taptu

Putting the Sec in DevSecOps

An approach to writing code that lets you sleep at night

Jakob Pennington
@JakobRPenny

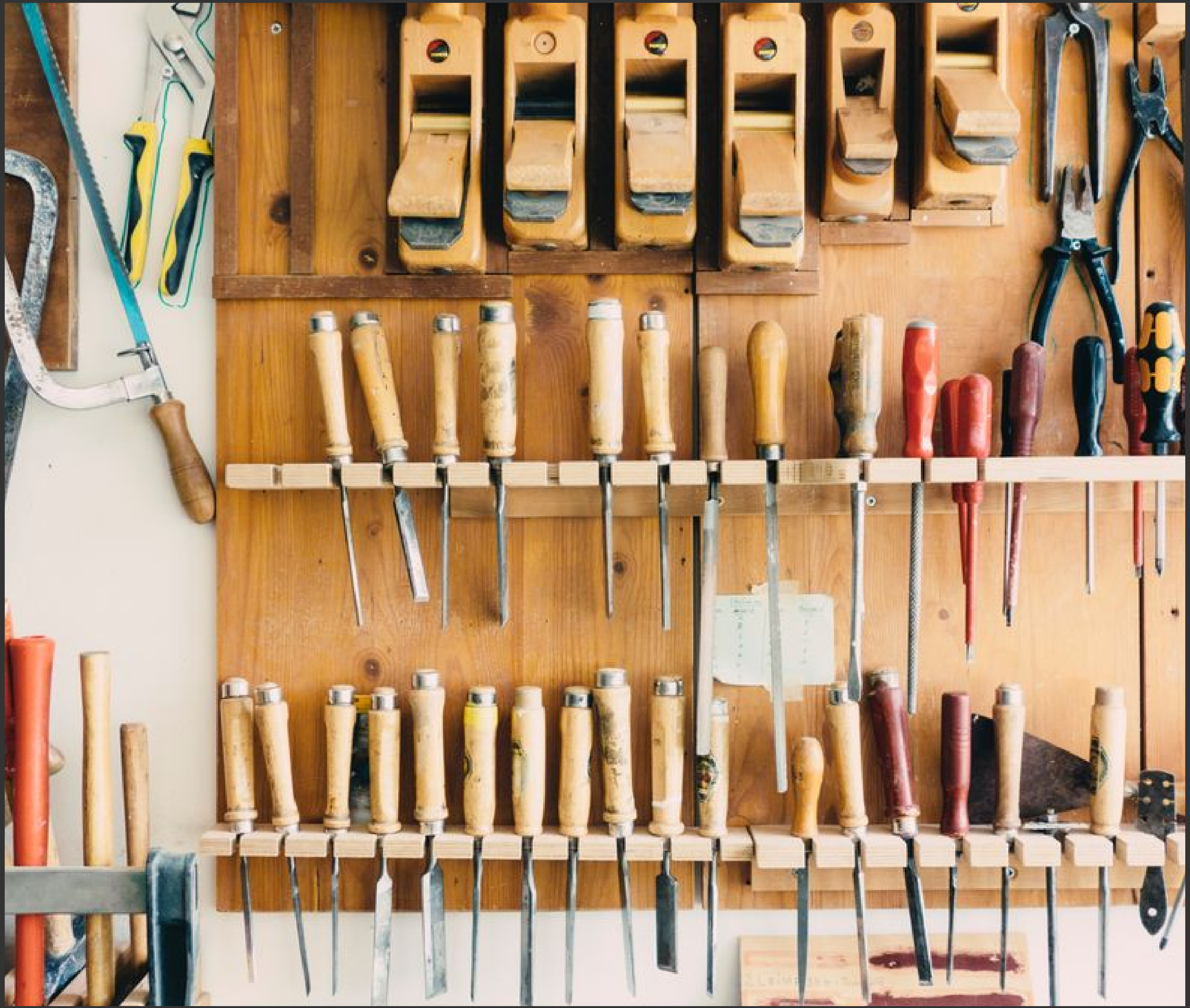
Programme

Background

Why am I speaking to you today

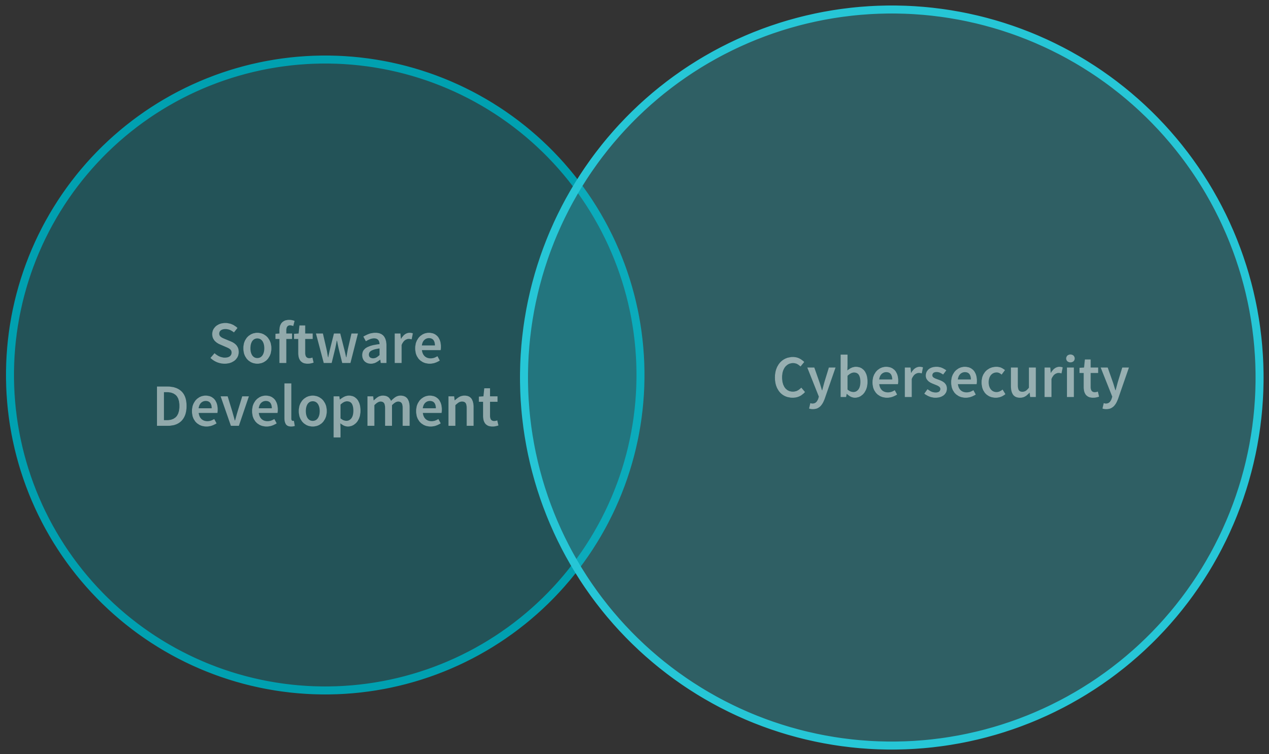
Things I Like to Do

Build Stuff + Break Stuff



Where My Interests Are

Right in that middle bit



Building Security Tools
Secure Software Development

Disclaimers

1. **I am not a .NET developer**

   Sorry... :)

2. **I'm totally biased towards Dev + Sec**

   There's cool operations stuff too!

3. **This is just my opinion**

   I believe building security checks into your development process is a good way to improve software security.

   There is more than one way to skin a cat.

Why all the talk about Cybersecurity?

We just want to build stuff...

Recent Security Breaches

AUSTRALIAN
PARLIAMENT HOUSE

Australian Parliament House

- Network breached
- Nation state actor
- Passwords reset
- IOCs shared with MPs and other Government Departments

Recent Security Breaches

Collection #1

- Largest known collection of breached credentials
- Forwarded to Troy Hunt, added to HIBP
- 2.68 billion records
- 1.16 billion unique username / password pairs
- 773 million unique email addresses
- There are now more records in HIBP than people on earth

Recent Security Breaches

VFEmail

Making email safe for the masses™

“Every VM is lost. Every file server is lost, every backup server is lost. Strangely, not all VMs shared the same authentication, but all were destroyed. This was more than a multi-password via ssh exploit, and there was no ransom. Just attack and destroy.”

