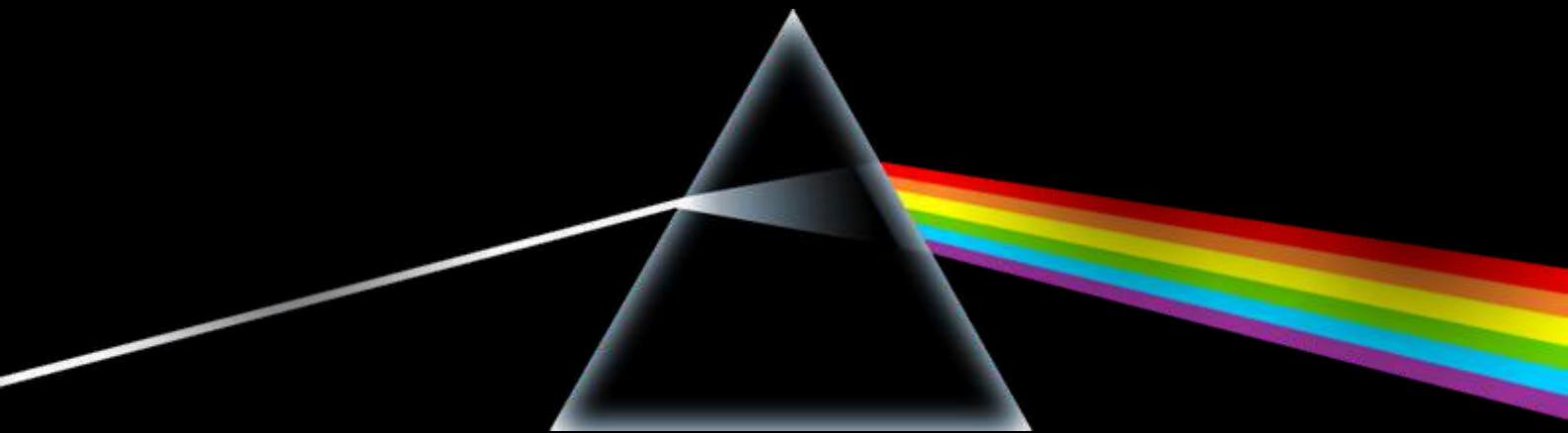


2023

# DEVSECOPS MATURITY MODEL

A roadmap for organizations to evolve their practices.



WWW.DEVSECOPSGUIDES.COM

# INTRODUCTION

The Evolving World of Software Development In the dynamic realm of software development, integrating security into the development process has become a paramount concern. This introductory section sets the stage for the journey through the DevSecOps Maturity Model, a framework that blends development, security, and operations into a cohesive and efficient workflow. The paragraph emphasizes the changing landscape of software development and the rising need for security to be a foundational element rather than an afterthought.

Understanding DevSecOps DevSecOps represents a cultural and technical shift in how organizations approach software development and security. This section defines DevSecOps, detailing how it extends beyond traditional development practices by embedding security at every stage of the software development life cycle. It underscores the philosophy of 'security as code' and its significance in modern software development.

The Need for a Maturity Model As organizations adopt DevSecOps practices, the need for a structured approach to gauge and enhance their progress becomes clear. This paragraph introduces the concept of a maturity model in the context of DevSecOps, explaining its role in helping organizations systematically improve their integration of security into development processes.

Overview of the DevSecOps Maturity Model The DevSecOps Maturity Model provides a roadmap for organizations to evolve their practices. This section offers an overview of the model, outlining its various stages from initial implementation to advanced stages where DevSecOps principles are deeply ingrained in the organizational culture and workflows.

Key Components of the Maturity Model Delving into the specifics, this part enumerates the key components of the DevSecOps Maturity Model, such as automation, policy as code, security integration, monitoring, and feedback loops. It briefly describes each component's role in enhancing the security and efficiency of the development process.

Challenges and Strategies Implementing DevSecOps is not without its challenges. This section discusses common obstacles organizations face, such as cultural resistance, tool integration issues, and skill gaps. It also suggests strategies to overcome these challenges, emphasizing the importance of leadership, training, and clear communication.

Case Studies and Real-World Examples To illustrate the practical application of the DevSecOps Maturity Model, this paragraph introduces a series of case studies and real-world examples. These narratives showcase how different organizations have successfully navigated their DevSecOps journey, providing insights and lessons learned.

Navigating Forward Concluding the introduction, this section outlines the book's structure, guiding the reader through the detailed exploration of each stage of the DevSecOps Maturity Model. It sets expectations for the comprehensive coverage of best practices, tools, and methodologies that will be discussed in subsequent chapters.



# HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

# ACKNOWLEDGMENT

- **Dan Williams**
- **Tanya Janca**
- **Aristide Bouix**
- **Okan Yıldız**
- **Carol Valencia**
- **Ahmed AlaaEldin**
- **Alessio dl**
- **Ayoub Najim**
- **Barnavo Chowdhury**
- **Carol Valencia**
- **Charles Chibueze**
- **David das Neves**
- **Devyani Vij**
- **Eslam Samy Hosney**
- **Ibrahim Akkulak**
- **Josh**
- **Mahesh M**
- **Mohammad Hadi Arghani**
- **Muthukumar Selvarasu**
- **Nditah Samweld**
- **Ahmed Saafan**
- **SANKALP SANDEEP PARANJPE**
- **Williams A. Danso**
- **David das Neves**

# DEVSECOPS LAYERS IN SECURITY CONTROLS, TOOLS, AND PRACTICES THROUGHOUT THE DEVOPS LIFECYCLE

DevSecOps represents the integration of security practices within the DevOps framework. It emphasizes the need for a collaborative approach between development, operations, and security teams. This integration is crucial for accelerating delivery, enhancing software quality, and ensuring reliable services in production. The DevSecOps Maturity Model outlines four stages of maturity across six major competency areas.

## The Stages of DevSecOps Maturity

### Beginner

- Focus: Cultural shift emphasizing collaboration across technical disciplines.
- Characteristics:
  - Beginning of the DevSecOps journey.
  - Emphasis on team performance improvement.
  - Initial integration of security practices.

### Intermediate

- Focus: Consistent software release with an evolving security integration.
- Characteristics:
  - Regular software releases with some bottlenecks and performance issues.
  - Security controls start shifting earlier in the development process.
  - Some friction among teams, with security work still concentrated towards the end of the process.

