How to Build a DevSecOps Pipeline

GUARDRAILS

www.guardrails.io

Contents

Introduction

In IBM's 2022 report, the average cost of a data breach was $9.44 million (USD)[1]. The same report showed the average time to identify, react, and contain a breach was 277 days. However, as Figure 1 below shows, breach discovery time is of the essence. Not least because the average cost difference between breaches whose life cycle was greater than 200 days was $1.12 million more than for those whose breach lifecycle was less than 200 days.

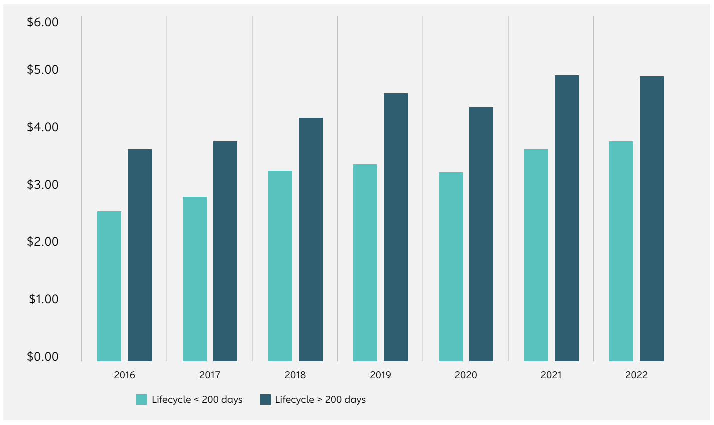

Figure 1 – Average cost of a Data Breach based on data breach lifecycle [1]

A significant number on its own, but also a sizable and tangible sum that would add significant weight to any organization looking to reason an increase in, or add security to, DevOps and their Software Development Lifecycle (SDLC). Secure DevOps—DevSecOps—is increasingly becoming the only viable solution to the many problems organizations face. However, for executives, cost is but one factor because there is far more at stake than just dollars.

As security professionals, you already know that a data breach has far wider repercussions including loss of reputation, customer trust, and market share.

Though some larger organizations might recover from each, it is never to the same state or position as before. With attacks on the rise, and hackers picking and choosing targets at will, organizations know they need to get better at security–and fast. DevSecOps helps to do just that.

DevOps itself has made great leaps in progress over the last decade—except with security. Today, still, DevOps teams lack the security knowledge and know-how to implement DevSecOps and their insecure processes, pipelines, and entire SDLC remain exposed and at risk. This paper will look at how to change that, how to build a DevSecOps pipeline, as well as the benefits for doing so. However, to get the full picture, it also pays to understand and dispel misconceptions around building a DevSecOps pipeline, as well as being aware of past problems that contributed to implementation failure. First, and to get a better understanding of the way forward, a quick refresher on how we have found ourselves where we are today helps provide context around the overall benefits of building a DevSecOps pipeline.

Shift-left was hailed as the answer

First announced in 2001, though Shift-left was initially related to software quality testing, it later expanded to moving Security (Sec) closer to Developers (Dev) and Operations (Ops) in the SDLC. Though the idea was, and still is, sound, many organizations lacked the flexibility, know-how, and capability to accomplish it. In the intervening years, and driven by the need to both accelerate product development and to bolster competitive advantages, Dev and Ops merged to form DevOps. Unfortunately, in time-honored fashion, Sec remained where it was, ostracized, unpopular, and something to be suffered at the end of a long journey.

Imagine being on a long-haul flight with multiple delays, changes, and hold ups. Exhausted, you finally land, and can't wait to get home. With your bags, you stroll, head-high into the Nothing to Declare customs channel, hoping to get through without so much as a pause. Unfortunately, you see the beckoning finger. You hate it, you know it's a bind, and you know it's going to delay you no end. But you also know there's nothing you can do, ***"Please drop your trousers and underwear, sir. Apologies, but my fingers are a little cold this morning..."*** That's DevOps and Security.

From security's perspective, they remained exasperated that developers still couldn't fix their own mistakes. Had they done so, had fewer bugs and vulnerabilities reached security, then not only would the time, cost, and complexity of fixing them be minimal, but so would the security bottleneck.

Regardless, and despite the rewards on offer, for many organizations Shift-left wasn't practical. The result being that, on the one hand, for those that did accomplish Shift-left, we see examples like Amazon releasing software over 3 times per minute; yet, on the other, for those that didn't, and despite the phenomenal acceleration of DevOps since Dev and Ops merged over a decade ago, security remains an afterthought:

"Developers have done phenomenal work in the DevOps space, whereas Security has not kept up with them. It's not feasible for DevOps to slow down, so security needs to step up. This is easier said than done since it requires a cultural shift in the way the software delivery teams, and security teams work together." [2, p 8]

It must be said that the cultural shift mentioned **is** the critical element of DevSecOps. Everything hinges on organizational culture. However, when you get it right, not only will security be an integral component of your entire SDLC, but it will also transform the way in which your organization thinks about, and implements, security. Organizations that want to remain competitive cannot keep doing the same thing. They must change the way they think about and treat security and, by making security a part of everything they do, they will experience True Left rather than Shift-left. (True Left embeds end-to-end security throughout your entire SDLC and is the key to transforming DevOps pipelines into DevSecOps pipelines.)