Why I don't want to be...

Steve Moore

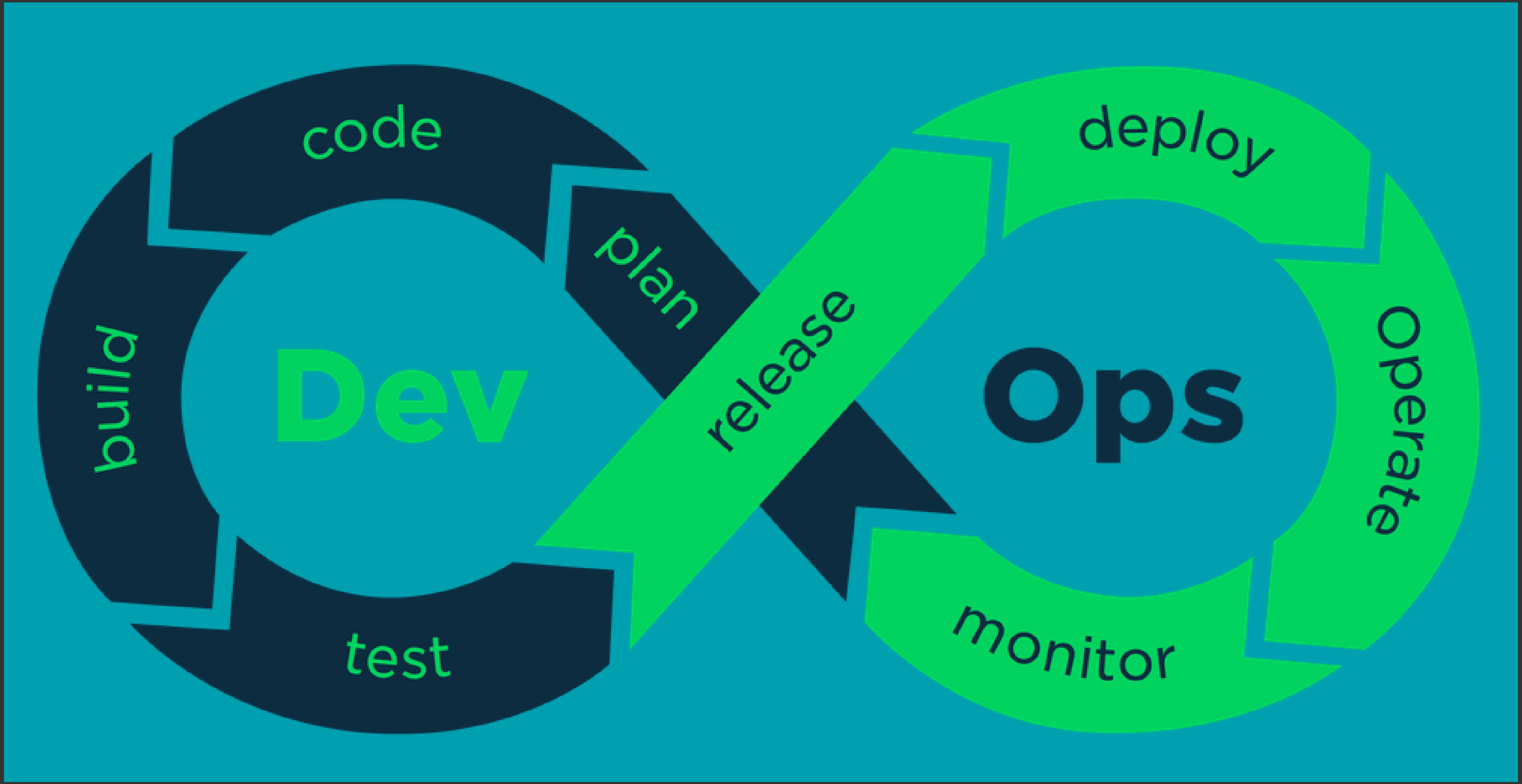
Notifiable Data Breach

DevSecOps

DEVELOPMENT + SECURITY + OPERATIONS

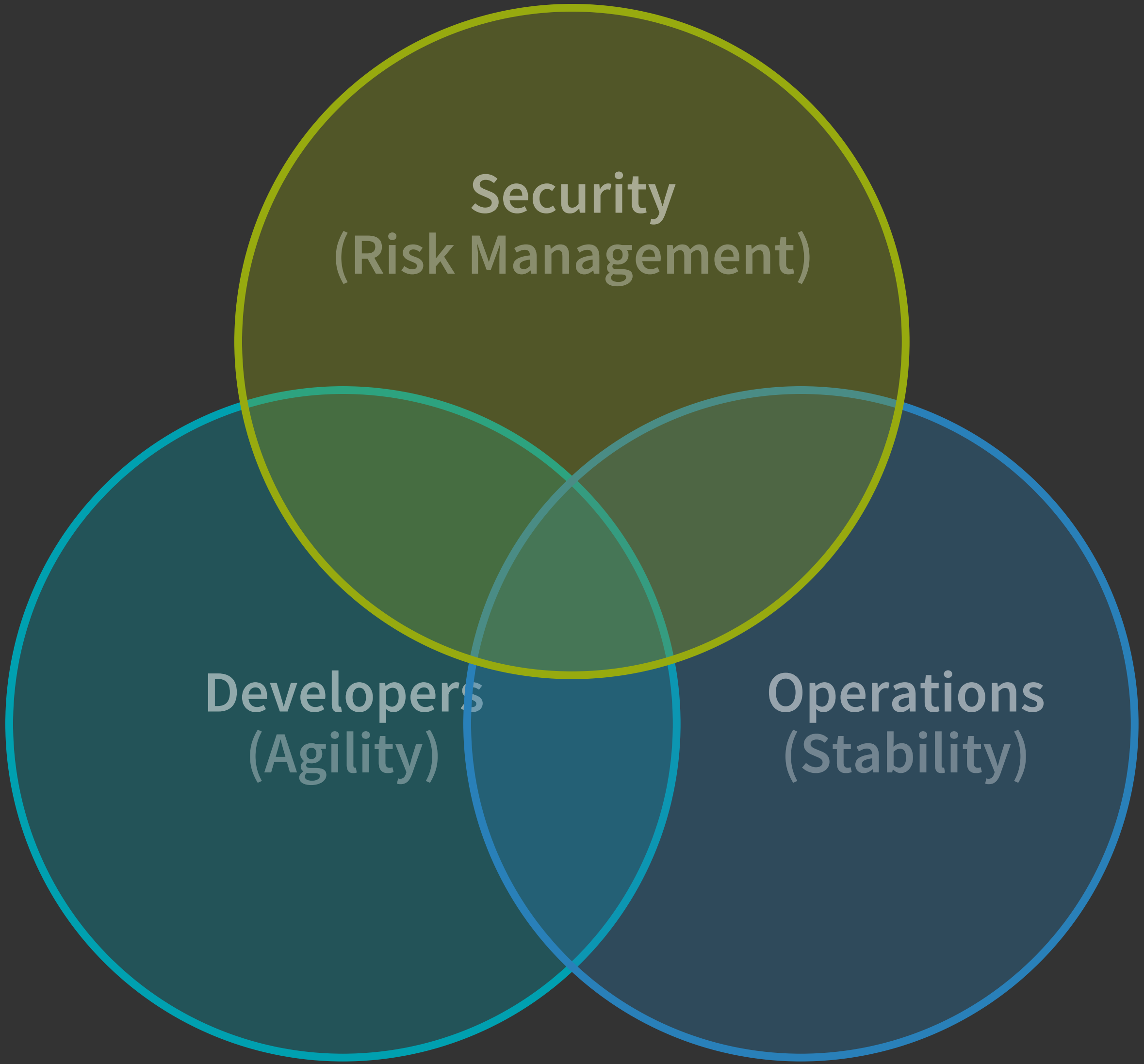
DevOps

Development + Operations



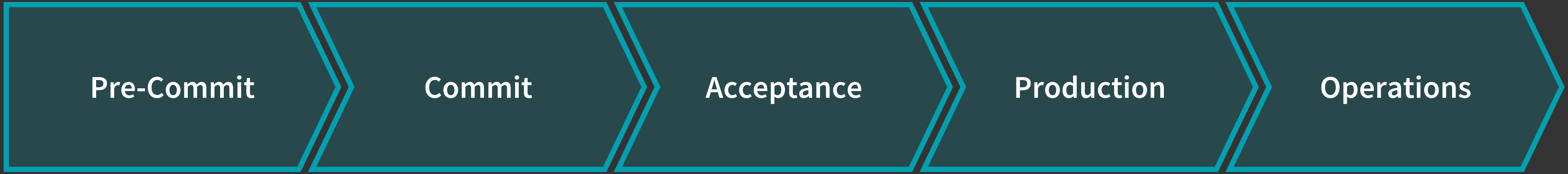
DevSecOps

Development + Security + Operations



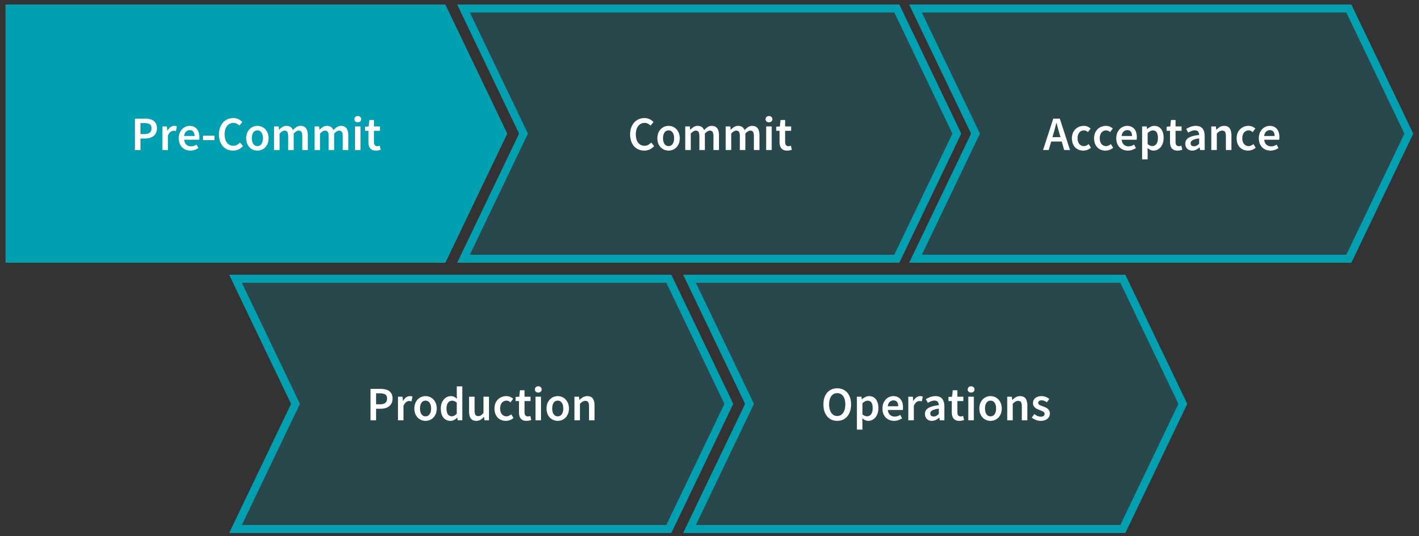
DevSecOps Pipeline

