DevSecOps – Securing the Stack

Date:	May 2017
Doc Ref:	SOS-WP-DSO-517
Version:	1.0
Authors:	Michael McKinnon & Murray Goldschmidt

Table of Contents

Overview

As enterprises continue transforming their IT methodology through the culture and practice known as DevOps, the need to address security concerns cannot be ignored.

With a myriad of early adoption developer tools being employed to dramatically improve throughput in support of the service delivery lifecycle, there exists multiple opportunities to secure the entire system (the stack) to minimise cyber risk.

While some common DevOps practices lean towards security best practice almost by accident, the need to provide visibility and leverage automation are key aspects of DevSecOps.

This document aims to help security managers, software developers and business stakeholders understand how they can help secure DevOps environments with a DevSecOps approach, holistically addressing the entire production system.

For further information on DevSecOps we invite you to read our earlier whitepaper entitled “DevSecOps – Agility with Security”[1].

The information contained herein has been compiled from the collective experience of the security consultants at Sense of Security. Through previous engagements and security assessments, we have observed what does and doesn’t work.

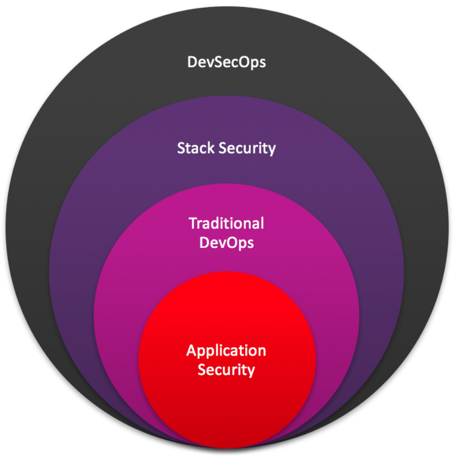
This paper is not intended to be a comprehensive “how-to” guide, nor does it cover the precise tools and methodology that Sense of Security has devised when conducting such engagements.

However, we do hope this whitepaper provides you with valuable insight into the primary areas of concern and obvious opportunities in DevSecOps, as well as exposure to key objectives and discussion points that will continue to evolve over time.

What is DevSecOps?

It is important at the outset to understand what we consider to be in scope when discussing DevSecOps, and particularly our concept of “Stack Security” in this context.

Traditional Application Security is primarily concerned with the quality of the production output. As such, it generally focuses only on the security posture of the service or application being delivered from the DevOps pipeline.



Our view, however, is not limited to the security of the application or service, it also includes the inherent security of the entire production system – including the technology, tools and practices used to deliver it including the cloud, development and production environments, covering the people and culture of the entire DevOps stack.

Hence our view of DevSecOps is high-level and holistic and covers the entire development production system. This scope should be taken into consideration when reading this document.

[1] https://www.senseofsecurity.com.au/research/it-security-articles

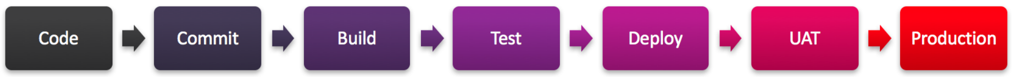
Service Delivery Lifecycle

Modern software development has evolved to a point where it almost always involves delivering a service continuously. As such, we argue that SDLC now more appropriately stands for ***Service Delivery Lifecycle***, as opposed to Software Development Lifecycle.

The notion of a service being delivered continuously means that modern software projects no longer require a fixed completion date. As researchers at Facebook openly acknowledge, "Sites like Facebook will never be completed. The mindset is that the system will continue to be developed indefinitely."[2]

This never-ending-development mindset brings about a major shift in thinking when applying traditional security techniques and controls in high-velocity DevOps environments. Added pressure for change also exists when you consider that web applications aren't evolving only in response to demand for new user features and performance, but also increasingly for security – and by extension, this means there is a proven market requirement for continuous security testing, validation and protection to be in place.