### Advanced

- Focus: Efficient and productive release of high-quality, secure software.
- Characteristics:
  - High efficiency in software development and release.
  - Security checkpoints embedded throughout the software development lifecycle (SDLC).
  - Regular release of secure and reliable software.

### Expert

- Focus: Cutting-edge practices with deep integration of security across the SDLC.
- Characteristics:
  - Frequent release of high-quality code, potentially multiple times per day.
  - Deeply embedded security controls throughout the SDLC.
  - High level of automation across Development, Operations, and Security.
  - Security is no longer a siloed domain but an integral part of the entire process.

## Key Aspects of DevSecOps Integration

- Collaboration and Sharing: Breaking down silos between security and DevOps teams.
- Shift-Left Security: Integrating security early in the development process.
- Continuous Feedback and Improvement: Leveraging feedback loops for continuous enhancement of security practices.
- Automation: Automating security tasks to increase efficiency and reduce human error.
- Regular and Reliable Releases: Ensuring that each release is secure and stable.
- Holistic Security Approach: Considering security at every stage of the SDLC.

DevSecOps is not just a set of practices but a cultural shift that requires commitment from all stakeholders involved in the software development and deployment process. The journey from Beginner to Expert stage in DevSecOps maturity involves a gradual and consistent integration of security practices, tools, and mindsets into the existing DevOps framework.

# Competency

## Competency: People & Culture

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- Functional teams siloed</li> <li>- High inter-team friction</li> <li>- Nascent onboarding processes</li> <li>- Burnout common</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Silos breaking down</li> <li>- Embracing experimentation &amp; transparency</li> <li>- Onboarding process exists</li> <li>- Burnout openly discussed</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- Continuous collaboration across teams</li> <li>- Blameless culture</li> <li>- Comprehensive onboarding process</li> <li>- Burnout quickly addressed</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- Cross-functional teams aligned to products and services</li> <li>- High trust, experimentation, learning culture</li> <li>- Burnout rare</li> </ul>

## Competency: Plan & Develop

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- Risk and security not considered</li> <li>- High technical debt</li> <li>- Excessive bug fix work</li> <li>- Code not validated</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Limited risk assessment</li> <li>- Moderate technical debt</li> <li>- Moderate bug fix work</li> <li>- Some code validation</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- Threat modeling and risk assessments</li> <li>- Low technical debt</li> <li>- Low bug fix work</li> <li>- All code validated</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- Extensive threat modeling/risk assessment</li> <li>- Minimal technical debt</li> <li>- New feature focus</li> <li>- All code validated automatically</li> </ul>

## Competency: Build & Test

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- Manual testing</li> <li>- No code scanning</li> <li>- No build/signature validation</li> <li>- Limited core functionality testing</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Partial test automation</li> <li>- Partial code scanning</li> <li>- Partial build/signature validation</li> <li>- Partial core functionality testing</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- High test automation</li> <li>- Dynamic code scanning</li> <li>- Significant build/signature validation</li> <li>- Significant core functionality testing</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- Complete test automation</li> <li>- Comprehensive dynamic code scanning</li> <li>- Comprehensive build/signature validation</li> <li>- Comprehensive core functionality testing</li> </ul>

## Competency: Release & Deploy

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- Manual deployments</li> <li>- Large, infrequent releases</li> <li>- No deployment security posture criteria</li> <li>- Difficult to remediate failed deployment</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Partial deployment automation</li> <li>- Medium-sized, monthly releases</li> <li>- Basic deployment security posture criteria</li> <li>- Acceptable failed deployment remediation times</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- High deployment automation</li> <li>- Small, weekly releases</li> <li>- Detailed deployment security posture criteria</li> <li>- Fast failed deployment remediation times</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- Full deployment automation</li> <li>- Numerous daily releases</li> <li>- Automated deployment failing</li> <li>- Bias to fast forward fixes</li> </ul>

## Competency: Operate

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- Manual provisioning/configuration</li> <li>- Long capacity planning cycles</li> <li>- Manual scaling</li> <li>- Single availability zone</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Partial configuration/provisioning automation</li> <li>- OpEx-based capacity planning</li> <li>- Partial autoscaling</li> <li>- Multi-availability zone/region</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- Extensive configuration/provisioning automation</li> <li>- Capacity planning based on seasonality/growth</li> <li>- Significant autoscaling</li> <li>- Multiple cloud providers / high availability</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- All infrastructure configurations and instructions instantiated as code</li> <li>- Capacity planning based on granular usage trends/predictions</li> <li>- Comprehensive auto-scaling</li> <li>- Multiple cloud providers / very high availability</li> </ul>

## Competency: Observe & Respond

Level	Description
Beginner	<ul style="list-style-type: none"> <li>- No chaos testing or red teaming</li> <li>- Poor patching hygiene</li> <li>- No disaster recovery strategy</li> <li>- No SLOs formed</li> </ul>
Intermediate	<ul style="list-style-type: none"> <li>- Basic chaos testing or red teaming</li> <li>- Basic patching hygiene</li> <li>- Basic DR strategy</li> <li>- Basic SLOs formed</li> </ul>
Advanced	<ul style="list-style-type: none"> <li>- Significant chaos testing &amp; red teaming</li> <li>- Fast patching</li> <li>- Comprehensive DR strategy</li> <li>- SLOs &amp; error budgets favored</li> </ul>
Expert	<ul style="list-style-type: none"> <li>- Continuous chaos testing &amp; red teaming</li> <li>- Patching SLA</li> <li>- DR plans tested often</li> <li>- SLOs &amp; error budgets drive decisions</li> </ul>

## DevSecOps Maturity Model Measurement

### Step 1: Assess Where Your Organization Is

## Maturity by Competency

Competency	Beginner	Intermediate	Advanced	Expert
Culture				
Plan & Develop				
Build & Test				
Operate				
Release & Deploy				
Observe & Respond				

## Step 2: Define Where Your Organization Needs to Go

### Target Maturity by Competency

Competency	Current Level	Target Level (12-18 months)
Culture		
Plan & Develop		
Build & Test		
Operate		
Release & Deploy		
Observe & Respond		

## Step 3: Determine How You'll Get There by Prioritizing Initiatives

### Initiatives and Ownership

Quarter	Competency	Initiative Owner
Q1	Culture	Q1 Owner
Q1	Plan & Develop	Q1 Owner
Q1	Build & Test	Q1 Owner
Q2	Operate	Q2 Owner/Leader
Q2	Release & Deploy	Q2 Owner
Q2	Observe & Respond	Q2 Owner
Q3	[Additional Initiatives]	Q3 Owner