Development + Security + Operations



DevSecOps Pipeline

Development + Security + Operations

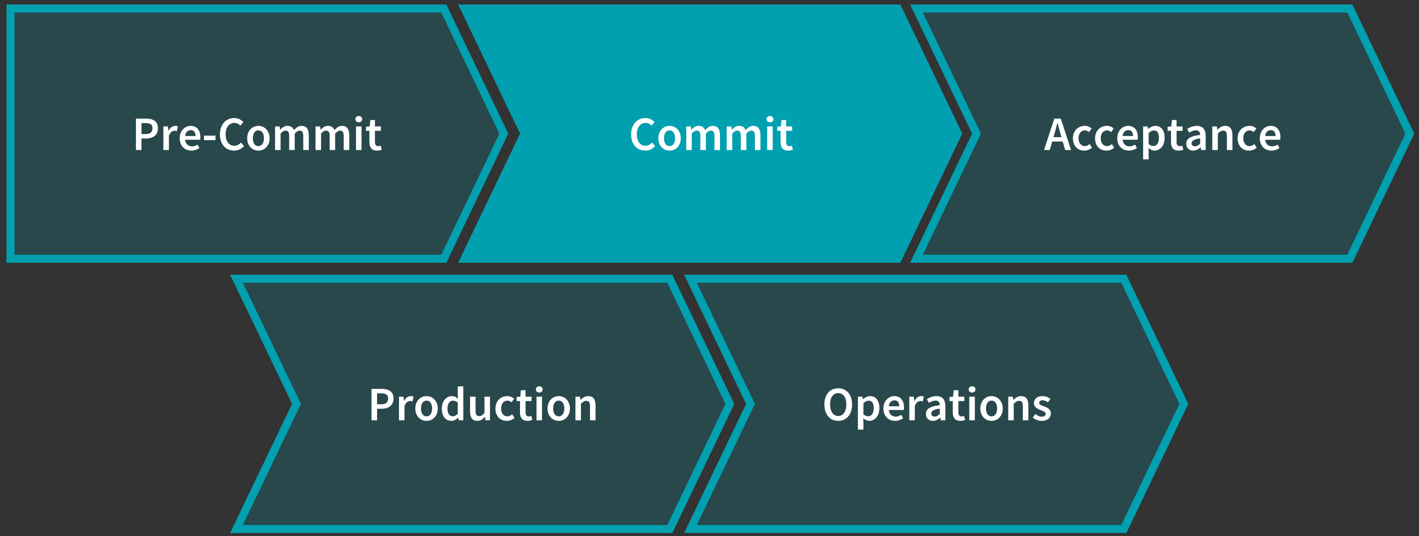


Everything that happens before code is checked into source control

LOCAL ENVIRONMENT

DevSecOps Pipeline

Development + Security + Operations



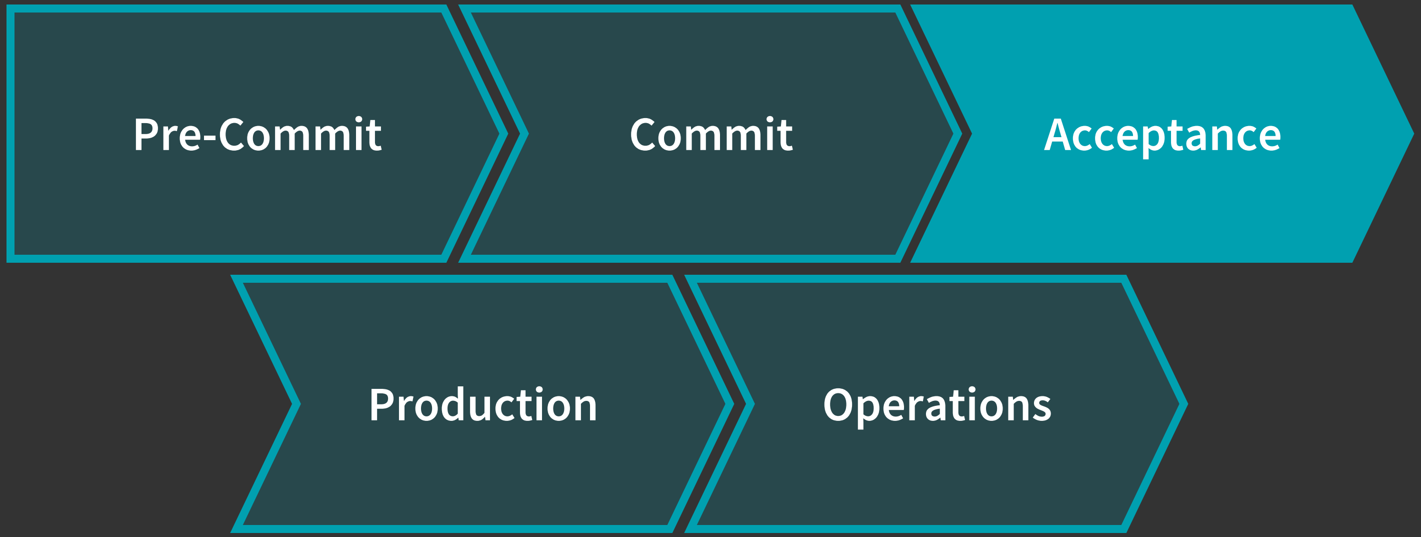
Processes and tests when code is commited and built for Continuous Integration

