logdna

Logging For DevSecOps

A LogDNA eBook

INTRODUCTION

If you want to take full advantage of the agility and responsiveness of a DevOps approach, security must have an established, important, and integrated role in your applications' life cycles. Logging can help with that. It will ensure that you have the visibility and transparent communication needed to detect and resolve issues quickly.

This eBook will introduce you to the fundamentals of DevSecOps and how to optimize your logging strategy for this style of work. It covers best practices for using LogDNA for security teams and how and why you should secure your CI/CD pipeline.

What are you waiting for?
Let's dive in.

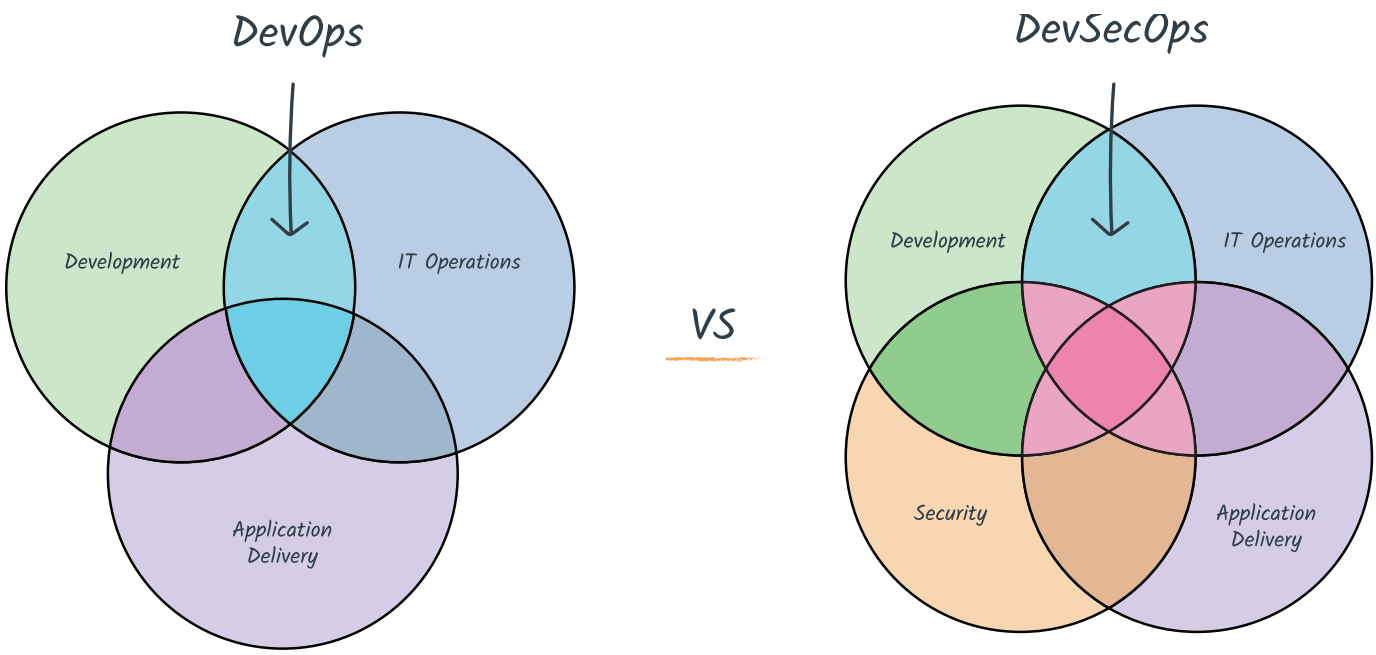
WHAT IS DEVSECOPS

In an automated development environment, a DevOps team is a combination of developers and operations people who work together to speed up software deployment and automate many repeatable procedures that don't need human interaction. During the automation process, vulnerability scans and testing can be added to ensure the safety of data and integrity of the application. A DevSecOps team—short for development, security, and operations—adds security professionals to operations and development staff so that every automated step includes the right standards and protocols that test your applications for common vulnerabilities. Security professionals build protocols and standards built into your DevOps procedures from penetration testing for vulnerabilities to protecting infrastructure from a compromise.