### Overall Maturity Progress

Current Level	Target Level	How We're Getting There
Beginner	Intermediate/Advanced/Expert	[Specific Steps and Initiatives]

## DevSecOps Metric Costs Customer Revenue

### Four Primary Value Drivers

1. Faster, More Agile Delivery and Reduced Time to Market
2. Improved Security Posture and Reduced Risk
3. Reduced Operational and Development Costs
4. Improved Customer Experiences and Satisfaction

### Metrics

Value Driver	Metrics	Costs	Customer Revenue
--------------	---------	-------	------------------

Faster, More Agile Delivery and Reduced Time to Market	Release Frequency, Time to Market	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time	Revenue from New Customers, Revenue from Increase in Share of Wallet, Revenue from Accelerated Time to Market, Revenue from New Products, Revenue from Pricing Innovation
Improved Security Posture and Reduced Risk	Issues Identified in Dev & QA Environments, MTTD/MTTR, FTEs Involved per Incident	Financial Losses Due to Performance Degradation or Security Incidents, Tech Support Center Costs, Engineer Time to Resolve Incidents	Lost Revenue Due to Outages, Revenue from Reduced Churn, Revenue from Higher Share of Wallet
Reduced Operational and Development Costs	Incidents/Outages, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time, Over-Provisioning Infrastructure	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time, Over-Provisioning Infrastructure	Revenue from Reduced Churn, Revenue from Higher Share of Wallet
Improved Customer Experiences and Satisfaction	Customer Complaint Calls, Customer Satisfaction, Customer Share of Wallet, Customer Churn	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time	Revenue from New Customers, Revenue from Increase in Share of Wallet, Revenue from Reduced Churn, Revenue from Higher Share of Wallet

Note: This table outlines the key metrics, costs, and customer revenue impacts associated with each value driver in DevSecOps.

## Model

ID	Stage	Level 1: Basic understanding of security practices	Level 2: Adoption of basic security practices	Level 3: High adoption of security practices	Level 4: Very high adoption of security practices	Level 5: Advanced deployment of security practices at scale
1	Build and Deployment: Build	Defined build process	Building and testing of artifacts in virtual environments	Pinning of artifacts	SBOM of components	Signing of code
		[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
		Signing of artifacts				
2	Build and Deployment: Deployment	Defined deployment process	Defined decommissioning process	Environment depending configuration parameters (secrets)	Evaluation of the trust of used components	Handover of confidential parameters
		[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
		Inventory of dependencies	Inventory of running artifacts	Rolling update on deployment	Same artifact for environments	Usage of feature toggles
3	Build and Deployment: Patch Management	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
		Blue/Green Deployment				
		[ none ]				
4	Build and Deployment: Patch Management	A patch policy is defined	Automated PRs for patches	Nightly build of images (base images)	Reduction of the attack surface	Usage of a maximum lifetime for images
		[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
		Usage of a short maximum lifetime for images				

4	Culture and Organization: Design	Conduction of simple threat modeling on technical level	Information security targets are communicated	Conduction of simple threat modeling on business level	Creation of simple abuse stories
		[ none ]	[ none ]	[ none ]	[ none ]
		Conduction of advanced threat modeling	Creation of advanced abuse stories		
5	Culture and Organization: Education and Guidance	[ none ]	[ none ]		
		Ad-Hoc Security trainings for software developers	Security consulting on request	Each team has a security champion	Regular security training of security champions
		[ none ]	[ none ]	[ none ]	[ none ]
6	Culture and Organization: Process	Reward of good communication	Security code review	Conduction of build-it, break-it, fix-it contests	Security Coaching
		[ none ]	[ none ]	[ none ]	[ none ]
		Simple mob hacking	Aligning security in teams	Conduction of collaborative team security checks	Conduction of war games
7	Implementation: Application Hardening	[ none ]	[ none ]	[ none ]	[ none ]
		Conduction of collaborative security checks with developers and system administrators			
		[ none ]			
8	Implementation: Development and Source Control	Definition of simple BCDR practices for critical components	Approval by reviewing any new version	Definition of a change management process	
		[ none ]	[ none ]	[ none ]	
		App. Hardening Level 1 (50%)	Contextualized Encoding	App. Hardening Level 1	App. Hardening Level 2 (75%)
9	Implementation: Infrastructure Hardening	[ none ]	[ none ]	[ none ]	[ none ]
		App. Hardening Level 3			
		[ none ]			
		Versioning	Source Control Protection	API design validation	gitignore
		[ none ]	[ none ]	[ none ]	[ none ]
		MFA for admins	Simple access control for systems	Usage of edge encryption at transit	Applications are running in virtualized environments
		[ none ]	[ none ]	[ none ]	[ none ]
		Baseline Hardening of the environment	Isolated networks for virtual environments	MFA	Usage of an security account
					Usage of encryption at rest

	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Usage of test and production environments	Virtual environments are limited	Filter outgoing traffic	Immutable infrastructure	Infrastructure as Code
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Limitation of system events	Role based authentication and authorization	Usage of internal encryption at transit	Usage of security by default for components	WAF baseline
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Hardening of the Environment	Production near environments are used by developers	Usage of a chaos monkey	WAF medium	Microservice-architecture
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	WAF Advanced				
	[ none ]				
10	Information Gathering: Logging	Centralized system logging	Logging of security events	Visualized logging	Centralized application logging
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	PII logging concept				
	[ none ]				
11	Information Gathering: Monitoring	Simple application metrics	Simple budget metrics	Simple system metrics	Alerting
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Visualized metrics	Advanced availability and stability metrics	Audit of system events	Deactivation of unused metrics	Grouping of metrics
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Targeted alerting	Advanced app. metrics	Coverage and control metrics	Defense metrics	Screens with metric visualization
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Metrics are combined with tests				
	[ none ]				
12	Test and Verification: Application tests	Security unit tests for important components	Security integration tests for important components	Smoke Test	High coverage of security related module and integration tests
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
13	Test and Verification: Consolidation	Simple false positive treatment	Treatment of defects with severity high or higher	Simple visualization of defects	Generation of Patch Management Statistics
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Integration of vulnerability issues into the development process	Treatment of defects with severity middle	Usage of a vulnerability management system	Advanced visualization of defects	Reproducible defect tickets
	[ none ]	[ none ]	[ none ]	[ none ]	[ none ]
	Treatment of all defects				