DEV ENVIRONMENT

(FAST / HIGH CADENCE)

DevSecOps Pipeline

Development + Security + Operations



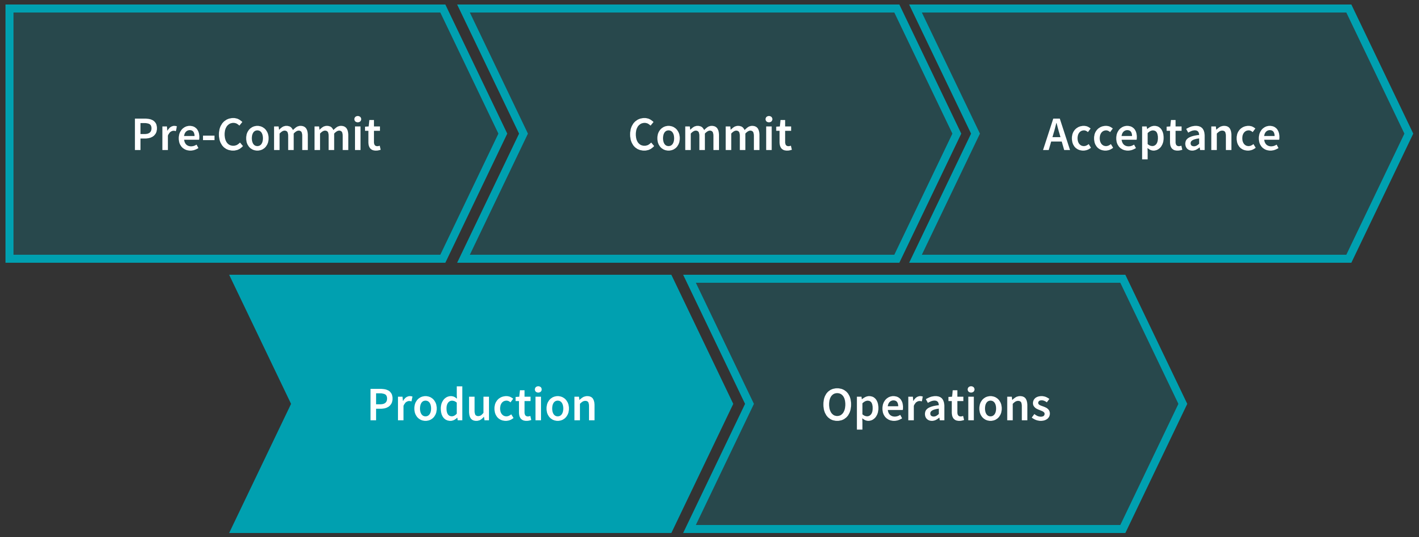
Security and functional testing of release candidates

UAT ENVIRONMENT

(SLOW / LOWER CADENCE)

DevSecOps Pipeline

Development + Security + Operations

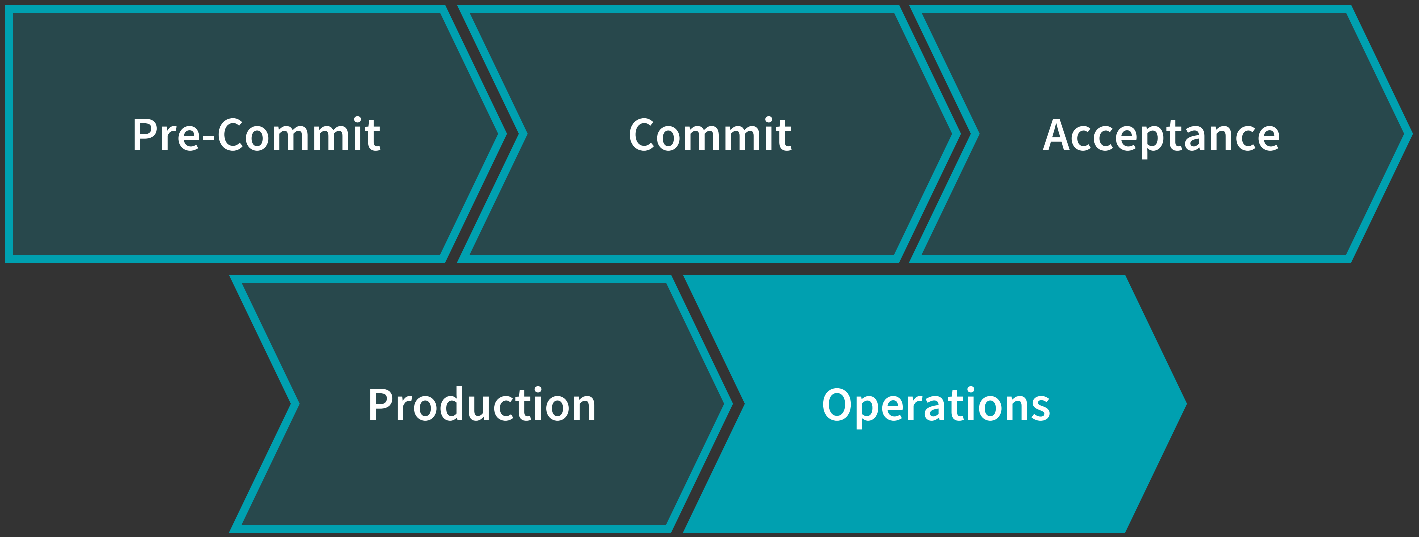


Final checks surrounding the deployment of code into production

PRODUCTION ENVIRONMENT

DevSecOps Pipeline

Development + Security + Operations



Periodic and continuous monitoring and testing for operation, auditing and compliance

PRODUCTION ENVIRONMENT

Building a Pipeline

Core Pipeline

Issue + Project Tracking

Problem

SOFTWARE DEVELOPMENT TEAM NEEDS A MECHANISM FOR TRACKING PROJECT DEVELOPMENT AND ISSUES

Features

- Manage epics, stories and tasks
- Assign tasks to developers
- Maintain a backlog and project timeline
- Feed issues back into development backlog

Azure Boards

Jira

Taiga

Trello

YouTrack

Source Control

Problem

DEVELOPERS NEED A WAY TO COLLABORATE ON THE SAME CODE BASE

Features

- Centralised code storage
- Manage conflicts between multiple commits
- DevSecOps pipeline partially orchestrated by git hooks

AWS CodeCommit

Azure Repos

BitBucket

GitHub

GitLab

Branching Policy

Problem

DIFFICULT FOR EVEN SMALL TEAMS TO COLLABORATE ON A SINGLE BRANCH. REQUIRE GATES BEFORE CODE COMMITED TO MASTER BRANCH.