What is a DevSecOps Pipeline?

It is important to understand what we mean when we talk about building a DevSecOps pipeline. First, what exactly do we mean by DevOps? Definitions abound, but though tools and tech are essential components of DevOps, DevOps is about cultural philosophies and approaches that educate and improve cooperation, coordination, and collaboration between teams. Automation is at the core of DevOps, and tools and tech help deliver that capability by assisting in dismantling the barriers between teams and tech. Eroding these barriers offers significant benefits, not least with better relationships and faster delivery of more efficient and reliable business software. That's what DevOps does. However, DevOps lacks security.

"Security is not the product of DevOps practices..." [2, p 47]

In its simplest form, DevSecOps is the process of securing DevOps.

In the same manner that DevOps isn't about tools and tech, neither is DevSecOps. It stands to reason that when neither tools nor tech are prime focus, there's little to gain in wrapping either with other security-focused tools and tech. Rather, DevSecOps refers to the adoption of a security culture within the organization—a culture where security becomes everyone's responsibility—following which security testing then becomes integral to your DevOps pipelines. As we will come onto shortly, adopting this approach delivers several benefits, including an overall improvement in speed, agility, and security across your entire SDLC (including all application development pipelines). DevSecOps also delivers the potential for rapid changes to existing processes, faster and more secure software development, and an increased speed to market. Additional benefits follow.

What are the benefits of a DevSecOps pipeline?

Establishing a DevSecOps culture, that is a culture where end-to-end security is a first class citizen within your organization, can offer significant benefits including:

- **Breaking down inter-organizational silos and removing friction**

  54% of security professionals said they didn't think their devs understood modern security threats[3], and working together is key to DevSecOps success.

  DevSecOps helps teams that are traditionally reluctant to change work closer together. This not only helps remove friction and dismantles interdepartmental silos, but also fosters stronger and better working relationships.

- **Shorter development cycles and faster development**

  One of the key tenets of DevOps is automating mundane/boring manual tasks that have traditionally been performed by humans. DevSecOps seeks to do the same with security. In turn, this frees up teams to focus on harder and more complex problems. Moreover, minimal disruptions means that development completes faster, with greater accuracy, and enhanced overall efficiency.

> **"DevSecOps teams simply don't exist."** [2, p 45]

- **Improved flexibility and scalability**

  Security automation allows organizations with limited resources and bandwidth to scale. Automation incorporates repeatable and adaptable security processes across all environments, and this flexibility provides both a rapid response capability and the means to adapt to change much easier.

- **Visibility, traceability, and accountability**

  In addition to improved visibility, automated processes afford full traceability and accountability across environments and pipelines. Being able to see who made what changes and when is excellent for bug tracking, awareness and accountability, and Just-in-time (JIT) learning. (Did you know that 98% of bugs are introduced in development?) In conjunction with your organization's strict 'No Blame' culture, this is not only an excellent method for educating your team, but also ensures that all team members are accountable for their work. This ownership is great for team morale and in fostering improvements.

- **Better application security**

  DevSecOps lays down layers of embedded security that strengthens security within your DevOps pipelines. This layered security delivers greater visibility for security threats and provides you with an early bug detection system. Being able to address and alleviate errors at source, as they arise, enables you to build better software, release it faster, and with fewer bugs and vulnerabilities.

- **Reduced costs**

  Fixing bugs earlier saves time and money. Given that repairing defects at the release stage can cost up to **640 times**[4] more than developers fixing them at source (and before they're committed), it's critical that organizations enforce scanning and remediation throughout. This not only results in greater cost savings but also minimizes overall disruption, improves workflow, helps build better software, and improves overall security.

- **Reduced time to market**

  Creating software with fewer bugs and vulnerabilities—and detecting and eliminating them at source—involves fewer people, processes, and purse-strings. Automated processes can offset the slow (costly), error prone (bugs and vulnerabilities), and hard to scale (inflexible) manual processes.

There are significant benefits to building a DevSecOps pipeline, but you must get there first. If you're new to this, there are several misconceptions and problems to consider.

Common misconceptions and problems with building a DevSecOps pipeline

Mindset is everything and if the prevailing attitude, below, exists, then your initiative most certainly will fail. However, it most certainly won't when you understand the main misconceptions and problems that organizations face with DevSecOps pipelines.

"Your DevSecOps initiative will fail." [5]

Misconceptions with building a DevSecOps pipeline

Common misconceptions include:

- **Security is a by-product of DevOps**

  It's not. However, the days where security was both a separate entity and something to be suffered prior to software release need to end. Security must be the responsibility of everyone in your organization, and they must switch to a security mindset, applying relevant security principles on a constant basis.

- **We tried it before, it didn't work, and history always repeats itself**

  A common misconception, preventing this from reoccurring is the purpose of this paper. Today, less than **50%** of companies scan code, so there is a way to go. By learning from the past, we can prepare better for what comes next. Besides, though you may have struggled before, there are important lessons to learn and in this you most definitely are not alone.