		[ none ]			
14	Test and Verification: Dynamic depth for applications	Coverage of client side dynamic components	Simple Scan	Usage of different roles	Coverage of hidden endpoints
		[ none ]	[ none ]	[ none ]	[ none ]
		Coverage of sequential operations	Usage of multiple scanners	Coverage analysis	Coverage of service to service communication
15		[ none ]	[ none ]	[ none ]	[ none ]
	Test and Verification: Dynamic depth for infrastructure	Test for exposed services	Test network segmentation	Test of the configuration of cloud environments	Test for unauthorized installation
		[ none ]	[ none ]	[ none ]	[ none ]
16		Load tests	Test for unused Resources		
		[ none ]	[ none ]		
	Test and Verification: Static depth for applications	Software Composition Analysis (server side)	API design validation	Static analysis for important server side components	Local development security checks performed
		[ none ]	[ none ]	[ none ]	[ none ]
		Static analysis for important client side components	Static analysis for all self written components	Usage of multiple analyzers	Dead code elimination
17		[ none ]	[ none ]	[ none ]	[ none ]
		Static analysis for all components/libraries	Stylistic analysis		
		[ none ]	[ none ]		
	Test and Verification: Static depth for infrastructure	Test for stored secrets	Test cluster deployment resources	Test for image lifetime	Test of virtualized environments
		[ none ]	[ none ]	[ none ]	[ none ]
18		Test the definition of virtualized environments	Analyze logs	Test for malware	Test for new image version
		[ none ]	[ none ]	[ none ]	[ none ]
		Test for known vulnerabilities	Test of infrastructure components for known vulnerabilities		Correlate known vulnerabilities in infrastructure with new image versions
		[ none ]	[ none ]		
	Test and Verification: Test-Intensity	Default settings for intensity	High test intensity	Regular tests	Deactivating of unneeded tests
		[ none ]	[ none ]	[ none ]	[ none ]

## Tools

### Git Secrets Management Tools

## git-secrets

---

- **Description:** AWS Lambda tool preventing you from committing secrets to a git repository.
- **Uses Method:** Prevents accidental commit of sensitive information to git repositories.
- **Suggested Maturity Model:** Early to Mature stages of DevSecOps implementation.

## git-hound

---

- **Description:** Searches secrets in git.
- **Uses Method:** Scans git repositories for secrets and sensitive information.
- **Suggested Maturity Model:** Mature DevSecOps practices.

## Threat Modeling Tools

---

### goSDL

- **Description:** Security Development Lifecycle checklist.
- **Uses Method:** Provides a checklist for integrating security into the development lifecycle.
- **Suggested Maturity Model:** Early stages of DevSecOps implementation.

### ThreatPlaybook

- **Description:** Threat modeling as code.
- **Uses Method:** Enables codified threat modeling for better integration into DevOps.
- **Suggested Maturity Model:** Intermediate to Mature stages.

### Threat Dragon

- **Description:** OWASP Threat modeling tool.
- **Uses Method:** Offers a visual tool for threat modeling in software development.
- **Suggested Maturity Model:** Any stage of DevSecOps maturity.

## Secrets Management

---

### GitLeaks

- **Description:** GitLeaks is a scanning tool for detecting hardcoded secrets.
- **Uses Method:** Scans repositories for secrets and sensitive information.
- **Suggested Maturity Model:** Intermediate to Mature stages.

### ggshield

- **Description:** GitGuardian shield (ggshield) is a CLI application that runs in your local environment or in a CI environment and helps you detect more than 350+ types of secrets and sensitive files.
- **Uses Method:** Local and CI environment scanning for secrets.
- **Suggested Maturity Model:** Mature DevSecOps practices.

## OSS and Dependency Management

## CycloneDX

- **Description:** CycloneDX format for SBOM.
- **Uses Method:** Provides a standard format for software bill of materials.
- **Suggested Maturity Model:** Intermediate to Mature stages.

## Snyk

- **Description:** Scans and monitors your projects for security vulnerabilities.
- **Uses Method:** Continuous scanning and monitoring of project dependencies.
- **Suggested Maturity Model:** Early to Mature stages of DevSecOps implementation.

## AST (Static Application Security Testing)

### Brakeman

- **Description:** Brakeman is a static analysis tool which checks Ruby on Rails applications for security vulnerabilities.
- **Uses Method:** Static code analysis for Ruby on Rails applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

### Semgrep

- **Description:** Hi-Quality Open source, works on 17+ languages.
- **Uses Method:** Static code analysis across multiple programming languages.
- **Suggested Maturity Model:** Any stage of DevSecOps maturity.

## DAST (Dynamic Application Security Testing)

### Zap proxy

- **Description:** Zap proxy providing various docker containers for CI/CD pipeline.
- **Uses Method:** Offers dynamic application security testing, particularly for web applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

### Wapiti

- **Description:** Light pipeline ready scanning tool.
- **Uses Method:** Dynamic security scanning of web applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

## Continuous Deployment Security

1. **SecureCodeBox**
  - **URL:** [SecureCodeBox](#)
  - **Description:** Toolchain for continuous scanning of applications and infrastructure.
  - **Uses Method:** SCB
  - **Suggested Maturity Model:** Advanced Continuous Scanning Integration
2. **OpenSCAP**
  - **URL:** [OpenSCAP](#)
  - **Description:** Open Source Security Compliance Solution.
  - **Uses Method:** oscap
  - **Suggested Maturity Model:** Compliance and Security Standards Enforcement
3. **ThreatMapper**

- **URL:** [Kube-hunter](#)
- **Description:** Hunts for vulnerabilities in production platforms, ranking them based on risk-of-exploit.
- **Uses Method:** kube-hunter
- **Suggested Maturity Model:** Real-time Vulnerability Assessment

## Kubernetes

---

1. **KubiScan**
  - **URL:** [KubiScan](#)
  - **Description:** A tool for scanning Kubernetes cluster for risky permissions.
  - **Uses Method:** Kubiscan
  - **Suggested Maturity Model:** Permission Management and Risk Assessment
2. **Kubeaudit**
  - **URL:** [Kubeaudit](#)
  - **Description:** Audit Kubernetes clusters for various security concerns.
  - **Uses Method:** kube-audit
  - **Suggested Maturity Model:** Comprehensive Cluster Auditing
3. **Kubescape**
  - **URL:** [Kubescape](#)
  - **Description:** First open-source tool for testing Kubernetes deployment according to NSA-CISA and MITRE ATT&CK®.
  - **Uses Method:** kubescape
  - **Suggested Maturity Model:** NSA-CISA Compliance and Security Testing