Features

- Enables development of multiple features in parallel
- Easier to manage releases
- Ensure hot-fixes are applied to all branches
- Prevent tampering of master / core branches

Centralised Workflow

Feature Branch Workflow

GitFlow Workflow

Forking Workflow

Pull Requests

Problem

CODE SHOULD BE REVIEWED FOR CODING STYLE AND CONVENTIONS

Features

- Sanity checks before code is merged with shared code base
- Continuously review in small pieces
- Reviews by security champions for sensitive areas of code
- Prevent poor / negligent / malicious code from joining shared code base

Continuous Integration

Problem

INTEGRATING NEW CODE INTO A SHARED CODE BASE REGULARLY INTRODUCES REGRESSION

Features

- Automatically build code on pull-request into core branches
- Run tests before and after build to highlight regressions
- Fail the pull request if tests fail

AWS CodeBuild

Azure Pipelines

Bamboo

Circle CI

Jenkins

TeamCity

Travis CI

Continuous Delivery / Continuous Deployment

Problem

BUILT CODE NEEDS TO BE DEPLOYED INTO ENVIRONMENTS

Features

- **Deploy build artifacts to environments**
- **Apply necessary configurations**

Continuous Delivery: Manual trigger

Continuous Deployment: Automatic trigger

AWS CodeDeploy

Azure

Buddy

Codeship

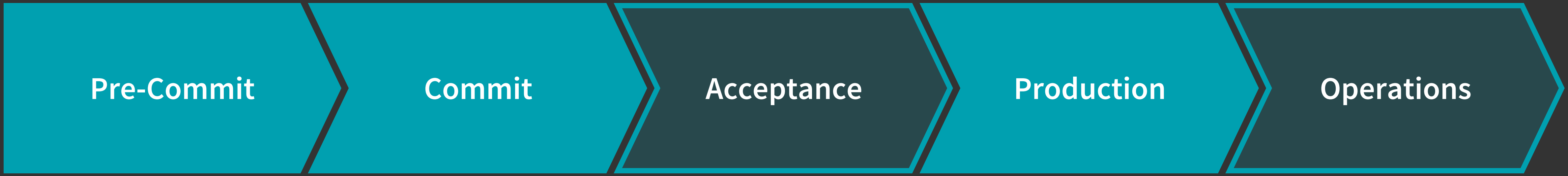
Jenkins

Octopus Deploy

Wercker

DevSecOps Pipeline V1

Development + Security + Operations



Project Tracking

Source Control
Branch Policy
CI
Pull Requests

CD

Building a Pipeline

Automated Scanning + Testing

Dependency Management

Problem

DEPENDENCY PACKAGES MAY CONTAIN VULNERABILITIES

Dependency Management Tools

- Scan for vulnerable packages
- Monitor source control
- Detailed security advisories
- Assist (often automatically) update to patched versions

Snyk

Dependabot

GitHub

IDE Plugins / Static Analysis

Problem

DEVELOPERS ARE UNAWARE WHEN THEY ARE USING INSECURE FUNCTIONS

Features

- Linting tools for security functions
- Appear as spellcheck and compiler warnings
- Can also improve overall code quality
- Educate developers

DevSkim

Puma Scan

Security Code Scan

SonarLint

Veracode

Security Scanning / Dynamic Analysis

Problem

SOME VULNERABILITIES MAY BE DIFFICULT TO DETECT WITH STATIC ANALYSIS

Features

- Use the same scanners that penetration testers do
- Generic crawling and scanning
- Define APIs and do regular fuzzing and specific tests

BurpSuite

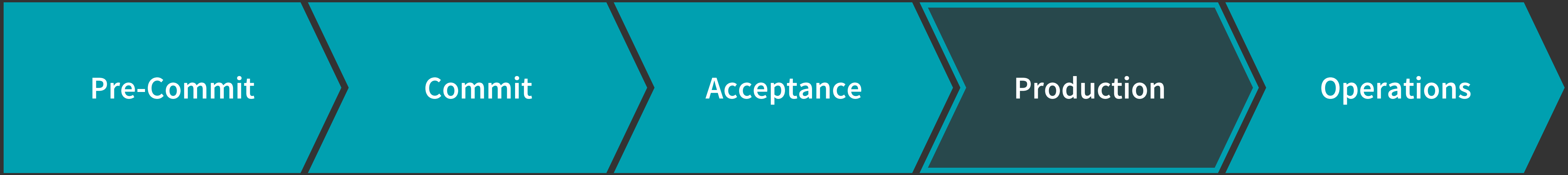
Nessus

nmap

OWASP ZAP

DevSecOps Pipeline V2

Development + Security + Operations



Project Tracking
Dependencies
IDE Plugins

Source Control
Branch Policy
CI
Pull Requests
Static Analysis

Dynamic Analysis

CD

Security Scanning

Building a Pipeline

Deployment Environments

Infrastructure as Code / Infrastructure Tests

Problem

DEPLOYMENT ENVIRONMENTS SHOULD BE PROVISIONED CONSISTENTLY AND SECURELY

Features

- Use code to define how environments are provisioned and configured
- Source controlled
- Align software version with infrastructure version
- Write tests to ensure environments are configured correctly

Ansible

Chef

Puppet

ServerSpec

Terraform

Secret Management

Problem

SECRETS FOR MULTIPLE ENVIRONMENTS NEED TO BE MANAGED SECURELY

Features

- Securely store keys
- Different keys for each environment
- Restrict access to production keys
- Rotate keys at intervals, when employees leave or in the event of a security incident

AWS KMS

Azure Key Vault

Chef Vault

Google Cloud KMS

Hashicorp Vault

Configuration Monitoring

Problem

WANT TO APPLY STRONG SECURITY CONFIGURATIONS IN PRODUCTION ENVIRONMENT

Features

- **Apply best practice security configuration**
- **Prioritise based on risk-rating**
- **Protect access to cloud infrastructure**

AWS Trusted Advisor

Azure Security Centre

CloudCheckr

Dome9

Netflix Security Monkey

Cloud and Application Monitoring

Problem

WE NEED TO MONITOR ACTIVITY IN CLOUD INFRASTRUCTURE AND APPLICATION

Features

- **Monitor activity at an infrastructure and application level**
- **Feed logs and analytics into any existing system**

Azure AppInsights

Azure Monitor

Azure Application Gateway

AWS CloudTrail

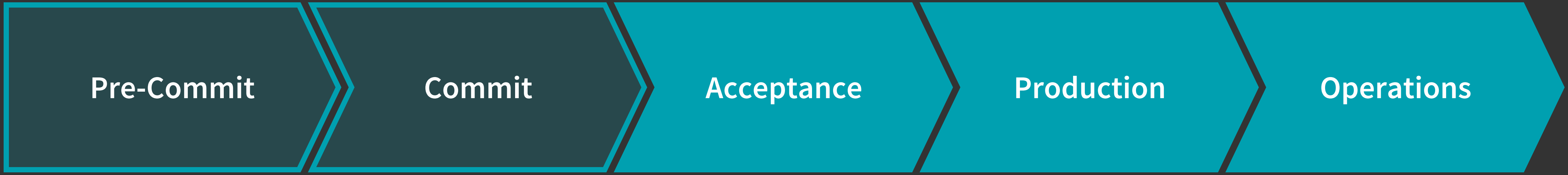
AWS CloudWatch

Backstory

Splunk

DevSecOps Pipeline V3

Development + Security + Operations



Pre-Commit	Commit	Acceptance	Production	Operations
Project Tracking	Source Control	Dynamic Analysis	CD	Security Scanning
Dependencies	Branch Policy	**Infrastructure as Code**	**Secret Management**	**Cloud and Application Monitoring**
IDE Plugins	CI			**Configuration Monitoring**
	Pull Requests			
	Static Analysis			

Building a Pipeline

Manual Security Testing

Third-Party Security Testing

Problem

WE'VE DONE EVERYTHING WE CAN TO BUILD A SECURE APPLICATION, BUT WE WANT A FINAL TEST BY SECURITY EXPERTS

Features

- Confident application has been thoroughly tested
- Third parties have fresh perspective
- Due diligence
- Compliance requirements

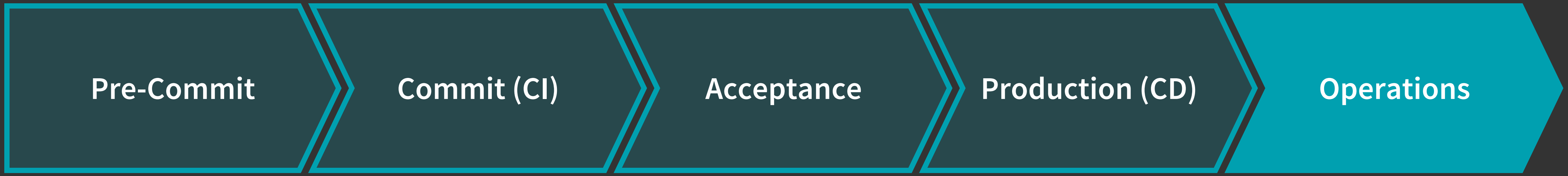
Bug Bounty Program

Code Review

Penetration Testing

DevSecOps Pipeline V4

Development + Security + Operations



Pre-Commit	Commit (CI)	Acceptance	Production (CD)	Operations
Project Tracking	Source Control	Dynamic Analysis	CD	Security Scanning
Dependencies	Branch Policy	Infrastructure as Code	Secret Management	Cloud and Application Monitoring
IDE Plugins	CI			Configuration Monitoring
	Pull Requests			**Third-Party Security Testing**
	Static Analysis			

Build what works for your team

Parting Notes

Tips I've found along the way

Things to Consider

- Your pipeline / development cycle may look entirely different
- Issues feed back as tickets
- DevSecOps more than just tools
- Consider having a pipeline dedicated for testing new tools

- When updating the pipeline, tell your team

  What is the tool / practice?