- **Too expensive in general**

  The actual reality is opposite: an effective DevSecOps pipeline reduces costs. As we covered in the introduction, with that $1.12 million 200 days threshold difference, a combined integral security culture and automated DevSecOps pipeline will mean better prepared and security-focused team members, earlier bug detection and remedy, and maybe even significant cost and other savings. For organizations that deem DevSecOps too expensive, consider the price of loss of trust, reputation, litigation damages, etc. For example, GDPR penalties are between 2%-4% of global revenue, so hardly an insignificant sum.

- **DevSecOps is an impossible task**

  The accelerated speed of DevOps over the past decade means that developers are now writing code faster than ever before. Unfortunately, given developers create 98% of vulnerabilities, speed is clearly at odds with security goals. Devs code, they need to produce the goods, but (in an ideal world) they would also need to own their mistakes.

  (Furthermore, asking developers to concentrate on the security aspect doesn't work. They either don't have the skills, the time, or the inclination for this. Consequently, an alternative approach is needed. An approach that takes the matter out of their hands, guides and educates them, and slides them down the correct path: think of barriers–or guard-rails–either side of the road to keep them safe and moving in the right direction. This approach not only works, but it also solves many long-standing security and culture problems.)

- **Shift-left has failed**

  Though a common misconception, it's understandable because its application was flawed: the tools shifted left, but the people, processes, and culture remained where they were. When you also combine this with organizational issues, ineffective strategies, pipeline problems, a lack of understanding, etc., failure was hardly a surprise.

"Cybersecurity is everyone's responsibility, not just that of a central security function." [2, p 102]

Problems with building a DevSecOps pipeline

Misconceptions are easy to dispel. However, underlying problems that organizations have experienced while building a DevSecOps pipeline are both a different story and in a different category. These are tangible, key underlying problems that all organizations who have failed DevSecOps share.

- **What does DevSecOps look like in your organization?**

  This is the very first question your organization must answer. DevSecOps is not a mystery tour, and to measure and determine success, you must know your destination.

- **Top Management Backing**

  Without which failure is almost guaranteed. People are resistant to change, and Top Management must visibly and actively drive change through. Without such backing, removing friction, breaking down silos, establishing effective collaboration, etc., will either not materialize or will only do so at snails-pace and with the utmost reluctance.

- **Adopting a Security (and learning) Culture**

  Again, DevSecOps is not about adding tools and tech, it's about people, processes, and culture. To make security an integral component will encompass every aspect of people, processes, governance, and the way you perform business itself. Fortunately, an effective security culture fosters a common goal and permeates confidence throughout. A confidence that brings out the best in your people and helps deliver exceptional results.

"You can't just buy DevSecOps."[6]

- **Misalignment and lack of communication between leaders and teams**

  A study by Immersive/Osterman showed that

  though

  80%

  of senior managers believed security was everyone's responsibility

  only

  27%

  of their front-line development staff saw security as a component of their job responsibilities

  This lack of congruency in approach, communication, and voice is frowned upon— messaging must be consistent.

"58% of cybersecurity practitioners report that cybersecurity leaders don't communicate the company vision to the rest of the cybersecurity team.[7]

- **Security Education**

  Is a critical component of organization culture and change and needs to underpin your entire DevSecOps transformation. Education is both integral and on-going and it is vital that your workforce knows about and understands security, how it relates to them, how it affects them, their job/business function, and the company. Security can no longer be just within the remit of those with that label in their job title.

“44% of all breaches were due to stolen user credentials.” [8]

- **Focused on technology**

  Automation relies on tools, but though technology is one key to your effective DevSecOps solution, technology alone will neither deliver your solution nor guarantee effective results. Results hinge on right people, processes, culture, focus, etc., all working together and towards a clear, achievable, and common goal.

- **Security is drowning in noise**

  Security teams typically have a multitude of security scanning tools in operation. All of which churn out extensive reports containing overwhelming volumes of information. Unfortunately, the vast majority is noise or false positives and, with so much to attend to, engineers find it physically impossible to investigate, let alone fix, even a fraction of those findings[9]. This practice has to change so that only useful reports and alerts become visible.

- **Ineffective DevSecOps Strategy**

  An ineffective or untried strategy will lead to a whole host of problems and be a likely guarantor of DevSecOps failure. Such problems include divergent views, sole reliance on tech, no connection with the business outcome, restricted scope for collaboration, a lack of flexibility, technical issues, being overly complicated, delivering a poor User Experience (UX).

- **Insufficient/Incorrect planning and resources**

  Important is creating a technology roadmap to understand and map out any skill or experience gaps within the organization. Designating *Security Champions* (see below) is fundamental, as well as defining clear project requirements, including understanding what pipelines you have, how to protect them, incorporating real-time feedback on only those pipelines that need it, providing adequate resources, etc.

- **Ineffective Defects Management**

  98% of vulnerabilities are introduced by developers but developers are neither security experts nor want to be. In one poll, **73%** of those surveyed did not even consider security to be within their remit. Consequently, how organizations manage defects management, and address the fundamental manner in which developers view security in their own role, must be encompassed within any cultural realignment.

- **Poor coding practices**

  Hiring fast, but diligence-lacking coders on a freelance, temporary basis can cause more hassle than it is worth. Typically, such coders will write their code as fast as they can, get paid for their work, and then move onto the next job. Far too often, this practice results in later problems where their code lacks the necessary controls and is sub-standard. Unfortunately, the freelancer is engaged elsewhere, and the organization is left to fix the problem code and mitigate any resultant technical debt.