## Continuous Deployment Security

---

1. **SecureCodeBox**
  - **URL:** [SecureCodeBox](#)
  - **Description:** Toolchain for continuous scanning of applications and infrastructure.
  - **Uses Method:** SCB
  - **Suggested Maturity Model:** Advanced Continuous Scanning Integration
2. **OpenSCAP**
  - **URL:** [OpenSCAP](#)
  - **Description:** Open Source Security Compliance Solution.
  - **Uses Method:** escap
  - **Suggested Maturity Model:** Compliance and Security Standards Enforcement
3. **ThreatMapper**
  - **URL:** [ThreatMapper](#)
  - **Description:** Hunts for vulnerabilities in production platforms, ranking them based on risk-of-exploit.
  - **Uses Method:** kube-hunter
  - **Suggested Maturity Model:** Real-time Vulnerability Assessment

## Kubernetes

---

1. **KubiScan**
  - **URL:** [KubiScan](#)
  - **Description:** A tool for scanning Kubernetes cluster for risky permissions.
  - **Uses Method:** Kubiscan
  - **Suggested Maturity Model:** Permission Management and Risk Assessment
2. **Kubeaudit**
  - **URL:** [Kubeaudit](#)
  - **Description:** Audit Kubernetes clusters for various security concerns.
  - **Uses Method:** kube-audit
  - **Suggested Maturity Model:** Comprehensive Cluster Auditing
3. **Kubescape**
  - **URL:** [Kubescape](#)
  - **Description:** First open-source tool for testing Kubernetes deployment according to NSA-CISA and MITRE ATT&CK®.
  - **Uses Method:** kubescape
  - **Suggested Maturity Model:** NSA-CISA Compliance and Security Testing
4. **kubesec**
  - **URL:** [kubesec](#)
  - **Description:** Security risk analysis for Kubernetes resources.
  - **Uses Method:** kubesec
  - **Suggested Maturity Model:** Kubernetes Resource Security Analysis
5. **kube-bench**
  - **URL:** [kube-bench](#)
  - **Description:** Kubernetes benchmarking tool.
  - **Uses Method:** Kubiscan

- Suggested Maturity Model: Kubernetes Compliance Benchmarking

6. kube-score

- URL: [kube-score](#)
- Description: Static code analysis of your Kubernetes object definitions.
- Uses Method: kube-score
- Suggested Maturity Model: Code Quality and Security Scoring

7. kube-hunter

- URL: [kube-hunter](#)
- Description: Active scanner for Kubernetes (purple teaming).
- Uses Method: kube-hunter
- Suggested Maturity Model: Active Vulnerability Scanning

8. Calico

- URL: [Calico](#)
- Description: Open source networking and network security solution for containers.
- Uses Method: Calico
- Suggested Maturity Model: Network Security and Management

9. Krane

- URL: [Krane](#)
- Description: Simple Kubernetes RBAC static analysis tool.
- Uses Method: krane
- Suggested Maturity Model: Role-Based Access Control Analysis

10. Starboard

- URL: [Starboard](#)
- Description: Integrates security tools outputs into Kubernetes CRDs.
- Uses Method: starboard
- Suggested Maturity Model: Security Tool Integration and Reporting

11. Gatekeeper

- URL: [Gatekeeper](#)
- Description: Open policy agent gatekeeper for Kubernetes.
- Uses Method: gatekeeper
- Suggested Maturity Model: Policy Enforcement and Management

12. Inspektr-gadget

- URL: [inspektr-gadget](#)
- Description: Collection of tools for debugging and inspecting Kubernetes.
- Uses Method: inspector
- Suggested Maturity Model: Debugging and Inspection Toolset

## Containers

1. Harbor

- Description: Trusted cloud native registry project.
- Uses Method: Harbor
- Suggested Maturity Model: Container Registry Security

2. Anchore

- Description: Centralized service for inspection, analysis, and certification of container images.
- Uses Method: Anchore
- Suggested Maturity Model: Container Image Analysis and Certification

3. Clair

- Description: Docker vulnerability scanner.
- Uses Method: Clair
- Suggested Maturity Model: Container Vulnerability Scanning

4. Deepfence ThreatMapper

- Description: Powerful runtime vulnerability scanner for Kubernetes, virtual machines, and serverless.
- Uses Method: ThreatMapper
- Suggested Maturity Model: Runtime Vulnerability Scanning

5. Docker Bench

- Description: Docker benchmarking against CIS.
- Uses Method: Docker Bench
- Suggested Maturity Model: Docker Security Benchmarking

6. Falco

- Description: Container runtime protection.
- Uses Method: Falco
- Suggested Maturity Model: Runtime Security Monitoring

7. Trivy

- Description: Comprehensive scanner for vulnerabilities in container images.
- Uses Method: Trivy
- Suggested Maturity Model: Container Image Vulnerability Scanning

8. Notary

- **Description:** Docker signing.
- **Uses Method:** Notary
- **Suggested Maturity Model:** Container Image Signing

3. Cosign

- **Description:** Container signing.
- **Uses Method:** Cosign
- **Suggested Maturity Model:** Secure Container Signing

10. Watchtower

- **Description:** Updates the running version of your containerized app.
- **Uses Method:** Watchtower
- **Suggested Maturity Model:** Continuous Container Update Management

11. Grype

- **Description:** Vulnerability scanner for container images and filesystems.
- **Uses Method:** Grype
- **Suggested Maturity Model:** Comprehensive Container Scanning

## Multi-Cloud

---

1. Cloudsploit

- **Description:** Detection of security risks in cloud infrastructure.
- **Uses Method:** Cloudsploit
- **Suggested Maturity Model:** Cloud Security Risk Assessment

2. ScoutSuite

- **Description:** NCCgroup multicloud scanning tool.
- **Uses Method:** ScoutSuite
- **Suggested Maturity Model:** Multicloud Security Scanning

3. CloudCustodian

- **Description:** Multicloud security analysis framework.
- **Uses Method:** CloudCustodian
- **Suggested Maturity Model:** Cloud Security Governance

4. CloudGraph

- **Description:** GraphQL API + Security for AWS, Azure, GCP, and K8s.
- **Uses Method:** CloudGraph
- **Suggested Maturity Model:** Cloud Resource Visualization and Security

5. Steampipe

- **Description:** Query your cloud, code, logs & more with SQL. Open-source benchmarks & dashboards for security & insights.
- **Uses Method:** Steampipe
- **Suggested Maturity Model:** Cloud Data Querying and Security Insights