How Does DevSecOps Work?

DevOps is meant to speed up development time, but automation can open new vulnerabilities that won't be detected until the organization falls victim to a cybersecurity incident. DevSecOps tools on the market help improve the security of an application automatically compiled and deployed to production. Many of these tools can also be integrated into current DevOps automation so that developers and security professionals can be alerted to any cybersecurity issues found during a scan without any manual overview during deployment.

In a typical development environment, developers deploy code to a testing environment where quality assurance (QA) runs automated and manual tests on the code. This step is meant to find bugs and other issues in the



application, but it's not meant to test for vulnerabilities. By adding security protocols into the testing and deployment automation process, you can reduce the number of vulnerabilities that could lead to critical data breaches in the future. These security protocols and standards are meant to find vulnerabilities before the code is deployed to production. It's referred to as "shift left" where cybersecurity is implemented automatically during the testing instead of scanning in production.

A typical workflow for DevSecOps is:

1. A developer creates and adds new code to the application repository (e.g., Github).
2. The developer creates a merge request.
3. At this point, DevOps automation compiles the code and then runs a series of tests.
4. Application code is deployed to a staging or testing environment to test before merging with the main branch.
5. DevSecOps automation uses scripted scans to find any common vulnerabilities in the application including configurations that could add the risk of a compromise.
6. If the application passes all tests, it can then be scheduled for deployment to production.

Automated tests check for any configuration issues, application crashes, and bugs that could allow an attacker to execute their own code (e.g.,

buffer overflow). By continually testing the application before it gets deployed to production, developers can offer better security and results and have fewer bug fixes in the future.

The Benefits of DevSecOps

Vulnerabilities in production software can lead to serious data breaches. Some of the world's largest data breaches start from a vulnerability in software. For example, the Equifax data breach started with an unpatched server application program with known vulnerabilities. Although automated tools can't find every vulnerability, they can find common ones that many attackers scan for across the Internet.

Finding vulnerabilities early in the development process isn't the only benefit. Having security professionals integrated with developers and operations helps all three collaborate better. It also helps operations and developers better understand cybersecurity and the many ways infrastructure and applications can be hacked. Developers that understand software vulnerabilities better can create code with fewer bugs and fewer possible risks.

You could have security professionals manually code review and scan for vulnerabilities, but this takes potentially weeks to complete. Manual security reviews are still necessary in some scenarios, but scanning for common vulnerabilities can be automated to speed up development time. Risks can be caught before code is deployed to production, so developers can prioritize bug fixes instead of rushing remediation for a known issue in production.

Compliance is another benefit in having a DevSecOps team or practice. In many compliance standards, testing, patching and monitoring the application are components in cybersecurity requirements. By practicing DevSecOps, you can catch many of the common vulnerabilities that would put your organization out of compliance and could cost millions of dollars in fines. With the right scanning tool, you find unpatched software faster so that you can update it, leaving a smaller window of opportunity for an attacker.

When security personnel work with dev and ops teams, better communication is facilitated among all team members. This will streamline software development, security testing, and deployment.

Vulnerabilities in production software can lead to serious data breaches. Some of the world's largest data breaches start from a vulnerability in software.

WHY LOGGING IS A CRITICAL INGREDIENT IN DEVSECOPS

The conversation surrounding DevSecOps has been happening for several years and most developers now realize that it's important to integrate security into the CI/CD process. Yet what many teams are still struggling to wrap their heads around is how they should go about implementing DevSecOps. It's one thing to discuss integrating security into CI/CD, but another to figure out practical approaches to doing so.

This chapter explains one key strategy for implementing DevSecOps: logging. As we'll see, effective logging from across the CI/CD pipeline helps lay the foundation for a development operation that bakes security right into the CI/CD process and enables the type of collaboration between all stakeholders that is essential for DevSecOps.