With development flow generally following a predictable and mature path from Code to Production (see below) we need to examine the ways that security can be implemented and automated to address the enterprise risks according to cyber risk appetite.



At each discrete phase, there are handover points already exploited by processes like code commit, continuous integration trigger points, deployment scripts, and automated builds – all of which provide opportunities to inject security tests that become integral to the pipeline that help determine success or fail with each build.

Therefore, providing a continuous Service Delivery through a robust DevSecOps environment must be achieved through providing Information Security practices that are delivered continuously and constantly, in step with the modern service delivery lifecycle.

Lastly, when it comes to developers, providing a quality application at scale requires that they're able to support the continuous operational use of their software. Developers can no longer avoid the responsibility of how their code performs in a production environment. This results in the need to address the cultural aspects that work so well in DevOps, but which now include the security team working cohesively in a DevSecOps cohort.

Identifying the Stack

A typical DevOps technology stack consists of a mix of many open source and commercial tools. This includes infrastructure and cloud platform providers, and specialised development tools, services, and libraries covering:

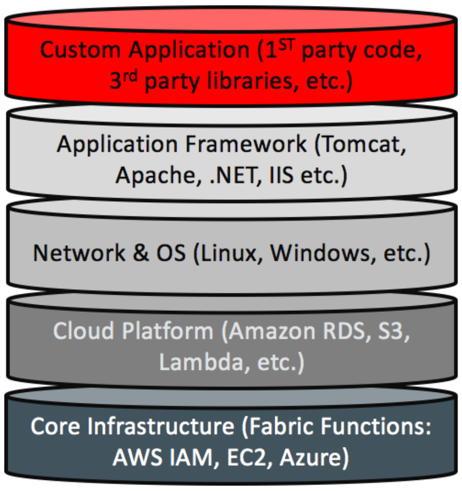
- Infrastructure/Platform
- Code Repository
- Configuration Management
- Build and Orchestration / Provisioning
- Test Management
- Monitoring

[2] https://research.facebook.com/publications/development-and-deployment-at-facebook/

- Deployment Environment

However, this view of the traditional DevOps stack is centred only around the tools and procedural categories, and from a security standpoint is, in our opinion, short-sighted.

Therefore, our view and concept of the "Stack" from a DevSecOps vantage point is one that encompasses not only the tools and processes that support the production of the application, but also the systems and practices that drive the entire production system.


Custom Application (1ST party code, 3rd party libraries, etc.)
Application Framework (Tomcat, Apache, .NET, IIS etc.)
Network & OS (Linux, Windows, etc.)
Cloud Platform (Amazon RDS, S3, Lambda, etc.)
Core Infrastructure (Fabric Functions: AWS IAM, EC2, Azure)

The Stack Security view allows us to move up and down through layers of individual concern that address not only the traditional concerns of application security from a code viewpoint, but also to address the need for security to exist at all layers down to the core infrastructure being used in the environment.

Our referring to the DevOps environment therefore is holistic and includes not only any publicly facing infrastructure, but also test and developer environments. Anything that touches the production system that is IT based that poses a security risk.

When it comes to major efficiencies in adopting this model for Stack Security, automation and orchestration (which has also played a huge part in DevOps itself) underpins the success here. Security practices in DevSecOps will only thrive through the ability to make tasks repeatable, and in ways that continually improve them and their associated risk mitigating properties.

Rarely is a technology product in the stack standalone, as each tool and platform invariably contains provision for automation or integration of some kind; we now live in a world filled with API's and SDK's for everything. This not only provides for speed in the development process by eliminating manual data handling, but also reliability and stability – and importantly should be eagerly exploited by enterprises embracing DevSecOps.

In summary, automation is vital in the stack, and while some vendors are beginning to adapt their tools to cater for better security through DevSecOps, in our experience there is still vast room for improvement.

Primary Objectives of Stack Security

The primary objective of a traditional DevOps environment is to improve throughput of the SDLC production system, while satisfying whatever business goals it must also deliver upon. Therefore, when addressing security concerns in DevSecOps, we assert that it must, at a minimum, not detract from the throughput, must also align with the overarching business goals, and must also improve security outcomes towards cyber resilience and risk minimisation.