  What does it do?

  How does it add value?

- When choosing tools:

  Integration into builds > reports

  Ability to accept / postpone issues

Helpful Resources

- The Secure Developer Podcast
- Notifiable Data Breach
- SANS Secure DevOps Toolchain

Secure DevOps Toolchain

Pre-Commit

Security activities before code is checked in to version control

Threat Modeling/Attack Mapping:
- Attacker personas
- Evil user stories
- Raindance
- Mozilla Rapid Risk Assessment
- OWASP ThreatDragon

Security and Privacy Stories:
- OWASP ASVS
- SAFECode Security Stories

IDE Security Plugins:
- DevSkim
- FindSecurityBugs
- Puma Scan
- SonarLint

Pre-Commit Security Hooks:
- git-hound
- git-secrets
- Repo-supervisor
- ThoughtWorks Talisman

Secure Coding Standards:
- CERT Secure Coding Standards
- OWASP Proactive Controls

Manual and Peer Reviews:
- Gerrit
- GitHub pull request
- GitLab merge request
- Review Board

Commit (Continuous Integration)

Fast, automated security checks during the build and Continuous Integration steps

Static Code Analysis (SCA):
- FindSecurityBugs
- Brakeman
- ESLint
- Phan

Security Unit Tests:
- JUnit
- Mocha
- xUnit

Infrastructure as Code Analysis:
- ansible-lint
- Foodcritic
- puppet-lint
- cfn_nag

Dependency Management:
- OWASP Dependency Check
- Bundler-Audit
- Gemnasium
- PHP Security Checker
- Retire.JS
- Node Security Platform

Container Security:
- Actuary
- Anchore
- Clair
- Dagda
- Docker Bench
- Falco

Container Hardening:
- Bane
- CIS Benchmarks
- grsecurity

Acceptance (Continuous Delivery)

Automated security acceptance, functional testing, and deep out-of-band scanning during Continuous Delivery

Infrastructure as Code:
- Ansible
- Chef
- Puppet
- SaltStack
- Terraform
- Vagrant

Immutable Infrastructure:
- Docker
- rkt

Security Scanning:
- Arachni
- nmap
- sqlmap
- sslyze
- ZAP
- ssh_scan

Cloud Configuration Management:
- AWS CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

Security Acceptance Testing:
- BDD-Security
- Gauntlt
- Mittn

Infrastructure Tests:
- Serverspec
- Test Kitchen

Infrastructure Compliance Checks:
- HubbleStack
- InSpec

Production (Continuous Deployment)

Security checks before, during, and after code is deployed to production

Security Smoke Tests:
- ZAP Baseline Scan
- nmap
- ssllabs-scan

Configuration Safety Checks:
- AWS Config
- AWS Trusted Advisor
- Microsoft Azure Advisor
- Security Monkey
- OSQuery

Secrets Management:
- Ansible Vault
- Blackbox
- Chef Vault
- Docker Secrets
- Hashicorp Vault
- Pinterest Knox

Cloud Secrets Management:
- AWS KMS
- Azure Key Vault
- Google Cloud KMS

Cloud Security Testing:
- CloudSploit
- Nimbostratus

Server Hardening:
- dev-sec.io
- SIMP

Host Intrusion Detection System (HIDS):
- fail2ban
- OSSEC
- Samhain

Operations

Continuous security monitoring, testing, audit, and compliance checks

Fault Injection:
- Chaos Kong
- Chaos Monkey

Cyber Simulations:
- Game day exercises
- Tabletop scenarios

Penetration Testing:
- Attack-driven defense
- Bug Bounties
- Red team exercises

Threat Intelligence:
- Diamond Model
- Kill Chain
- STIX
- TAXII

Continuous Scanning:
- OpenSCAP
- OpenVAS
- Prowler
- Scout2
- vuls

Blameless Postmortems:
- Etsy Morgue

Continuous Monitoring:
- grafana
- graphite
- statsd
- seyren
- sof-elk
- ElastAlert
- 411

Cloud Monitoring:
- CloudWatch
- CloudTrail
- Reddalert

Cloud Compliance:
- Cloud Custodian
- Compliance Monkey
- Forseti Security

Building a DevSecOps Program (CALMS)

Culture
Break down barriers between Development, Security, and Operations through education and outreach

Automation
Embed self-service automated security scanning and testing in continuous delivery

Lean
Value stream analysis on security and compliance processes to optimize flow

Measurement
Use metrics to shape design and drive decisions

Sharing
Share threats, risks, and vulnerabilities by adding them to engineering backlogs

Start Your DevOps Metrics Program

- Number of high-severity vulnerabilities and how long they are open
- Build and deployment cycle time
- Automated test frequency and coverage
- Scanning frequency and coverage
- Number of attacks (and attackers) hitting your application

First Steps in Automation

- Build a security smoke test (e.g., ZAP Baseline Scan)
- Conduct negative unit testing to get off of the happy path
- Attack your system before somebody else does (e.g., Gauntlt)
- Add hardening steps into configuration recipes (e.g., dev-sec.io)
- Harden and test your CI/CD pipelines and do not rely on developer-friendly defaults

Learn to build, deliver, and deploy modern applications using secure DevOps and cloud principles, practices, and tools.

DEV540: Secure DevOps and Cloud Application Security

www.sans.org/DEV540

SANS AppSec
APPLICATION & SOFTWARE SECURITY

SANS APPSEC CURRICULUM

PLATFORM SECURITY	CORE	SPECIALIZATION
DEV531 Defending Mobile Applications Security Essentials	STH.DEVELOPER Application Security Awareness Modules	SEC542 Web App Penetration Testing and Ethical Hacking *GWAPT*
DEV541 Secure Coding in Java/JEE *GSSP-JAVA*	DEV522 Defending Web Applications Security Essentials *GWEB*	SEC642 Advanced Web App Penetration Testing, Ethical Hacking, and Exploitation Techniques
DEV544 Secure Coding in .NET *GSSP-NET*	DEV534 Secure DevOps: A Practical Introduction	**ASSESSMENT**
	DEV540 Secure DevOps and Cloud Application Security	AppSec CyberTalent Assessment sans.org/appsec-assessment

Poster contributors:
- Ben Allen
- Jim Bird
- David Deatherage
- Mark Geeslin
- Eric Johnson
- Frank Kim
- Jason Lam
- Gregory Leonard
- Dr. Johannes Ullrich

SANS

SANS AppSec
APPLICATION & SOFTWARE SECURITY

Securing Web Application Technologies (SWAT) CHECKLIST

The SWAT Checklist provides an easy-to-reference set of best practices that raise awareness and help development teams create more secure applications. It's a first step toward building a base of security knowledge around web application security. Use this checklist to identify the minimum standard that is required to neutralize vulnerabilities in your critical applications.