Logging As a Pillar of DevSecOps

No matter how you approach DevSecOps, it's hard to imagine an effective strategy that doesn't include logging – not just for production applications and the infrastructure that hosts them but logging across the entire CI/CD pipeline. In a variety of ways, logging reinforces the practices and goals at the heart of DevSecOps.

Shared Visibility

You can't practice DevSecOps if your developers, IT engineers, and security engineers lack shared visibility into the state of each application release and which features are coming next (and therefore need to be secured).

You could try to gain this shared visibility by asking the various stakeholders to collaborate manually. They could hold meetings, chat on Slack, and so on. Yet while some live collaboration is always helpful, you're unlikely to be able to achieve complete shared visibility through manual collaboration alone.

That's why it's also critical to leverage logs as a single source of truth to provide visibility into the pipeline. When security engineers, developers, and IT engineers have access to log data from across the pipeline,

they can use that data to assess its state. As a result, they are better able to find security issues that other stakeholders may have overlooked and prepare for whichever security challenges may come next as they test, deploy, and write new code.

Fast Resolution of Security Incidents

Along similar lines, when a security issue arises in production, developers, IT engineers, and security engineers and analysts need to react quickly and efficiently to resolve the problem.

Here again, logging is critical for enabling fast and coordinated responses. Waiting on manual sharing of sensitive data can slow down reaction times. But when every stakeholder can get the data they need from logs, access to information is no longer the weakest link in the security incident remediation process.

Continuous Improvement

Like DevOps, DevSecOps emphasizes the principle of continuous improvement, meaning that security operations should become more effective over time and able to accommodate more and more types of threats.

It's impossible to know how well your team continuously improves on the DevSecOps front without log data. Using application logs to track metrics such as the frequency of security vulnerabilities within each application release or the number of security bugs per line of code enables teams to log data about the CI/CD pipeline and security operations. Ultimately, this increases developers' ability to track the effectiveness of detection and remediation efforts.

How often do you perform rollbacks in response to a security issue? How long does it take to detect and remediate vulnerabilities caught in the pre-deployment stages of the pipeline? By tracking metrics like these, teams know whether they are trending better or worse in their ability to find and manage security issues, which is one vital proxy for overall DevSecOps health.

Logging As a Key to DevSecOps

Again, there are many ways to implement DevSecOps, and many resources to drive DevSecOps success. Some will vary depending on the nature of the software delivery pipeline. For example, a team that deploys to Kubernetes clusters in a public cloud will rely on somewhat different DevSecOps tools than one that deploys to on-prem VMs, because of the security vulnerabilities and metrics associated with each environment vary.

Virtually any organization and any type of CI/CD pipeline requires logging for an effective DevSecOps strategy. Logging provides shared visibility, easy access to information, and self-assessment opportunities, all of which teams need to make the most of DevSecOps over the long term.

DevSecOps emphasizes the principle of continuous improvement, meaning that security operations should become more effective over time and able to accommodate more and more types of threats.

LOGDNA BEST PRACTICES FOR SECURITY TEAMS

If you don't already have security integrated into your development process, some staff structure changes are often necessary. Adding security staff to your development team should be a painless process, but you should build some best practices into your new structure. These best practices will help you not only with logging but using automation testing for bugs and adding security scans to your process.

Best Practices

Automate Repeatable Processes

Anything that doesn't need manual interaction should be automated. An automation tool can be used to ensure the software compiles without issues, scans for bugs including ones that create vulnerabilities, and identifies configuration issues. By adding security scans into the automation process, you can cut down on delivery time from manual code reviews.

Teams that work with a DevOps mindset use several tools to automate software delivery, and each tool has its own pros and cons. Find a security scanning solution that fits well with your current code deployment and delivery tools.

In addition to this, it's important to educate developers and operations on the latest threats and risks. Developers who better understand cybersecurity will keep vulnerabilities in mind as they structure their code. When developers understand cybersecurity, they are less likely to deploy buggy software and deployment will be faster.