In traditional IT environments, security teams are often seen as bottlenecks as they expend time and resources devising and implementing controls to protect the enterprise. However, in a DevSecOps environment such delays are not acceptable and allowing this to occur would mean the throughput of the entire system is limited by the security team.

Additionally, one of the challenges is that existing developer and operations teams continue to utilise tools and methods where security in the SDLC is either ignored, overlooked, or not seen as a primary concern.

In DevSecOps, security can no longer be implemented as a one-off or standalone undertaking, and therefore we must look for ways to automate and streamline many of the repetitive tasks that can be used to easily identify low-hanging-fruit issues.

Leveraging the power of automation for security tasks can also free up valuable human resources that can then focus on improving the production system, or dig deeper into non-trivial issues like finding business logic flaws in applications which is not easily automated. A more efficient use of resources results, which can only be good for the bottom line.

The Primary Objectives we're aiming for with Stack Security are as follows.

Communicating security as a primary concern

It is no longer optional for enterprises to implement good security, it is now a requirement. This means finding ways and means to adopting a DevSecOps environment and identifying issues in the build cycle can lower attack risk, and improve resilience and robustness of your entire enterprise. You'll win favour with consumers who reward companies that respect security and privacy and that punish companies who don't!

Security must keep up with speed of delivery

DevSecOps can only be successful if it contributes meaningfully to the existing DevOps movement without negating any of the benefits of speed or reliable service delivery.

Surround dynamic processes with protection

Every stage of the production lifecycle is a potential opportunity to provide protection, or inspection. For example, dynamic or static code analysis can be automatically triggered immediately when code is committed to a source code repository. And continuous scanning can be used to report critical changes to all environments.

Automate security tests as code

Security testing that is done right once, with tests set in code (just like application code) is one of the best ways to begin realising the benefit of DevSecOps. Discard complex long range security roadmaps in favour of a "baked-in" security approach. This will address security incrementally in achievable and actionable chunks inside the core of your development pipeline.

Quality at Source

Maintaining quality at the source is based on the premise that if you close the gaps and variation within the SDLC you will eliminate problems from occurring, before they occur – this is practically achieved through triggered tests that are fully automated. Recycling known good configurations and source code modules also eliminates wasted development and deployment cycles and provides confidence about the integrity of the building blocks of your platform and the application delivered on it.

Identify Defects Earlier (Shifting Left)

As defects progress through the development pipeline undetected, the cost associated with remediation grows significantly. It is important to keep "shifting left" – whereby priority is given to techniques that will identify defects earlier in the pipeline.

Securing Application Code

Developing application code that is as free from defect as it can be, and that conforms to recognised standards is a critical part of any effective Application Security program.

When we look at the DevOps pipeline and the opportunities for implementing security in a highly-automated way we consider two aspects – the code written in-house by your development team (including operations staff who are often writing code too) and third party libraries that are utilised.

First Party Code

Any application code, regardless of language, platform, or purpose that is authored by members of your enterprise can be considered "first party code".

Code snippets or sections of source code that have been copied from other sources, for example from popular websites such as StackOverflow should also be considered as first party code, even though they may be thought of as third party code.

Nissan Corporation learned this lesson the hard way in 2016 when artefacts of cut-and-pasted code appeared in a published mobile application.[3]

While we discuss third party code in the next section, it is also important to draw a distinction that we do not consider entire shared bodies of code – even if they are provided as source code – to be first party code. However, the moment any line of source is modified in-house in an externally provided library is the moment it should then be considered first party; as is widely accepted with forking any open source project.

When it comes to automating a security solution at this layer, as you might expect since we're operating at the Integrated Developer Environment (IDE) there's no fully-automatic option, but if we can reduce the work of the developer to a simple button-push or menu-item-click, then we can achieve something.

Providing security enhancements at this layer then becomes a function of several things.