- **Costly defect remediation**

  Early remediation is key because the greater the length of time between the defect being introduced and its detection the more expensive the fix. As with poor-coding practices, above, once a build is complete, your own developers will be engaged elsewhere and will have diverted their entire focus to that new project. Following the discovery of defects weeks or even months later and then trying to get the developer off their existing project to remediate recent bugs is usually impractical. However, should you be fortunate enough to be allocated this resource, they must now pick up where they left off, including deciphering additional code on top that was written by someone else. Too often, this leads to a *whack-a-mole* type exercise where finding and fixing one bug causes another to pop-up elsewhere. Early remediation is always the best answer.

- **Security defects lower confidence and elevate risk**

  When a customer discovers a defect, then it will sway confidence in the product and may even impact the relationship. As bad as that may be, it's far less than an adversary discovering the defect and then using it to compromise your software and the people using it. (This is a long-standing and widespread problem with security being where they are, so far along in the SDLC process: by the time the defects get to security and are discovered, it's often too costly and timely to get them fixed. It then comes down to the business to determine the next step: remediate and postpone the release, or just fix the critical bugs and then release with the remainder in its current condition.)

- **Devs and security mix like oil and water**

  According to the 2020 FOSS Contributor Survey[10], one developer described security as being "soul withering". Another said, "I find security an insufferably boring procedural hindrance." Worryingly, only **2.3%** of developers improved their code. Unsurprisingly, even fewer expressed any tangible desires to soul-wither any further, and it seemingly will require a seismic shift to alter attitudes.

  (Though it is now over 20 years since Shift-left, and over 13-years since DevOps began, one argument is this mindset and approach is indicative of everything that's wrong with software development in the modern era. However, this is only one side of the story as, during this same period, and as we shall cover in Security Champions, below, nothing was preventing security teams from stepping up into more of a cheerleading/mentorship/coaching type security role which would alter their perceptions around security. Both ends of the development pipeline were, and still are, misaligned.)

- **Governance issues**

  Key to continual monitoring and improving is ensuring that you can measure all key business metrics. To accomplish this, it's vital to ensure that any chosen tools support and allow you to measure what is important to you. This not only means correct tool selection, but also correct integration, configuration, and management. Having the ability to both measure and display results is key to continual improvement and in overcoming the natural resistance between people, processes, tech, and tools.

Being aware of and addressing the above ensures that you are best placed to maximize your chances of DevSecOps success.

How to build a DevSecOps pipeline

To reiterate,

“What exactly does DevSecOps look like in your organization?”

It’s vital that you both understand and know this. Unfortunately, there is no one-size fits all model, but it’s key that you have a clear vision on what success looks like for you. One common model for constructing a DevSecOps pipeline is:

1. **Planning your DevSecOps pipeline**
2. **Establishing and Measuring Governance**
3. **Automating Security**
4. **Educating your Developers**
5. **Monitoring and Improving**

As mentioned earlier, fundamental to planning are:

- **Top Management backing**
- **Adopting a security culture and mindset within your organization**
- **Taking a "security-first" approach.**

Naturally, the former will be easier and quicker to obtain and implement than the latter, so the sooner the wheels are in motion, the better.

Once in motion, the focus of planning your pipeline will be people, processes, and product/tech. Existing barriers, or silos, between your teams need to be dismantled and, to eliminate resistance and performance conflicts, you must get everyone on the same page and work towards your collective common goals and to overcome any conflicts.

For example, one common conflict may be that a primary DevOps goal is to get the product to market as soon as possible. However, this will directly conflict with the security team's primary goal of minimizing the number of bugs and vulnerabilities the product ships with. This conflict causes friction whereby the former consider security to be a bottleneck in the release process and a key hindrance to DevOps missing release deadlines; whereas the latter can't believe that despite 20-years of adding bugs and vulnerabilities, DevOps still don't do their own due diligence, don't own their mistakes, and don't remediate these bugs at source. If they did, then there wouldn't be any delays preventing release.

It's this type of friction and conflict that DevSecOps removes. When everyone and everything is working together–people, processes, and product–the path to operational success is much smoother.

- **People**

  Education is key to overcoming change resistance. The field of security is ever evolving and, to be effective, continuous learning and education must keep abreast. This naturally strengthens the organization, but it also serves to focus and unify your team towards that common goal, and assists with team building, breaking down silos, and improving morale. Moreover, instilling a "security first" mindset helps orientate your team to understand and be extra diligent towards both external and internal attacks. e.g., attackers obtaining restricted access; brute-force attacks; data loss; Dancing Pigs[1] and compromised user credentials, etc.

"Hackers don't break-in, they log in."[11]

[1] "Given a choice between dancing pigs and security, users will pick dancing pigs every time." (Edward Felton)

- **Processes**

  Analyzing and understanding all your current and future processes, pipelines, and environments is crucial to planning and implementing effective protection and security. This may involve establishing regular security assessments, security policies that ensure only reviewed and accepted code can be committed, limiting access to production environments, establishing an immutable infrastructure, fixing vulnerabilities on a priority basis, etc. Analyzing your processes will help determine gaps between what you have, what you need, what needs protecting (and what doesn't), and how you will achieve it.