ERROR HANDLING AND LOGGING

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Display generic error messages	Error messages should not reveal details about the internal state of the application. For example, file system path and stack information should not be exposed to the user through error messages.	CWE-209
☐	No unhandled exceptions	Given the languages and frameworks in use for web application development, never allow an unhandled exception to occur. Error handlers should be configured to handle unexpected errors and gracefully return controlled output to the user.	CWE-391
☐	Suppress framework-generated errors	Your development framework or platform may generate default error messages. These should be suppressed or replaced with customized error messages, as framework-generated messages may reveal sensitive information to the user.	CWE-209
☐	Log all authentication and validation activities	Log any authentication and session management activities along with all input validation failures. Any security-related events should be logged. These may be used to detect past or in-progress attacks.	CWE-778
☐	Log all privilege changes	Any activities or occasions where the user's privilege level changes should be logged.	CWE-778
☐	Log administrative activities	Any administrative activities on the application or any of its components should be logged.	CWE-778
☐	Log access to sensitive data	Any access to sensitive data should be logged. This is particularly important for corporations that have to meet regulatory requirements like HIPAA, PCI, or SOX.	CWE-778
☐	Do not log inappropriate data	While logging errors and auditing access are important, sensitive data should never be logged in an unencrypted form. For example, under HIPAA and PCI, it would be a violation to log sensitive data into the log itself unless the log is encrypted on the disk. Additionally, it can create a serious exposure point should the web application itself become compromised.	CWE-532
☐	Store logs securely	Logs should be stored and maintained appropriately to avoid information loss or tampering by intruders. Log retention should also follow the retention policy set forth by the organization to meet regulatory requirements and provide enough information for forensic and incident response activities.	CWE-533

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH

SANS

Security Roadmap
P O S T E R

Securing Web Application Technologies (SWAT)
C H E C K L I S T
Version 1.5

AND

Secure DevOps Toolchain

Ingraining security into the mind of every developer.

software-security.sans.org

DATA PROTECTION

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Use HTTPS everywhere	Ideally, HTTPS should be used for your entire application. If you have to limit where it's used, then HTTPS must be applied to any authentication pages as well as to all pages after the user is authenticated. If sensitive information (e.g., personal information) can be submitted before authentication, those features must also be sent over HTTPS. Always link to the HTTPS version of URL if available. Relying on redirection from HTTP to HTTPS increases the opportunity for an attacker to insert a man-in-the-middle attack without raising the user's suspicion. **EXAMPLE:** sslstrip	CWE-311 CWE-319 CWE-523
☐	Disable HTTP access for all protected resources	For all pages requiring protection by HTTPS, the same URL should not be accessible via the insecure HTTP channel.	CWE-319
☐	Use the Strict-Transport-Security header	The Strict-Transport-Security header ensures that the browser does not talk to the server over HTTP. This helps reduce the risk of HTTP downgrade attacks as implemented by the sslsniff tool.	
☐	Store user passwords using a strong, iterative, salted hash	User passwords must be stored using secure hashing techniques with strong algorithms like PBKDF2, bcrypt, or SHA-512. Simply hashing the password a single time does not sufficiently protect the password. Use adaptive hashing (a work factor), combined with a randomly generated salt for each user to make the hash strong. **EXAMPLE:** LinkedIn password leak	CWE-257
☐	Securely exchange encryption keys	If encryption keys are exchanged or pre-set in your application, then any key establishment or exchange must be performed over a secure channel.	
☐	Set up secure key management processes	When keys are stored in your system they must be properly secured and only accessible to the appropriate staff on a need-to-know basis. **EXAMPLE:** AWS Key Management Service (KMS), Azure Key Vault, AWS CloudHSM	CWE-320
☐	Use strong TLS configurations	Weak ciphers must be disabled on all servers. For example, SSL v2, SSL v3, and TLS protocols prior to 1.2 have known weaknesses and are not considered secure. Additionally, disable the NULL, RC4, DES, and MD5 cipher suites. Ensure all key lengths are greater than 128 bits, use secure renegotiation, and disable compression. **EXAMPLE:** Qualys SSL Labs	
☐	Use valid HTTPS certificates from a reputable certificate authority	HTTPS certificates should be signed by a reputable certificate authority. The name on the certificate should match the FQDN of the website. The certificate itself should be valid and not expired. **EXAMPLE:** Let's Encrypt https://letsencrypt.org	
☐	Disable data caching using cache control headers and autocomplete	Browser data caching should be disabled using the cache control HTTP headers or meta tags within the HTML page. Additionally, sensitive input fields, such as the login form, should have the autocomplete attribute set to off in the HTML form to instruct the browser not to cache the credentials.	CWE-524
☐	Encrypt sensitive data at rest	Encrypt sensitive or critical data before storage.	CWE-311 CWE-312
☐	Limit the use and storage of sensitive data	Conduct an evaluation to ensure that sensitive data elements are not being unnecessarily transported or stored. Where possible, use tokenization to reduce data exposure risks.	

CONFIGURATION AND OPERATIONS

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Automate application deployment	Automating the deployment of your application, using Continuous Integration and Continuous Deployment, helps to ensure that changes are made in a consistent, repeatable manner in all environments.	
☐	Establish a rigorous change management process	A rigorous change management process must be maintained during operations. For example, new releases should only be deployed after proper testing and associated documentation has been completed. **EXAMPLE:** DevOps Audit Defense Toolkit https://itrevolution.com/devops-audit-defense-toolkit	CWE-439
☐	Define security requirements	Engage the business owner to define security requirements for the application. This includes items that range from the whitelist validation rules all the way to nonfunctional requirements like the performance of the login function. Defining these requirements up front ensures that security is baked into the system.	
☐	Conduct a design review	Integrating security into the design phase saves money and time. Conduct a risk review with security professionals and threat model the application to identify key risks. This helps you integrate appropriate countermeasures into the design and architecture of the application.	CWE-701 CWE-656
☐	Perform code reviews	Security-focused code reviews can be one of the most effective ways to find security bugs. Regularly review your code looking for common issues like SQL Injection and Cross-Site Scripting. Leverage automated tools to maximize breadth of coverage and consistency.	CWE-702
☐	Perform security testing	Conduct security testing both during and after development to ensure the application meets security standards. Testing should also be conducted after major releases to ensure vulnerabilities did not get introduced during the update process. Leverage automation by including security tests into the CI/CD pipeline.	
☐	Harden the infrastructure	All components of infrastructure that support the application should be configured according to security best practices and hardening guidelines. In a typical web application this can include routers, firewalls, network switches, operating systems, web servers, application servers, databases, and application frameworks.	CWE-15 CWE-656
☐	Define an incident handling plan	An incident handling plan should be drafted and tested on a regular basis. The contact list of people to involve in a security incident related to the application should be well defined and kept up to date.	
☐	Educate the team on security	Training helps define a common language that the team can use to improve the security of the application. Education should not be confined solely to software developers, testers, and architects. Anyone associated with the development process, such as business analysts and project managers, should all have periodic software security awareness training.	

AUTHENTICATION

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Don't hardcode credentials	Never allow credentials to be stored directly within the application code. While it can be convenient to test application code with hardcoded credentials during development, this significantly increases risk and should be avoided. **EXAMPLE:** Hard-coded passwords in networking devices https://www.us-cert.gov/control_systems/pdf/ICSA-12-243-01.pdf	CWE-798
☐	Develop a strong password reset system	Password reset systems are often the weakest link in an application. These systems are often based on users answering personal questions to establish their identity and in turn reset the password. The system needs to be based on questions that are both hard to guess and brute force. Additionally, any password reset option must not reveal whether or not an account is valid, preventing username harvesting. **EXAMPLE:** Sara Palin password hack http://en.wikipedia.org/wiki/Sarah_Palin_email_hack	CWE-640
☐	Implement a strong password policy	A password policy should be created and implemented so that passwords meet specific strength criteria. **EXAMPLE:** https://pages.nist.gov/800-63-3/sp800-63-3.html	CWE-521
☐	Implement account lockout against brute-force attacks	Account lockout needs to be implemented to prevent brute-force attacks against both the authentication and password reset functionality. After several tries on a specific user account, the account should be locked for a period of time or until it is manually unlocked. Additionally, it is best to continue the same failure message indicating that the credentials are incorrect or the account is locked to prevent an attacker from harvesting usernames.	CWE-307
☐	Don't disclose too much information in error messages	Messages for authentication errors must be clear and, at the same time, be written so that sensitive information about the system is not disclosed. For example, error messages that reveal that the user id is valid but that the corresponding password is incorrect confirm to an attacker that the account does exist on the system.	
☐	Store database credentials securely	Modern web applications usually consist of multiple layers. The business logic tier (processing of information) often connects to the data tier (database). Connecting to the database, of course, requires authentication. The authentication credentials in the business logic tier must be stored in a centralized location that is locked down. Scattering credentials throughout the source code is not acceptable. Some development frameworks provide a centralized secure location for storing credentials to the backend database. These encrypted stores should be leveraged when possible.	CWE-257
☐	Applications and middleware should run with minimal privileges	If an application becomes compromised it is important that the application itself and any middleware services be configured to run with minimal privileges. For instance, while the application layer or business layer need the ability to read and write data to the underlying database, administrative credentials that grant access to other databases or tables should not be provided.	CWE-250