1. Ensuring the code author is sufficiently trained in the principles of secure coding, such that they understand the larger ramifications of their actions;
2. Providing the code author with advanced scanning and code analysis tools, but presenting the information in a clear and concise manner that makes sense;
3. Using code helpers in the IDE turned on by default that are always "linting" the written code to detect and alert the author to non-compliant coding.

In practice, we address these three key challenges by implementing training workshops for developers, suggesting code analysis techniques that can be integrated into the development pipeline, and setting policies around minimum coding standards.

What can be automated however, is the security scanning of software projects or compiled applications, and this is normally achieved through a trigger at the Continuous Integration – Continuous Deployment (CI/CD) layer of your DevOps stack.

[3] https://www.theverge.com/tldr/2016/5/4/11593084/dont-get-busted-copying-code-from-stack-overflow

Third Party Code / Supply Chain Risk

Using shared software libraries, either in compiled or non-compiled form that are available freely from many repositories on the Internet, saves considerable time and development effort. But there are latent associated risks that must be dealt with.

We consider any shared body of code not authored by your development or operations teams to be third party code. As such it must have a different treatment applied to it, as the risk and exposure is considerable.

Embedding third party code in your application projects provides clear benefits on many levels, from the advantage of using thoroughly tested code, to not having to invest in authoring complex algorithms such as cryptography, or image processing for example.

In fact, history has shown that for complex applications such as cryptography, it is far more desirable to use a shared library to provide assurance that the cryptographic implementation is correct and robust.

Sony's PS3 gaming console in 2010, for example, was found to have a weak random-number generating algorithm, and as a result the hardware device protections were defeated.[4]

Research shows that modern software development practices can include up to 90% of third party and shared libraries.[5] Easily obtained software components and libraries from untrusted sources provides exposure to vulnerabilities. Even when developers are diligent about using secure libraries, they may not be aware that those libraries use other libraries of their own, resulting in latent exposure.

However, should vulnerabilities be identified in third party libraries, this means the exposure and impact of the vulnerability is experienced by all those organisations who have utilised the same library. Hence the damage spreads much further as a result.

This was most evident with "Heartbleed" (CVE-2014-0160)[6] which was a vulnerability in the OpenSSL library that allowed memory to be leaked, resulting in serious information disclosure. To this day there are still hardware appliances and older versions of software that continue to be vulnerable to Heartbleed, further proving the catastrophic scale that a major flaw in a third-party library can create.

Providing security enhancements at this layer then becomes a function of several things.

1. Examining and cross-referencing third party libraries used in your development pipeline against known vulnerabilities, including CVE's and CVSS;
2. Implementing a policy and/or mechanisms that prevent high-risk libraries from being utilised in your development pipeline;
3. Inspecting third party libraries that contain source code as you would for first party code to consider opportunities for forking and improvement.

In practice, we address these three key challenges by suggesting platform tools and IDE plugins that allow for the inspection and identification of known-vulnerable third party libraries, and set policies based on risk appetite that prevent latent unwanted exposure.

[4] https://arstechnica.com/gaming/2010/12/ps3-hacked-through-poor-implementation-of-cryptography/
[5] "Continuous Acceleration: Why Continuous Everything Requires A Supply Chain Approach", Joshua Corman - https://www.its.fh-muenster.de/owasp-appseceu/2015/
[6] https://www.us-cert.gov/ncas/alerts/TA14-098A

Securing the Application Delivery Framework

Beneath the application code (consisting of the first and third party code components) is the delivery framework layer in our stack view. This layer consists of server applications or frameworks with similar areas of concern that assist the application in being delivered, but are not integral to the functioning of the underlying operating system.

The most common delivery framework for a web application will be the web server daemon serving up the content via HTTP or HTTPS, but may also include the language interpreter such as PHP, or .NET runtime for example.

At this layer, we're often concerned from a security perspective with items such as:

- Configuration management
- Performance monitoring
- Audit trails and logging
- Vulnerability and Patch management

Many of these items, while also relevant at the operating system and network layer, are quite distinct and often separate as we're dealing specifically with the framework that is closely bound to the application code.