- **Product**

  This determines the security automation tools that will protect your SDLC and pipelines. Tools such as Software Composition Analysis (SCA), Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Vulnerability Assessments (VA), and Secrets Management. As part of the initial planning phase, you may also need to determine whether your processes need to change to integrate with the tools, or whether you will need to select tools that deliver functionality that supports your existing processes (much of this depends on your current process maturity level). Whatever your choice, they will need to be developer friendly, fast and reliable, provide a low rate of false positives, and also feedback into your workflow in a smooth, workable and effective manner. (We will look more at this in the section on automating security, below.)

It's also vital that your chosen tools afford the opportunity to establish and measure governance metrics.

Establishing and Measuring Governance

"Platform governance consists of the processes around and advertisement of changes to the platform, inclusive of managing the security and availability of the platform." [12]

Processes define the way you do business. Not only that but, from a governance perspective, when the right people, processes, and products are implemented and configured correctly, the resulting data and reports are of greater value to the business. Subsequently, managing them becomes easier too. One method of grading how mature your organization's current governance is, is to use the Platform Governance Maturity Model:

- **Level 1**

  System changes are ad hoc, uncontrolled, lack transparency, and are unadvertised to users of the platform. Akin to making changes on the fly, this level is considered non-viable for a DevSecOps platform.

- **Level 2**

  All platform changes are conducted through defined, mostly manual processes. Here, criteria for change depends more on consensus than on objective criteria.

- **Level 3**

  All platform changes are conducted through strictly defined processes with clearly defined criteria that allows for rapid change. Where possible, this level uses automation to minimize the number of application developers impacted through that change.

(DevSecOps Guide: Standard DevSecOps Platform Framework)[12]

Level 3 with security automation is the goal for many successful DevSecOps implementations. At this level, organizations can track and measure their key metrics.

Key Governance Metrics

Though many more metrics can be tracked, 4 key metrics adapted from the lean manufacturing principles are[13]:

- **Lead Time**

  This is the average amount of time from checking the code into the Version Control System (VCS) to when it is deployed to production.

- **Mean Time to Restore**

  The time it takes to resolve or rollback an error discovered in production.

- **Deployment Frequency**

  The number of software releases to production during a specified period (our earlier Amazon example was 3 per minute).

- **Change Fail Percentage**

  The actual percentage of software releases and configuration changes to production that fail.

For example, if an organization has a low lead time, a Deployment Frequency and an MTTR in seconds, and a low Change Fail Percentage, then this likely indicates an agile, mature, flexible, and well supported software development team that can remediate issues fast, minimize downtime, and maximize customer value.

Note: it's important that metrics are not used in isolation, i.e, an organization that has a low Deployment Frequency for a long-standing, mature, and well-supported product, is in stark contrast to one with the same Deployment Frequency but whose product is new, relatively unstable, and has a high failure rate. Other metrics are also required to determine the current effectiveness of your environments.

Automating Security

The following 5 security phases[14] are one means of segmenting your DevSecOps pipeline and developer workflow into logical security stages:

1. Threat modeling

Using Bruce Schnier's *Sensible Security Model*, we have five simple steps: identification of organizational assets, discover risks or vulnerabilities to those assets, work out countermeasures to the vulnerabilities, analyze possible risks that the countermeasures may introduce, and then the evaluation of costs and trade-offs. Threat modeling underlies all of the following security phases.

2. Security Testing

Involves code-based SCA, SAST, as well as Mobile Applications Security Testing (MAST), and Interactive Application Security Testing (IAST). Experience shows that IAST is where most security testing falls and, together with DAST, all the above testing methods are often used for both penetration testing and testing that is performed by developers. The emphasis here is to minimize manual work and automate where possible. Within this Security Testing phase[15], and assuming the usual quality types of testing are already established (unit, integration, acceptance, UI, and manual testing), security conscious development teams typically perform the following 4 essential automation checks:

2.1 Pre-Commit Checks

Identify security issues before changes are committed to source code repositories and are often triggered by a Git (or other Distributed Version Control System (DVCS)) hook. At this pre-commit stage, an effective DevSecOps pipeline may involve secret scanning and light-weight code scanning, among others. By minimizing manual intervention, these early automated checks help identify bugs and code defects and aid developer productivity. Because they are in close proximity to development and are fast to run, they should be run often (*fail fast* being the key). This immediacy enables the developer that is about to push code to the remote, to fix bugs before doing so. Tests at this stage include Static Application Security Testing (SAST).

2.2 On Pull/Merge Request Checks

Automated application security tests run when local feature branches have been pushed to a remote source code repository and a Pull or Merge Request created. On Pull/Merge Request Checks focus on high security and critical code issues and include activities such as scanning for secrets, scanning for out of date dependencies (SCA), scanning for insecure code in the Pull/Merge Request (usually performed via SAST which is fast, whereas DAST, due to its longer running build, is slower.)

The Pull/Merge Request Check's goal is to break the build, i.e., if the code doesn't pass certain tests, then simply flag less severe concerns, and create additional Issues and/or Pull/Merge Requests, as necessary. Pair programmers–two developers working in tandem–are usually responsible for reviewing and approving the merge from the feature or bug fix branch to the development branch.