Log Meaningful Information

You can customize the information stored in logs to help them identify exactly where and when an error occurred, but events should be meaningful for human reviews. For example, the following information doesn't tell you much about an error:

```
Unhandled Exception:
System.IndexOutOfRangeException: Index
was outside the bounds of the array
```

You know what happened is that a loop likely attempted to retrieve data from an index that does not exist in an array, but when did this happen? Where did it happen? Who received the error?

An even better log looks like the following:

```
Unhandled Exception:

System.IndexOutOfRangeException: Index
was outside the bounds of the array.
Transaction ID 47492389 failed on
2021-01-28T20:04:13Z at /checkout/pay.
```

In addition to the exception, the above information tells you the transaction, date, time, and the endpoint where the error occurred. A collection of more verbose error information better helps root cause analysis during bug fixes and remediation. For system administrators with hundreds of machines to manage, more information gives a quicker overview of the problem coupled with the information that both locates the machine and the issue causing errors.

Create Logs Using Structured Formatting

Formatting logs in a structured way is beneficial for two reasons: it makes reading easier for humans and machines. At some point, you may need to import large volumes of logs. In a large enterprise environment, infrastructure and applications could create thousands of events a day making it difficult to use them for analysis without third-party tools. Import logs into an analytical solution to parse data and get visual output that represents the health of the system. For example:

```
APP:commerce Transaction:47492389
TIME:2021--01-24T08:38Z ENDPOINT:/
checkout/pay
```

The above log event uses a structure that makes it easier to import data into an analysis tool. It also structures the event so that humans can better identify important information when reviewing logs for specific data.

Logs, while important, are not the most intuitive data types to work with. It is up to development teams to structure raw logs in a way that makes them human readable. LogDNA can help with these efforts by automatically parsing most major log types, as well as by providing templates to help build custom parsing capabilities to meet any team's needs.

Include Logs in Backup Routines

Logs can be parsed and used in data recovery efforts, but administrators often forget to make them a part of the backup process. Logs should be backed up just like any other critical file. Should an event such as ransomware permanently destroy or damage logs, they can then be restored and used during forensics and data recovery.

Somewhat related, logs should also be redundant just like data and storage. Should one logging solution fail, the system can use the alternative until system administrators can restore the original. This strategy requires more storage space, but the cloud gives organizations the ability to scale storage to accommodate increased requirements.

Don't Log Sensitive Information

While logs should contain enough information for audit trails and root cause analysis, the information should not expose sensitive data. Only specific accounts should have access to logs, but creating events with sensitive information adds risk of threats should an attacker compromise security surrounding logs. Not only could logs be used to mount additional attacks, but they could also violate compliance rules.

In addition to always determining if data improperly discloses personally identifiable information (PII), here is a short list of items that should not be logged:

- **Passwords**
- **Social security numbers**
- **API keys or secrets**

- **Private encryption keys**
- **Credit card numbers**

Centralize Log Aggregation

It's easy to get overwhelmed with numerous log storage locations, system monitoring solutions, and application alerts. Centralized logging reduces much of this overhead and eliminates many of the issues with fragmented log storage across several systems. It also facilitates better analytics, especially if these logs are imported into third-party tools.

Centralized logging solutions like LogDNA also provide easier management for backups, cybersecurity, and monitoring. It mitigates risks of losing logs and information and provides collaboration between several individual resources and monitoring solutions that use events to determine anomalies to alert administrators of suspicious activity.

Don't Forget Endpoint and Device Events

Any network resource that provides critical infrastructure or adds risk to the organization should be included in log strategies. In an enterprise environment, users could potentially have their own devices connected to the network, and mobile applications might connect to internal processes using API endpoints. A component of good event logging strategies includes collecting data from devices where users are able to connect to the network.

By logging events on all endpoints, it helps you understand the user experience and interpret feedback from application activity. It also helps administrators recognize bottlenecks and scale resources before they create severe productivity limitations.