## AWS

---

1. Dragoneye

- **Description:** Dragoneye Idenit AWS scanner.
- **Uses Method:** Dragoneye
- **Suggested Maturity Model:** AWS Security Scanning

2. Prowler

- **Description:** Command line tool for AWS security assessment, auditing, hardening, and incident response.
- **Uses Method:** Prowler
- **Suggested Maturity Model:** AWS Security Assessment and Auditing

3. AWS Inventory

- **Description:** Helps to discover all AWS resources created in an account.
- **Uses Method:** AWS inventory
- **Suggested Maturity Model:** AWS Resource Management and Security

4. PacBot

- **Description:** Policy as Code Bot (PacBot).
- **Uses Method:** PacBot
- **Suggested Maturity Model:** Policy as Code for AWS

5. Komiser

- **Description:** Monitoring dashboard for costs and security.
- **Uses Method:** Komiser
- **Suggested Maturity Model:** AWS Cost and Security Monitoring

6. Cloudsplaining

- **Description:** IAM analysis framework.
- **Uses Method:** Cloudsplaining
- **Suggested Maturity Model:** AWS IAM Security Analysis

7. ElectricEye

- **Description:** Continuously monitor your AWS services for configurations.

- **Uses Method:** ElectricEye
- **Suggested Maturity Model:** Continuous AWS Security Monitoring
- 8. **CloudMapper**
  - **Description:** Helps you analyze your Amazon Web Services environments.
  - **Uses Method:** CloudMapper
  - **Suggested Maturity Model:** AWS Environment Analysis
- 9. **Cartography**
  - **Description:** Consolidates AWS infrastructure assets and the relationships between them in an intuitive graph.
  - **Uses Method:** Cartography
  - **Suggested Maturity Model:** AWS Asset Visualization and Management
- 10. **Policy Sentry**
  - **Description:** IAM Least Privilege Policy Generator.
  - **Uses Method:** Policy Sentry
  - **Suggested Maturity Model:** AWS IAM Policy Optimization

... (and so on for each tool in the AWS category)

## Google Cloud Platform

---

- 1. **Forseti**
  - **Description:** Complex security orchestration and scanning platform.
  - **Uses Method:** Forseti
  - **Suggested Maturity Model:** GCP Security Orchestration
- 2. **GCP Insights**
  - **Description:** Visualize GCP inventory and permissions through relationship graphs.
  - **Uses Method:** GCP Insights
  - **Suggested Maturity Model:** GCP Security Visualization
- 3. **GCP Compliance**
  - **Description:** Check compliance of GCP configurations to security best practices.
  - **Uses Method:** GCP Compliance
  - **Suggested Maturity Model:** GCP Compliance Assessment

## Microsoft Azure

---

- 1. **Azure Insights**
  - **Description:** Visualize Azure inventory and permissions through relationship graphs.
  - **Uses Method:** Azure Insights
  - **Suggested Maturity Model:** Azure Security Visualization
- 2. **Azure Compliance**
  - **Description:** Check compliance of Azure configurations to security best practices.
  - **Uses Method:** Azure Compliance
  - **Suggested Maturity Model:** Azure Compliance Assessment

## Policy as Code

---

- 1. **Open Policy Agent (OPA)**
  - **Description:** General-purpose policy engine for unified, context-aware policy enforcement across the stack.
  - **Uses Method:** OPA
  - **Suggested Maturity Model:** Cross-Platform Policy Enforcement
- 2. **Kyverno**
  - **Description:** Policy engine designed for Kubernetes.
  - **Uses Method:** Kyverno
  - **Suggested Maturity Model:** Kubernetes Policy Management
- 3. **Inspec**
  - **Description:** Open-source testing framework for infrastructure with language for compliance, security, and policy requirements.
  - **Uses Method:** Inspec
  - **Suggested Maturity Model:** Infrastructure Compliance Testing
- 4. **Cloud Formation Guard**
  - **Description:** Cloud Formation policy as code.
  - **Uses Method:** CF-Guard
  - **Suggested Maturity Model:** AWS CloudFormation Policy Management
- 5. **CNSpec**
  - **Description:** Cloud-native and powerful Policy as Code engine for security and compliance assessment across various infrastructures.
  - **Uses Method:** CNSpec
  - **Suggested Maturity Model:** Multi-Infrastructure Security Assessment

# Chaos Engineering

---

1. Chaos Mesh
  - Description: Cloud-native Chaos Engineering platform that orchestrates chaos on Kubernetes environments.
  - Uses Method: Chaos Mesh
  - Suggested Maturity Model: Kubernetes Resilience Testing
2. Chaos Monkey
  - Description: Randomly terminates instances in production to ensure service resilience.
  - Uses Method: Chaos Monkey
  - Suggested Maturity Model: Production Resilience Assurance
3. Chaos Engine
  - Description: Tool designed to intermittently destroy or degrade application resources in cloud infrastructure.
  - Uses Method: Chaos Engine
  - Suggested Maturity Model: Cloud Infrastructure Resilience Testing
4. Chaoskube
  - Description: Test how your system behaves under arbitrary pod failures.
  - Uses Method: Chaoskube
  - Suggested Maturity Model: Kubernetes Pod Failure Testing
5. Kube-Invaders
  - Description: Gamified chaos engineering tool for Kubernetes.
  - Uses Method: Kube-Invaders
  - Suggested Maturity Model: Kubernetes Gamified Resilience Testing