[2]https://www.synopsys.com/blogs/software-security/devsecops-pipeline-checklist/

2.3 Staging Checks

Occur in a staging or testing environment and are the final tests before a product is released into production. It may be necessary to copy and anonymize databases from production to staging prior to running checks. Staging checks can include activities such as IAST testing, if you have a series of manual tests to run; Fuzz Testing to see if you can trigger an error state or crash the application; DAST testing for common, critical and high severity issues; and even running 1 or more web application scanners in both automated and manual mode, as required. All staging checks must be completed prior to deployment.[2]

2.4 Production Deployment Checks

Once the application has successfully completed all stages and checks, it is ready for a production deployment check. This check consists of pre-deployment, deployment, and post-deployment checks. Pre-deployment checks include, where necessary, making environment backups, announcing the impending release and confirming that your development branch is ready to be merged into your main or production branch, configuration management, automated provisioning of the runtime environment (CD - AKA Continuous Delivery), etc. Post-deployment involves smoke testing of new features and/or bug fixes, etc.

3. Analysis and prioritization

Either your security and development teams or, if established, your Security Champions will analyze, review, and prioritize all potential threats and vulnerabilities that were discovered in the security testing phase, above. Ideally, the analysis and prioritization is performed by your Security Champions who guide their respective dev teams without involving your security teams. Countermeasure Product Backlog items are created and inserted into the team's product backlog.[16] If multiple pipelines or environments are involved, then one common practice is for several Security Champions to function under the guidance of a Security Consultant.

4. Remediation

Is where, in order of priority, all threats and vulnerabilities that will lower the asset's risk rating are fixed. Security engineers will be on standby to advise and mentor Security Champions, as required (more below).

5. Monitoring

Involves the development teams performing continual assessment and tracking of all DevSecOps production pipelines to determine the overall security posture. The monitoring stage is essential for discovering new bugs, vulnerabilities, and spotting overlooked weaknesses or misconfigurations that either slipped through or have materialized since. During monitoring, security assessments and pentests should be run to keep abreast of real-world developing threats and, as we shall look at briefly below, organizations should also consider implementing Bug Bounty Programs to reward 3rd-parties (both internal and external) that detect weaknesses, vulnerabilities, etc.

For DevSecOps to work, developers must change the way they both think and act about security.

"You code it, you own it."[16]

Though this long-standing mentality and practice must change, it must be done within an already established, proven, and effective *No-blame culture* (this point cannot be overstated). Finger-pointing only delays change and will be detrimental to your DevSecOps initiative. Few will argue that developers must become more security aware and must transform this awareness into their coding and practices, but it's highly unlikely you can undo 2 decades overnight. Though it will take time, one proven method of accelerating transformation is through Security Champions.

Appointing Security Champions

Security Champions are developers who already have expressed an interest in security, and it is they who help bridge the gap between DevOps and your application security specialists. Some organizations might have a ratio of 80:1 or 100:1 with developers/security, but developers appointing Security Champions from within their own ranks has proven to help overcome the longstanding rifts between teams and to break down barriers.

If your organization has a security team in place, then your security engineers can act as mentors and train Security Champions. "Security Champions are to security what Scrum Masters are to Scrum."[3] These Champions empower their own development teams to embed the required security themselves. Security professionals working with a select group of Security Champions—one per development team is ideal—who act as conduits in filtering information down, and educating those they work with, is far more scalable and effective than blocking bug-ridden and vulnerable work.

Dev teams with a Security Champion that they know, work with, and inherently trust, helps build a cohesive and tight development team. In turn, this leads to more frequent, higher quality, and better communication, as well as improved performance in areas such as reducing the quantity of defects and the cost of their remediation, discovering security defects earlier, and in creating secure products.

Reducing the Quantity of Defects and Cost of Remediation

As specified earlier, a vulnerability discovered in a production environment can cost 640 times what it does to fix it in the design phase. Naturally, it is in the organization's interest to ensure this does not happen. Not only because of the direct cost savings, but also because while developers are working on a project, as we mentioned previously, they are focused on the project at hand, have everything to mind, and can react quickly to remediate any faults that they or others spot–they are deemed 'code fresh'. However, once the current project is complete and their talents and focus allocated and diverted elsewhere, they will quickly forget the details of the old project.

Consequently, when a bug is pushed to production (or even if a bug is discovered post-release because an ever-changing security landscape can uncover weaknesses that never existed previously), then it could be weeks or even months before the bug is discovered. Meanwhile, that developer is busy coding elsewhere. As such, discovering bugs further on in the life cycle may cause major disruptions and need extended time to fix. Disruptions that, had the processes and practices discussed above been integrated earlier, would either have been avoided or remediated long before.

[3](Kim Carter, 2022)

Discovering Early Security Defects

With data breaches a daily occurrence, early discovery and resolution of security defects is critical. Having Security Champions in place, educating and upskilling your development teams, can have a direct effect on early defect detection and on the quantity of defects. Naturally, they can also help remediate and reduce the quantity of defects making it to the next lifecycle stage; and the fewer bugs and vulnerabilities that make it through to release, the more secure your software will be, and the harder you are making it for your attackers.