Limit Logs to High-Privileged User Accounts

If logs are disorganized, low-privileged users could accidentally have access to sensitive information. Centralizing your logs with a solution that offers Role Based Access Control can help you manage who has access to what information. For example, you may want to provide high-privileged users access to read all raw data logs but only allow standard users to see logs from certain sources or visualization tools that provide high-level overview of systems and applications.

Allowing unnecessary access to logs increases your attack surface. If just one user falls victim to a phishing scam, logs would disclose information that can be used in future attacks. It also provides critical information about the infrastructure of network resources and applications. An advanced persistent threat (APT) giving attackers access to logs could provide them with numerous data points leading to a severe compromise and systemwide breach.

Log Successful and Failed Events

Not every anomaly results in a failed event such as an application error or an unsuccessful authentication attempt. To get the full picture, administrators need several events that tell a story during investigations. Without enough events, anomalies and suspicious activity could be missed. For instance, a cyber-criminal launching brute-force attacks against account passwords would show several unsuccessful authentication attempts, but logs would not show suspicious activity from stolen credentials and successful authentication after phishing attacks.

Administrators should develop a strategy for events that should be logged. Too much information makes logs undecipherable and wasteful of storage, but verbose logs with useful information can be used in effective monitoring and auditing.

Conclusion

Logging is essential in enterprises for an investigation into cyber-events, monitoring, root cause analysis, forensics, and overall maintenance of your systems. Developing a strategy before implementing logging solutions is just as important as the actual logging solution. Before diving into logging solutions, ensure that you put together a plan and follow best practices.

Continue on to the next chapter, where we'll give you practical ways to build a secure CI/CD pipeline.

FIVE PRACTICAL WAYS TO BUILD A SECURE CI/CD PIPELINE

Given the seemingly unending stream of cyberattacks, most developers don't need anyone to remind them that securing development pipelines is important. But what eludes many teams is how you design and implement a secure CI/CD pipeline. It's one thing to talk about CI/CD security and another to put it into practice.

Here's a look at five practical strategies for securing CI/CD pipelines with that reality in mind. Although not every one of these practices will make sense as a security strategy for every team and pipeline, most organizations can benefit from these processes to bolster CI/CD security.

Why Secure the CI/CD Pipeline?

Before delving into practices for securing the CI/CD pipeline, let's explain why CI/CD security is essential.

To some developers, the importance of securing CI/CD processes may seem so obvious that it's not even worth discussing. However, because most CI/CD pipeline stages occur before applications are in production, it can be easy to overlook security there and instead focus on securing production environments, where the most significant risks tend to exist.

It's true that most threats don't touch your applications until the deployment stage of the pipeline or later. Yet, because the vulnerabilities that those threats seek to exploit are often introduced in earlier stages of the pipeline, baking security into all stages of CI/CD is critical for delivering applications that are as secure as possible in production.

In addition, certain risks, such as insecure management of secrets within development environments, can creep into the pre-deployment stages of the pipeline. That's another reason to secure all stages of the pipeline.

Five Best Practices for Securing the CI/CD Pipeline

Now, let's look at actual practices for keeping CI/CD pipelines secure.

Add Security Tests to Your Testing Routine

You probably (hopefully) already run pre-production tests on your application releases to vet them for reliability and performance. Doing so is a best practice for ensuring quality releases for your end-users.

But it's equally important to include security tests within your testing. Although you may not be accustomed to thinking of security testing as something that you can do alongside performance testing, the fact is that you can extend frameworks like Selenium to perform security tests, too. See this SQL Injection as an example.

So, if you're not already running security tests as part of your testing routine, now's the time to start. After all, performing security testing early in the CI/CD pipeline is what "shift-left" security is all about.

Leverage Feature Flags to Manage Risks

Developers often use feature flags (or feature toggles) to add new features to applications while mitigating the risk that those features will introduce performance or stability issues. By using "flags" to turn the features on and off, developers can easily integrate them into the codebase while retaining the ability to turn them off quickly if necessary.

Although most developers likely don't think of feature flags as a security tool, they can be used for that purpose, too. After all, new features are prime vectors for security vulnerabilities, especially if developers haven't thoroughly tested them yet.