In a modern DevOps environment, where we want to integrate security concerns in a highly-automated way to maintain current levels of throughput, there are several opportunities that exist.

Configuration as Code – the need to understand the state of your configuration at any point in time, and to have the assurance that it hasn't been tampered with cannot be underestimated. Through the same principles of best practice DevOps, it is critical that the configuration state of the delivery framework is incorporated into the deployment process, and that the configuration is stored in a version controlled code repository.

Configuration should never be conducted manually, and additional file monitoring measures that detect configuration changes in live environments should be considered.

The other reason why the application delivery framework configuration should be tied to your application code is that when your application expects certain features in the framework to be activated they need to be available. For example, authentication and session management can be implemented by the framework and made accessible to the application through configuration – in this case, directly addressing A2 in the OWASP Top 10 aimed at preventing broken authentication and session management.

Moreover, it should be noted that any hardening configurations, or additional security plugins or library configurations are assumed to be part of the overall configuration of the application delivery framework and should be treated as code.

Performance & health checks – availability of systems, regardless of whether they are in a publicly facing production role is an important factor in a DevOps environment that is expected to maintain operational throughput.

An overlooked aspect we have observed includes checking for predictable future failures, such as SSL/TLS Certificate expiry events related to HTTPS delivery. If a certificate expires on a public facing web application, the impact is easily noticeable to customers. Yet, simple continuous scanning that can inspect the certificate expiry date and provide alerts well in advance so this does not have to eventuate.

Moreover, with new freely available SSL/TLS certificate services such as Let's Encrypt[7] it is possible not only to monitor and alert for the upcoming expiry, but also to automatically renew the certificates as well – without any user intervention.

Audit trails & logging – recording activity in an application can be performed by the application code, but there is usually always a degree of responsibility imposed on the application delivery framework. Security teams are better practised at understanding and interpreting log files and monitoring output in a production system, hence they need to be involved in the process.

Meanwhile, developers should ensure they're collaborating with the appropriate security experts to ensure the minimum audit and logging requirements are met. Naturally, all applications need to log relevant actions including user identification, type of event, date and time, success or failure, and origination of event. All privileged actions must be auditable, including any changes made by an administrator.

There are already mature products and automated means that accomplish the common goals for this requirement, suffice it to say that audit logging general happens automatically by default, and then it's a matter of ensuring the integrity of the log files.

One of the emerging areas is the concept of Runtime Application Self Protection (RASP) whereby either the application delivery framework, or components at this layer are able to detect potentially malicious activity in the application and dynamically block or reroute the application logic to prevent any impact.

We continue to perform research in this area, and collect information on how this is being achieved and if it is successful. At the time of writing this, we believe there is too much variation and experimentation to consider a standard approach.

Securing the Network & Operating System

We view Networking and Operating Systems together at a discrete layer in our stack view on the basis that the two are often intrinsically linked.

System hardening guides often include measures that impact on network configuration and routing, although there is some overlap in cloud environments, for example AWS Security Group configurations.

We also consider that system services included with the operating system that provide network services are included in this layer, for example the Secure Shell (SSH) service.

Providing security enhancements at this layer then becomes a function of several things.

1. Continuous network scanning (either through external means, or agent based methods) is deployed and that it provides visibility always (not point in time);
2. Ensuring that operating systems are hardened according to best practice and that all configurations are captured and stored in immutable ways;
3. User account creation and credential storage is managed efficiently with individual accounts and certificate based access, or multifactor authentication;

In practice, we address these three key challenges by suggesting scanning tools and vulnerability management programs, advising on CIS standards and other best practices for hardening, and guiding implementations to use strong authentication mechanisms.

Accordingly, we address common areas of concern and high impact as follows.

[7] https://letsencrypt.org/

Vulnerability management – establishing a baseline and then continually scanning for the introduction of new vulnerabilities in an environment is critical. Fortunately, many security teams have extensive experience with this already; knowledge that can be instantly leveraged in a DevSecOps environment.