"When asked about how their data is being protected, 78% of consumers reported being either "concerned or very concerned" [17]

Your adversaries are constantly seeking opportunities to compromise your organization, and the harder that you make it for them, then the greater chance that they will go elsewhere. Though most attacks are targeted, some are opportunistic. However, when opportunistic attackers come up against secure defenses, they won't waste time before moving on to seek easier pickings. Targeted attackers who come up against the same defenses will need to determine what the cost in time, energy, and overall aggravation is likely to be.

If they have the budget, then they will try alternative methods–social engineering and lax user security practices being a common entry point. But if not, then they know there is no shortage of easier targets and, whether funded or not, resources are finite and a decision as to whether to continue or not will need to be made.

Creating Secure Products

With a strong security culture, the right plan, people, processes, and products, you are best equipped to create that secure product. Not only that, but bolstered by the correct team, tools, and techniques, you will be able to deliver that secure product much faster, more efficiently, and also to a much higher quality. However, security doesn't stop just because your product is released. Unfortunately, with only a finite number of resources, many organizations lack the capability to actively test all aspects of their product post-release on an on-going basis. However, there are many hackers—ethical and unethical—who might be happy to test it for you. Ideally, you only want to attract the former, and one commonplace and accepted method of doing so is with a Bug Bounty program.

Bug Bounty Program

A Bug Bounty program is an incentive system where individuals can earn rewards: monetary, reputation/ status, etc., by discovering and reporting software bugs, vulnerabilities, and other weaknesses in your software, defenses, or other access areas. From an organizational perspective, establishing such a program offers several benefits and also broadcasts that your organization is serious about security and your product. So much so that you're willing to offer a financial reward to anyone who can crack your defenses. Naturally, the higher the financial reward then the more weight your argument and program will have, and the greater chance that it will be more attractive to ethical hackers who will divert the time, energy, and resources to put your program and bounty to the test.

Note: if you do implement a Bug Bounty program, then ensure you are true to both your word and your reward. Examples abound where a bounty was offered but when claimed, the conditions turned out to be either too stringent or the company decided to drag their feet on payout. Subsequently, the hackers lost patience, went public with the vulnerability, and the resultant fallout harmed the company in several ways: First, their reputation. Second, the remaining ethical hackers decided to take their expertise where it was more appreciated. Third, the loss of face when the company's 'our word is our bond' claim was stripped bare to reveal that it was, in fact, nothing of the sort.

"95% of all attacks involve some form of social engineering." [18]

Monitoring and Improving

Once your DevSecOps pipeline is operational, it's time to continually monitor and improve all aspects of both it and all your organizational assets. Application Security is never stationary and when the correct tools are implemented, configured, and tested, this delivers high visibility and capability across all environments. Both of which mean faster reaction and more effective management of bugs and vulnerabilities. However, configuring, testing, and tuning your entire system takes time and expertise. When configured incorrectly, much as you are now, you will be flooded with noise, false positives, alerts and alarms. However, when done correctly, you will only see the alerts and notifications that you need to see. Moreover, this will also position you to reclaim the initiative and to then be proactively vigilant in monitoring and improving all aspects of your pipeline security. At this point, you've not only successfully built your DevSecOps pipeline, but also know that it works.

Conclusion

Like DevOps, DevSecOps has never been about tools or technology. Rather, DevSecOps is about adopting a security culture within the organization. A culture where security is everyone's responsibility, and which underpins everything that you do as an organization. However, for those yet to implement DevSecOps, and to ensure success, it's important to be aware of and understand the misconceptions and problems that may arise. First and foremost, it's essential that you understand exactly what DevSecOps will look like in your organization. Second, you must have the wholehearted and active backing of Top Management. People are resistant to change, and Top Management is vital in smoothing out unforeseen bumps. Third, as part of educating your developers, the recommendation is to appoint Security Champions. These champions will not only help change your developer's overall outlook on security, but also their stance, long-standing resistance to security, culture around secure code, and in improving overall security education. Merging security with DevOps has always been the missing link, and is where Shift-left failed. However, implementing the above is fundamental in bringing people, processes, and products together within a security-focused culture and to successfully building a DevSecOps pipeline within your organization.

Questions about this paper?

If you're an experienced CISO and your organization already has a DevSecOps pipeline, then most of what's been written in this paper will be for information purposes only. However, if your organization either lacks this capability or this is mostly new to you, then there is a good chance that you have several questions around tools, times, costs, approach, best practices, previous failures, problems to look out for, etc. Unfortunately, due to the sheer number of considerations, unknowns, options, and possibilities, this extends far beyond the scope of this paper.

Incorporating DevSecOps into your organization and building your DevSecOps pipelines does take time, investment, and will involve several considerations, questions, and conversations. One indication of how involved this can be is with Shift-left and how it remains a pipe dream for many to this day. A DevSecOps pipeline is both doable and worth doing, and we hope you have found this paper useful. Please feel free to contact us with any questions, comments, or feedback.

References

[1] IBM Security, “Cost of a Data Breach,” 30 July 2022. [Online]. Available: https://www.ibm.com/downloads/cas/3R8N1DZJ. [Accessed 12 August 2022].