By using security flags to disable new features within production environments until they have appropriately vetted them, developers can reduce the risk that those features will cause security issues. In this way, feature flags help developers continue to improve their applications while ensuring the associated risks are always in check.

Use Roll Backs to Manage Production Security Bugs

No one likes a rollback, which means reverting an application to an earlier version. Unfortunately, though, rollbacks are a fact of life. Sometimes, the fastest and smoothest way to solve an issue is to revert the application to a known, stable version.

That's true not just when performance or stability bugs arise but also in the case of security issues. If you can quickly roll back an insecure release, you can mitigate the amount of harm caused by inadvertently pushing such an application into production.

Designing for fast and reliable rollbacks from the beginning is a best practice for overall CI/CD security. Just keep the binaries for earlier releases on hand so that you can redeploy them if necessary. Make sure your deployment tools can efficiently pull one version of the app out of production and replace it with another one quickly.

Securely Saving Secrets

It can be tempting to cut corners with secret management in the development and testing stages of the pipeline, even for developers who are dead serious about securely managing passwords, SSH keys, and other secrets in production environments.

After all, only trusted internal users would have access to dev/test environments in theory. It wouldn't seem like a big deal to do something like letting multiple users share a single account or hard-coding passwords into source code, then removing them before deploying the application.

The reality, of course, is that these are not best practices from a security standpoint. How easy is it to forget to remove a hard-coded secret before deployment? Shared accounts are never a scalable route, even if it's just for dev/test.

The solution to risks like these is to apply the same security standards to environments used during the dev/test stages of the pipeline as you do to production. To simplify things, use secret managers to store credentials securely, even if it's just for testing. It's crucial to require each internal user to have a separate account. And, most critically, do not store sensitive data without enforcing authentication and authorization rules to govern it, even if it is only accessible from a local or private network.

Anomaly Detection Across the Pipeline

You probably already use a SIEM or similar security platform to detect anomalies within production environments that could be signs of a breach. That's an essential best practice for securing user-facing applications.

An even better best practice, however, is to detect anomalies at all stages of the pipeline. That means analyzing logs from dev/test environments and logs from various CI/CD tools themselves. Any patterns in this data representing a departure from the norm could signify something is off – such as a misconfigured file, failure to adhere to internal governance rules, or a security issue within upstream software (like open source libraries) that you incorporate into your application.

LogDNA provides a centralized place for all of your log data. Logs are the single source of truth for everything that is happening in your environment. So while SREs use logs to troubleshoot issues that are happening in production, developers might use them to understand how their code is performing before the pre-commit.

The more data you scan for anomalies and the more representative that information is of the entire pipeline, the greater your ability to detect and address vulnerabilities before the software is in production.

Securing the Entire Pipeline

You can't secure your application without securing your CI/CD pipeline. Although most real-world threats don't materialize until the software is already in production, the goal of developers should be to leverage practices that minimize the risk of active vulnerabilities reaching production environments in the first place. The earlier the better. These actions consist of security tests, feature flags, and across-the-pipeline anomaly detection. The more secure your CI/CD pipeline is, the fewer security issues you'll have to contend with once your code is in production.

By leveraging best security practices, you can minimize the risk of active vulnerabilities from reaching production environments.

CONCLUSION

Throughout this eBook, you've learned about the importance of DevSecOps, logging, and the intertwining of them both. You've also learned a few tips from us to start you on your journey or if you've already made the first steps, help you along the way.

It would be an overstatement to say that logging and log management are the only essential ingredients in DevSecOps.

A lot of other factors come into play in order to achieve a successful DevSecOps culture, such as sustainable delivery and having scalable governance models. However, logging and log management are often unappreciated yet critical facets of DevSecOps. You can't have DevSecOps if you don't manage logs effectively.

So the question remains. Are you looking for a way to put DevSecOps principles into practice? If so, your logging strategy is a good place to start.

logdna

Thank You

Sales Contact: outreach@logdna.com
Support Contact: support@logdna.com
Media Inquiries: press@logdna.com