SESSION MANAGEMENT

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Ensure that session identifiers are sufficiently random	Session tokens must be generated by secure random functions and must be of sufficient length to withstand analysis and prediction.	CWE-6
☐	Regenerate session tokens	Session tokens should be regenerated when the user authenticates to the application and when the user privilege level changes. Additionally, should the encryption status change, the session token should always be regenerated.	CWE-384
☐	Implement an idle session timeout	When a user is not active, the application should automatically log the user out. Be aware that Ajax applications may make recurring calls to the application, effectively resetting the timeout counter automatically.	CWE-613
☐	Implement an absolute session timeout	Users should be logged out after an extensive amount of time (e.g., 4-8 hours) has passed since they logged in. This helps mitigate the risk of an attacker using a hijacked session.	CWE-613
☐	Destroy sessions at any sign of tampering	Unless the application requires multiple simultaneous sessions for a single user, implement features to detect session cloning attempts. Should any sign of session cloning be detected, the session should be destroyed, forcing the real user to reauthenticate.	
☐	Invalidate the session after logout	When the user logs out of the application, the session and corresponding data on the server must be destroyed. This ensures that the session cannot be accidentally revived.	CWE-613
☐	Place a logout button on every page	The logout button or logout link should be easily accessible to users on every page after they have authenticated.	
☐	Use secure cookie attributes (i.e., HttpOnly and Secure flags)	The session cookie should be set with both the HttpOnly and the Secure flags. This ensures that the session id will not be accessible to client-side scripts and will only be transmitted over HTTPS.	CWE-79 CWE-614
☐	Set the cookie domain and path correctly	The cookie domain and path scope should be set to the most restrictive settings for your application. Any wildcard domain scoped cookie must have a good justification for its existence.	
☐	Set the cookie expiration time	The session cookie should have a reasonable expiration time. Non-expiring session cookies should be avoided.	

Website
software-security.sans.org
Free resources, white papers, webcasts, and more

Blog
software-security.sans.org/blog

Twitter
@sansappsec
Latest news, promos, and other information

AppSec CyberTalent Assessment
sans.org/appsec-assessment

INPUT AND OUTPUT HANDLING

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Conduct contextual output encoding	All output functions must contextually encode data before sending the data to the user. Depending on where the output will end up in the HTML page, the output must be encoded differently. For example, data placed in the URL context must be encoded differently than data placed in a JavaScript context within the HTML page. **EXAMPLE:** Resource: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet	CWE-79
☐	Prefer whitelists over blacklists	For each user input field, there should be validation on the input content. Whitelisting input is the preferred approach. Only accept data that meet a certain criteria. For input that needs more flexibility, blacklisting can also be applied where known bad input patterns or characters are blocked.	CWE-159 CWE-144
☐	Use parameterized SQL queries	SQL queries should be crafted with user content passed into a bind variable. Queries written this way are safe against SQL injection attacks. SQL queries should not be created dynamically using string concatenation. Similarly, the SQL query string used in a bound or parameterized query should never be dynamically built from user input. **EXAMPLE:** Sony SQL injection Hack http://www.infosecurity-magazine.com/view/27930/lulzsec-sony-pictures-hackers-were-school-chums	CWE-89 CWE-564
☐	Set the encoding for your application	For every page in your application, set the encoding using HTTP headers or meta tags within HTML. This ensures that the encoding of the page is always defined and that the browser will not have to determine the encoding on its own. Setting a consistent encoding like UTF-8 for your application reduces the overall risk of issues like Cross-Site Scripting.	CWE-172
☐	Validate uploaded files	When accepting file uploads from the user make sure to validate the size of the file, the file type, and the file contents, and ensure that it is not possible to override the destination path for the file.	CWE-434 CWE-616 CWE-22
☐	Use the nosniff header for uploaded content	When hosting user uploaded content that can be viewed by other users, use the X-Content-Type-Options: nosniff header so that browsers do not try to guess the data type. Sometimes the browser can be tricked into displaying the data type incorrectly (e.g., showing a GIF file as HTML). Always let the server or application determine the data type.	CWE-430
☐	Prevent tabnabbing	When including a link to a page on a different site that opens in a new tab (such as by using target="_blank"), include rel="noopener noreferrer" to prevent the linked page from changing the opener's tab (such as to a look-a-like phishing site).	
☐	Validate the source of input	The source of the input must be validated. For example, if input is expected from a POST request, do not accept the input variable from a GET request.	CWE-20 CWE-346
☐	X-Frame-Options or CSP headers	Use the X-Frame-Options header or Content-Security-Policy (CSP) header frame-ancestors directive to prevent content from being loaded by a foreign site in a frame. This mitigates Clickjacking attacks. For older browsers that do not support this header add framebusting Javascript code to mitigate Clickjacking (although this method is not foolproof and can be circumvented).	CAPEC-103 CWE-693
☐	Use secure HTTP response headers	The Content Security Policy (CSP), X-XSS-Protection, and Public-Key-Pins headers help defend against Cross-Site Scripting (XSS) and Man-In-the-Middle (MITM) attacks. **EXAMPLE:** OWASP Secure Headers Project https://www.owasp.org/index.php/OWASP_Secure_Headers_Project	CWE-79 CWE-692

ACCESS CONTROL

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Apply access control checks consistently	Always apply the principle of complete mediation, forcing all requests through a common security "gate keeper." This ensures that access control checks are triggered whether or not the user is authenticated.	CWE-284
☐	Apply the principle of least privilege	Use a Mandatory Access Control system. All access decisions will be based on the principle of least privilege. If not explicitly allowed then access should be denied. Additionally, after an account is created, rights must be specifically added to that account to grant access to resources.	CWE-272 CWE-250
☐	Don't use direct object references for access control checks	Do not allow direct references to files or parameters that can be manipulated to grant excessive access. Access control decisions must be based on the authenticated user identity and trusted server-side information.	CWE-284
☐	Don't use unvalidated forwards or redirects	An unvalidated forward can allow an attacker to access private content without authentication. Unvalidated redirects allow an attacker to lure victims into visiting malicious sites. Prevent this from occurring by conducting the appropriate access control checks before sending the user to the given location.	CWE-601

SANS APPSEC CURRICULUM

PLATFORM SECURITY
- DEV531 Defending Mobile Applications Security Essentials
- DEV541 Secure Coding in Java/JEE *GSSP-JAVA*
- DEV544 Secure Coding in .NET *GSSP-.NET*

CORE
- STH.DEVELOPER Application Security Awareness Modules
- DEV522 Defending Web Applications Security Essentials *GWEB*
- DEV534 Secure DevOps: A Practical Introduction
- DEV540 Secure DevOps and Cloud Application Security

SPECIALIZATION
- SEC542 Web App Penetration Testing and Ethical Hacking *GWAPT*
- SEC642 Advanced Web App Penetration Testing, Ethical Hacking, and Exploitation Techniques

ASSESSMENT
- AppSec CyberTalent Assessment sans.org/appsec-assessment

Thanks!

Jakob Pennington

Cybersecurity / Software Specialist
Taptu IT Advisory (@TaptuIT)

jakob.pennington@taptu.com.au

linkedin.com/in/jakobpennington

medium.com/@jakob.pennington

@JakobRPenny