[2] G. Wilson, DevSecOps: A leader’s guide to producing secure software without compromising flow, feedback and continuous improvement, Rethink Press, 2020.

[3] Immersive Labs, “Research: Imperfect People, Vulnerable Applications,” 13 May 2021. [Online]. Available: compromising flow, feedback and continuous improvement, Rethink Press, 2020.

[4] C. Jones, Applied Software Measurement: Global Analysis of Productivity and Quality, 3 ed., New York: McGraw Hill, 2008, p. 697.

[5] M. Rothman, “Why Your DevSecOps Initiative Will Fail,” 5 July 2022. [Online]. Available: https: //devops.com/why-your-devsecops-initiative-will-fail/. [Accessed 16 August 2022.

[6] W. Kelly, “DevSecOps: 5 tips for seeding a culture transformation,” RedHat, 18 August 2022. [Online]. Available: https://www.redhat.com/architect/devsecops-culture. [Accessed 24 August 2022].

[7] Code42, “Annual Data Exposure Report,” 2022.

[8] Markets Insider, “IBM Report: Cost of a Data Breach Hits Record High During Pandemic,” Markets Insider, 28 July 2021. [Online]. Available: https://markets.businessinsider.com/news/stocks/ibm-report-cost-of-a-data-breach-hits-record-high-during-pandemic-1030653188. [Accessed 18 August 2022].

[9] Nixu Cybersecurity, “Five Reasons Why Your DevSecOps Fails and How to Tackle Them,” Nixu Cybersecurity, 28 May 2020. [Online]. Available: https://www.nixu.com/blog/five-reasons-why-your-devsecops-fails-and-how-tackle-them. [Accessed 19 August 2022].

[10] F. Nagle, D. Wheeler, H. Lifshitz-Assaf, H. Ham and J. Hoffman, “Report on the 2020 FOSS Contributor Survey,” 17 February 2021. [Online]. Available: https://www.linuxfoundation.org/tools/foss-contributor-survey-2020/. [Accessed 19 August 2022].

[11] V. Jakkal, “The passwordless future is here for your Microsoft account,” Microsoft, 15 September 2021. [Online]. Available: https://www.microsoft.com/security/blog/2021/09/15/the-passwordless-future-is-here-for-your-microsoft-account/. [Accessed 21 August 2022].

[12] Tech at GSA, “DevSecOps Guide: Standard DevSecOps Platform Framework,” [Online]. Available: https://tech.gsa.gov/guides/dev_sec_ops_guide. [Accessed 19 August 2022].

[13] C. Chin, “How to Measure Software Development, from ‘Accelerate: The Science of Lean Software and DevOps’,” Holistics Blog, 18 March 2020. [Online]. Available: https://www.holistics.io/blog/accelerate-measure-software-development/. [Accessed 19 August 2022].

[14] Hackerone, “5 Security Stages of the DevSecOps Pipeline,” Hackerone, 28 June 2022. [Online]. Available: https://www.hackerone.com/application-security/5-security-stages-devsecops-pipeline. [Accessed 23 August 2022].

[15] R. Varma, “How to Build a Successful DevSecOps Pipeline?,” Opstree, 28 January 2022. [Online]. Available: https://blog.opstree.com/2022/01/28/how-to-build-a-successful-devsecops-pipeline/. [Accessed 24 August 2022].

[16] K. Carter, Holistic Info-Sec for Web Developers, vol. 0, Auckland: LeanPub, 2019.

[17] A. Zhinitsky, “You Code It, You Own It: Announcing OverOps Support for Git Blame,” 26 August 2020. [Online]. Available: https://www.overops.com/blog/you-code-it-you-own-it-announcing-overops-support-for-git-blame/. [Accessed 25 August 2022].

[18] Privitar, “New Privitar Survey Reveals Business Opportunity to Build Consumer Loyalty Through Data Privacy,” Privitar, 26 August 2020. [Online]. Available: https://www.privitar.com/press-releases/new-privitar-survey-reveals-business-opportunity-to-build-consumer-loyalty-through-data-privacy/. [Accessed 27 July 2021].

[19] T. Koulopoulos, “60 Percent of Companies Fail in 6 Months Because of This (It’s Not What You Think),” Inc, 5 January 2021. [Online]. Available: https://www.inc.com/thomas-koulopoulos/the-biggest-risk-to-your-business-cant-be-eliminated-heres-how-you-can-survive-i.html. [Accessed 22 August 2022].

GuardRails is a platform that provides Application Security for modern development teams and helps them to rapidly experience and achieve True Left within seconds. With GuardRails, True Left means transforming your current insecure DevOps pipelines into end-to-end DevSecOps secure pipelines throughout your entire SDLC. GuardRails instantly bridges the gaping void between DevOps and your security.

With highly parallelized and differential scanning across Static, Dynamic, Dependencies, Secrets, and infrastructure, GuardRails not only rapidly secures your pipelines, but it also helps to break down internal departmental silos while simultaneously solving your development/bug creation problems.

Just-in-time (JIT) training delivers on-the-spot targeted training that educates and guides developers to produce bug-free code. In turn, this means better training and upskilling, increased development velocity, decreased costs, and faster roll-out and scan times.

www.guardrails.io