Exploit. This would enable more frequent penetration testing, and give it a natural role in the SDLC.

6 Conclusion

Writing secure code is something that always has been important and it is something that most likely will remain very important and considering this, an approach like DevSecOps is probably here to stay. With tools like SonarQube and logz.io being explored and developed every day, developing secure code and features to new and existing programs will hopefully only get easier in the future as well. The great thing about DevOps is also that we already have these amazing tools for automating processes, which means that we probably only need to alter these tools a tiny bit to help us in this quest to always write secure code.

Whether or not your company is working according to the DevOps model; shifting left and focusing more on security, like the DevSecOps model suggests, in every step of your SDLC is something that everyone should focus on adopting more. If this would be the case then not only would their customer's everyday life in applications and programs be more secure, but these companies would also save a lot of time and money on not having to delay and rewrite a bunch of code, just because someone made a mistake or overlooked something early in the development process.

References

[1] Atlassian. *What is Devops?* URL: https://www.atlassian.com/devops.

[2] Daniel Berman. *Speeding Up Security Investigation with Logz.io Threat Intelligence.* URL: https://logz.io/blog/speeding-security-investigation-with-logz-io-cloud-siem-threat-intelligence/.

[3] Kevin Casey. *How to build a strong DevSecOps culture: 5 tips.* URL: https://enterprisersproject.com/article/2018/6/how-build-strong-devsecops-culture-5-tips?page=1.

[4] CWE. *Common Weakness Enumeration - A Community-Developed List of Software Hardware Weakness Types.* URL: https://cwe.mitre.org/.

[5] Fortinet. *2019 State of DevOps Security Report.* URL: https://www.fortinet.com/content/dam/maindam/PUBLIC/02_MARKETING/08_Report/report-devops.pdf.

[6] Sylvia Fronczak. *How to transition into DevSecOps.* URL: https://blog.sqreen.com/how-to-transition-into-devsecops/.

[7] Buck Hodges. *MindSet Shift to a DevSecOps culture.* URL: https://www.youtube.com/watch?time_continue=660&v=7fW4wZbJtg0&feature=emb_logo.

[8] Oleksii Kharkovyna. *CyberSecurity + AI: defined, explained and explored.* URL: https://towardsdatascience.com/cyber-security-ai-defined-explained-and-explored-79fd25c10bfa.

[9] Vikram Kunchala et al. *DevSecOps and the cyber imperative - Elevating, embedding, and evolving your risk response*. URL: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/technology/deloitte-uk-tech-trends-2019-chapter7-devsecops.pdf.

[10] Brandon Lammey. *Cyber Defense and AI: Automating Penetration Testing*. URL: https://medium.com/brandon-lammey-intro-to-ai/cyber-defense-and-ai-automating-penetration-testing-91ccb56dd93a.

[11] Daniel Newman. *5 Reasons DevOps And Security Need To Work Together*. URL: https://www.forbes.com/sites/danielnewman/2018/09/30/5-reasons-devops-and-security-need-to-work-together/#2541d603714a.

[12] OWASP. *OWASP Top Ten*. URL: https://owasp.org/www-project-top-ten/.

[13] Jacob Pennington. *Shifting Left: DevSecOps as an Approach to Building Secure Applications*. URL: https://medium.com/taptuit/shifting-left-devsecops-as-an-approach-to-building-secure-products-3a418fbbafbe.

[14] Jacob Pennington. *The Eight Phases of a DevOps Pipeline*. URL: https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba.

[15] SANS. *CWE/SANS TOP 25 Most Dangerous Software Errors*. URL: https://www.sans.org/top25-software-errors.

[16] Sonarqube. *Code Security for Everyone - Detect security issues in code review with Static Application Security Testing (SAST)*. URL: https://www.sonarqube.org/features/security/.

[17] Emmanuel Urias. *Applying Machine Learning and AI to Improve Cyber Security*. URL: https://invidgroup.com/applying-machine-learning-and-ai-to-improve-cyber-security/.