Patch management – if vulnerability management is the process that lets you see the vulnerability, then it's patch and configuration management that lets you act on it. Patch management can and should be completely automated where possible. Where it is not automated, there should be compensating controls placed around legacy systems, or work should be underway to migrate to systems that can be automatically patched.

Use Immutable Systems – an immutable system cannot be changed, and if deployed and implemented correctly can reduce or even eliminate external tampering of production infrastructure. Through the systematic separation of code and data, and making systems read-only, high levels of security can be achieved.

Adopt Phoenix Upgrades – just like the mythical "Phoenix" bird that would burn and rise again from the ashes, this concept involves a one-way path when it comes to upgrading production systems. Instead of applying patches and upgrades to running infrastructure, you only upgrade pristine system images, updating the approved version in the source repository as the new known-good, and then you "burn" or discard your production systems and replace them with the new images through a phoenix upgrade.

Embrace end-to-end continuous deployment – for existing DevOps teams that are not yet using automation or orchestration, from a security perspective they are missing out on many advantages. Automating security processes, some of which can be very time-consuming, such as hardening systems are a must these days. This item alone has been the catalyst for some to consider adopting DevSecOps practises.

Securing the Cloud Platform & Infrastructure

The last two layers of our stack security model are where we consider the need to integrate security and automate high-level actions for good cyber resilience.

Regardless of your cloud platform and infrastructure, for example Amazon AWS, Microsoft Azure, Google Cloud Platform, or others including on-premises and virtualised, special care must be taken to address key risks.

Providing security enhancements at this layer then becomes a function of several things.

1. Restrict access to root account credentials that provide broad access to the entire cloud infrastructure;
2. Monitoring tools that provide visualisation to aid in best practice configuration, and alert for configuration changes in production systems;
3. Configure authentication for staff into cloud accounts using federated access where possible and with multifactor authentication.
4. Implement network flow logs and capturing of data for forensics purposes.

In practice, we address these key challenges by guiding policy on the storage of critical credentials, suggest tools and systems to implement that provide continuous monitoring of cloud accounts, and advise on authentication platforms that can be leverage for secure access for staff.

In regards of securing cloud account access, for an Amazon AWS account for example, this includes the root credentials given to the account holder which must be kept as secret as possible, and never used in the day to day operations of the account. It is already possible for AWS accounts to be “taken over” through nothing more than poor configuration, locking enterprises out of their own cloud infrastructure.

Summary

Introducing security into a fast-moving high-velocity DevOps production system is possible, through automation and orchestration, provided the model used is capable of address the complete needs across all layers of the technology stack, and appropriate to the risk appetite of the enterprise.

As transformation of the understanding of applying IT security to the modern SDLC continues, we will hopefully see increasing uptake and awareness for the importance of security – and certainly we are encouraged by that from our client base to date.

However, some challenges remain in this space. Notwithstanding, it continues to move at pace, including the development and release of new tools and methods for addressing many of the concerns communicated in this document.

What may be relevant to DevSecOps today, may be quickly made irrelevant with new ways of working – for example, in containerisation, significant shifts are occurring that are starting to appear be reflected in popular cloud platforms.

Despite the fast rate of change external to your enterprise however, the right approach should still be one of continuous learning and appreciation for the art of adopting analytical principles that allow you to address your own enterprise requirements in creative and bespoke ways that provide a competitive edge.

About Sense of Security

Sense of Security Pty Limited is an Australian based information security and risk management consulting practice delivering industry leading services and research to organisations throughout Australia and abroad.

Our strategic approach to security provides our clients with a capability to understand the security risks relevant to their organisation and knowledge to protect their information assets.

We provide expertise in governance & compliance, strategy & architecture through to risk assessment, assurance, incident response & security testing.

For more information, please contact us:

Web: www.senseofsecurity.com.au

Email: info@senseofsecurity.com.au

Phone: 1300 922 923