# Infrastructure as Code Security

---

1. KICS
  - Description: Checkmarx security testing open source for Infrastructure as Code (IaC).
  - Uses Method: KICS
  - Suggested Maturity Model: Open Source IaC Security Testing
2. Checkov
  - Description: Static code analysis tool for infrastructure-as-code.
  - Uses Method: Checkov
  - Suggested Maturity Model: Static Code Analysis for IaC
3. tfsec
  - Description: Uses static analysis of Terraform templates to spot potential security issues.
  - Uses Method: tfsec
  - Suggested Maturity Model: Terraform Security Analysis
4. terrascan
  - Description: Static code analyzer for Infrastructure as Code.
  - Uses Method: terrascan
  - Suggested Maturity Model: IaC Code Scanning
5. cfsec
  - Description: Scans CloudFormation configuration files for security issues.
  - Uses Method: cfsec
  - Suggested Maturity Model: CloudFormation Security Scanning
6. cfn\_nag
  - Description: Looks for insecure patterns in CloudFormation.
  - Uses Method: cfn\_nag
  - Suggested Maturity Model: CloudFormation Pattern Analysis
7. Sysdig IaC Scanner Action
  - Description: Scans your repository with Sysdig IaC Scanner and reports vulnerabilities.
  - Uses Method: Sysdig IaC Scanner
  - Suggested Maturity Model: Repository Vulnerability Scanning
8. Terraform Compliance for AWS
  - Description: Checks compliance of Terraform configurations to AWS security best practices.
  - Uses Method: Terraform Compliance
  - Suggested Maturity Model: AWS Terraform Compliance
9. Terraform Compliance for Azure
  - Description: Checks compliance of Terraform configurations to Azure security best practices.
  - Uses Method: Terraform Compliance
  - Suggested Maturity Model: Azure Terraform Compliance
10. Terraform Compliance for GCP
  - Description: Checks compliance of Terraform configurations to GCP security best practices.
  - Uses Method: Terraform Compliance
  - Suggested Maturity Model: GCP Terraform Compliance
11. Terraform Compliance for OCI

- **Description:** Checks compliance of Terraform configurations to OCI security best practices.
- **Uses Method:** Terraform Compliance
- **Suggested Maturity Model:** OCI Terraform Compliance

## Orchestration

1. StackStorm
  - **Description:** Platform for integration and automation across services and tools supporting event-driven security.
  - **Uses Method:** StackStorm
  - **Suggested Maturity Model:** Event-Driven Security Automation
2. Camunda
  - **Description:** Workflow and process automation.
  - **Uses Method:** Camunda
  - **Suggested Maturity Model:** Process Automation for Security
3. DefectDojo
  - **Description:** Security orchestration and vulnerability management platform.
  - **Uses Method:** DefectDojo
  - **Suggested Maturity Model:** Vulnerability Management Orchestration
4. Faraday
  - **Description:** Security suite for Security Orchestration, vulnerability management, and centralized information.
  - **Uses Method:** Faraday
  - **Suggested Maturity Model:** Comprehensive Security Orchestration

## Controls

### Development Environment

#### Security Control: Secure Code Training

- **Description:** Training developers in secure coding to reduce security bugs and enhance system design.
- **Control(s):** CISB, APRA234, NIST 800-53B, SSDF1.1, ISO27001

#### Security Control: Source Code Versioning

- **Description:** Using Version Control Systems for peer review, auditable history, and consistent work patterns.
- **Control(s):** APRA234, CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

#### Security Control: .gitignore

- **Description:** Preventing accidental commits of sensitive data with .gitignore files.
- **Control(s):** APRA234, CISB, NIST 800-53B, SSDF1.1

#### Security Control: Pre-Commit Hook Scans

- **Description:** Using Pre-Commit Hooks for security scans to provide timely feedback and prevent vulnerabilities.
- **Control(s):** APRA234, CISB, NIST 800-53B, SSDF1.1

#### Security Control: Commit Signing

- **Description:** Signing all commits to verify author authenticity.
- **Control(s):** APRA234, CISB, NIST 800-53B, SSDF1.1

#### Security Control: IDE Plugins

- **Description:** Implementing IDE plugins to highlight security issues in real-time.
- **Control(s):** APRA234, CISB, NIST 800-53B, SSDF1.1

#### Security Control: Local Software Composition Analysis

- **Description:** Finding and fixing libraries with known security issues.
- **Control(s):** APRA234, CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: Local Static Code Analysis

---

- Description: Identifying and resolving security vulnerabilities in source code.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: Local Sensitive Data Analysis

---

- Description: Auditing repositories for secrets, credentials, and API keys.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: Application Baseline

---

- Description: Creating a comprehensive "recipe" for building applications considering risk, compliance, and technical components.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53B, SSDF1.1

## Source Code Management (SCM)

---

### Security Control: Source Code Management

- Description: Using centralized SCM systems like Bitbucket, GitHub, or Gitlab.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

### Security Control: User Roles

- Description: Creating unique user and team roles for tailored source code access.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53B, SSDF1.1

### Security Control: SSH

- Description: Accessing repositories using SSH protocol instead of HTTPS.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53B, SSDF1.1

### Security Control: GPG Key

- Description: Adding a GPG key to SCM providers for identity verification.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53B, SSDF1.1

### Security Control: Multi-Factor Authentication

- Description: Ensuring MFA is used for SCM interactions.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53B, SSDF1.1

### Security Control: Server Side Git Hook

- Description: Utilizing server-side git hooks for automatic scans.
- Control(s): APRA234, CIS8, NIST 800-53B, SSDF1.1

### Security Control: Developer Collaboration

- Description: Using collaboration tools for documenting software changes.
- Control(s): APRA234, CIS8, NIST 800-53B, SSDF1.1

### Security Control: Pull Requests

- Description: Enforcing pull or merge requests for code verification.
- Control(s): APRA234, CIS8, ISO27001, NIST 800-53B, SSDF1.1

### Security Control: Peer Reviews

- Description: Enforcing peer reviews to enhance code quality and security.
- Control(s): APRA234, CIS8, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: CODEOWNERS

- Description: Creating a CODEOWNERS file to identify repository owners.
- Control(s): APRA234, CIS8, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: SECURITY.md

- Description: Creating a SECURITY.md file for reporting security issues.
- Control(s): APRA234, CIS8, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: .github Repository

- Description: Creating a .github repository for standard files across the organization.
- Control(s): APRA234, CIS8, ISO27001, NIST 800-53B, SSDF1.1

## CI/CD Pipelines and Automation

### Security Control: CI/CD Pipeline

- Description: Implementing a CI/CD pipeline for continuous integration and deployment.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, SSDF1.1

### Security Control: Application Environments

- Description: Creating separate environments for development, staging, and production.
- Control(s): CIS8, ISM GSD, ISO27001, SSDF1.1

### Security Control: Application Data Separation

- Description: Ensuring data separation between dev/test and production environments.
- Control(s): CIS8, ISM GSD, ISO27001, SSDF1.1

### Security Control: CI/CD Administration

- Description: Enforcing user or team roles for CI/CD pipeline management.
- Control(s): CIS8, ISM GSD, ISO27001, SSDF1.1

### Security Control: Credential Store

- Description: Creating a secure place for storing sensitive credentials.
- Control(s): APRA234, CIS8, ISM GSD, NIST 800-53.2b, SSDF1.1

### Security Control: Centralized Software Composition Analysis

- Description: Scanning source code for vulnerable libraries from within a CD stage.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53.2a, SSDF1.1

### Security Control: Centralized Static Code Analysis

- Description: Scanning source code for vulnerabilities from within a CD stage.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53.2b, SSDF1.1

### Security Control: Centralized Sensitive Data Analysis

- Description: Scanning source code for secrets and credentials from within a CD stage.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

### Security Control: DAST

- Description: Scanning running applications for vulnerabilities.
- Control(s): CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Security Control: Transient Test Compute

- Description: Ensuring up-to-date compute resources in CI/CD pipelines.
- Control(s): CISB, ISM GSD, ISO27001, SSDF1.1

## Security Control: Harden Transient Compute

- Description: Hardening transient compute resources used in pipelines.
- Control(s): CISB, ISM GSM, ISM GOSH, SSDF1.1

## Valid SSL Certificate

- Description: Create and use a valid SSL certificate for each application URL, or implement a wildcard cert.
- Control(s): APRA234, CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Encrypt Traffic

- Description: Encrypt all traffic that's public facing.
- Control(s): APRA234, CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1, DSOMM: Infrastructure Hardening Level 1

## Redirect to HTTPS

- Description: Configure web service to redirect all inbound requests to port 80 to the secure HTTPS endpoint.
- Control(s): CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1, DSOMM: Application Hardening Level 1

## HSTS

- Description: Enable HSTS in your webserver, load balancer or CDN.
- Control(s): CISB, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1, DSOMM: Application Hardening Level 1

## CSP

- Description: Enable content security policy (CSP) in the webserver, load balancer or CDN.
- Control(s): CISB, ISM GSD, ISO27001, NIST 800-53B, DSOMM: Application Hardening Level 1

## Use Current Software

- Description: Use the most recent versions of application components, languages, frameworks and operating systems.
- Control(s): CISB, ISM GSD, ISO27001, SSDF1.1, DSOMM: Application Hardening Level 1

## Alternative Deployment

- Description: Have tested and working alternative way to deploy changes to your application other than using your standard process with GitHub or Bitbucket in case they go down. This must include the ability to push to PROD from local in emergencies.
- Control(s): CISB, NIST 800-53B, SSDF1.1

## security.txt

- Description: Create a security.txt file in the root of your application so people know how to contact you about security issues.
- Control(s): CISB, ISM GSD, SSDF1.1

## X-Forwarded-By

- Description: Configure your webservers, load balancers & web proxies to include the X-Forwarded-By: header.
- Control(s): APRA234 ATM D-2-d-l, CISB, NIST 800

## Logging

- Description: Collect application logs in realtime and send to centralized storage or SIEM.
- Control(s): CISB 16.11, APRA234, ISM GSM, NIST 800, SSDF1.1, DSOMM: Logging Level 1

## WAF

- Description: Implement a web application firewall (WAF) to protect your application from known attacks.
- Control(s): APRA234, CIS8, NIST 800-53.2a

## CDN

- Description: Use a content delivery network (CDN) whenever possible to add availability and security to your applications.
- Control(s): APRA234, CIS8, ISM GN, NIST 800-53.2a, DSOMM: Application Hardening Level 1

## Harden Operating System

- Description: Harden operating system using industry best practices from CIS, ISM, etc.
- Control(s): CIS8, ISM GSM, ISM GOSH, SSDF1.1

## Encrypt Storage

- Description: Encrypt all filesystems, disks and cloud storage.
- Control(s): CIS8, NIST 800-50b, SSDF1.1, DSOMM: Infrastructure Hardening Level 1

## SBOM

- Description: Generate a real-time software bill-of-materials (SBOM).
- Control(s): CIS8, ISM GSD, NIST 800-53B, SSDF1.1, DSOMM: Build Level 2

## Monitor Application

- Description: Monitor your application in real-time so you know when its state changes for the worse (or better). This includes uptime, performance and security monitoring.
- Control(s): CIS8, NIST 800-53B, SSDF1.1, DSOMM: Monitoring Levels 1 - 4

## Cloud Security Posture

- Description: If your application is deployed in the cloud or uses cloud native services then a solution should be employed to verify that those cloud resources are secure and follow best practices.
- Control(s): CIS8, NIST 800-53B, SSDF1.1

## Centralized Container Analysis

- Description: Scan any containers built for deployment for vulnerabilities.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53.2a, SSDF1.1, DSOMM: Build Level 2

## IaC

- Description: Use infrastructure as code to build all application environments.
- Control(s): CIS8, ISM GSM, ISM GOSH, SSDF1.1, DSOMM: Infrastructure Hardening Level 3

## TLS 1.3

- Description: Use TLS 1.3 instead of earlier versions. TLS 1.2 is still okay, but you should enable 1.3 soon as its more secure than any of the earlier versions.
- Control(s): APRA234, CIS8, ISM GSD, ISO27001, NIST 800-53B, SSDF1.1

## Organizational Techniques

### Penetration Testing

- Description: Have your application pentested regularly.
- Control(s): CIS8, ISM GSD, NIST 800-53B, SSDF1.1

### Threat Modeling

- Description: Build a collaborative way for developers and security staff to understand the threat landscape for an individual application.

## SIEM

---

- Description: Implement a SIEM and send all application, system and cloud logs to it.
- Control(s): CISB, NIST 800-53B, SSDF1.1

## Attack Surface Management

---

- Description: Identify public facing resources via automation.
- Control(s): CISB, NIST 800-53B, SSDF1.1, DSOMM: Logging Level 4, DSOMM: Dynamic Depth for Applications Level 2

## Sovereignty

---

- Description: Require that all code is written in, stored in, or otherwise served from a location and/or sovereignty that aligns with your org's requirements.
- Control(s): ISM GCSR, ISO27001

## Vulnerability Disclosure

---

- Description: Create and publish a set of procedures to let people contact you when they find security issues in your app.
- Control(s): CISB, ISM GSD, SSDF1.1

## Bug Bounty

---

- Description: Setup a bug bounty program to incentivize security researchers to tell you about vulnerabilities they find.
- Control(s): CISB, ISM GSD, NIST 800-53B, SSDF1.1

## Licensing

---

- Description: Track licensing of all software that your organization uses or depends on. Utilize a license tracking solution to enable searching for license types.
- Control(s): ?

## Vulnerability Aggregation and Management

---

- Description: Implement a system that aggregates vulnerability data from multiple tools and allows teams to prioritize, collaborate on, and manage the lifecycle of said vulnerabilities.
- Control(s): CISB, ISM GSD, SSDF1.1







