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. For authentication errors, do not indicate that the username exists.	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

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 strong TLS configurations	TLS must be configured to the secure configurations that only support the recent versions of TLS, prefer the use of the strongest cipher suites and avoid the use of any weak ciphers. For example, SSL and TLS protocols prior to TLS 1.2 have known weaknesses and are not considered secure. Additionally, disable the cipher suites using RC4, DES or MD5 and prefer the ciphers that support Perfect Forward Secrecy. **EXAMPLE:** Qualys SSL Labs	
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: https://haveibeenpwned.com**	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 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 that the application meets security standards. Testing should also be conducted after major releases to ensure that 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 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. Proper secrets management tools can provide proper encryption and credentials rotation to provide extra resiliency to attacks. **EXAMPLE:** Hardcoded 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:** Sarah 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 accidentially 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	The session cookie should have the HttpOnly, Secure, and SameSite flags set. This ensures that the session ID will not be accessible to client-side scripts, will only be transmitted over HTTPS, and will only be sent with requests from the same site (mitigates CSRF).	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.	
Use non-persistent cookies	If a cookie has the "Max-Age" or "Expires" attributes, the browser treats it as a persistent cookie and stores it to disk until the expiration time. Do not do this for session cookies.	

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. **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
Prevent insecure deserialization	Do not accept serialized objects from untrusted sources, define known good data types when deserializing data, and implement integrity checks on serialized objects.	CWE-502
Use tokens to prevent forged requests	In order to prevent Cross-Site Request Forgery attacks, you must embed a random value that is not known to third parties into the HTML form. This CSRF protection token must be unique to each request. This prevents a forged CSRF request from being submitted because the attacker does not know the value of the token.	CWE-352
Prevent Server Side Request Forgery (SSRF)	Features that require requests to be sent to web services need to carefully restrict URLs by validating input and properly encoding output.	
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	Use the "rel" anchor tag attribute with values of "noopener" or "noreferrer" to prevent an opened tab from tampering with the calling tabs location in the browser. In JavaScript this can be prevented by setting window.opener to null.	CWE-1022
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 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, 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 resources	An unvalidated forward or resource use can allow an attacker to access private content without authentication. Unvalidated redirects allow an attacker to lure victims into visiting malicious sites. Similarly, unvalidated usage of URLs can lead to issues such as Server Side Request Forgery (SSRF). Prevent this from occurring by conducting the appropriate access control checks before sending the user to the given location or accessing resource locations provided by the user.	CWE-601

SANS CLOUD SECURITY

Cloud Security and DevSecOps Best Practices

AND

Securing Web Application Technologies (SWAT) Checklist

Version 1.8

Ingraining security into the mind of every developer.

sans.org/cloud-security

APSPS_SEC540_v1.9_1220

SANS CLOUD SECURITY

Curriculum

LONG COURSES

SEC488: Cloud Security Essentials
License to learn cloud security.

SEC510: Public Cloud Security: AWS, Azure, and GCP
Multiple clouds require multiple solutions.

SEC522: Defending Web Applications Security Essentials
Not a matter of "if" but "when." Be prepared for a web attack. We'll teach you how.

SEC540: Cloud Security & DevOps Automation
The cloud moves fast. Automate to keep up.

SEC545: Cloud Security Architecture & Operations
In the cloud, no one canhear you scream. Architect it properly and you won't have to.

SEC557: Continuous Automation for Enterprise and Cloud Compliance
Using Cloud Security and DevOps Tools to Measure Security and Compliance

SEC584: Cloud Native Security: Defending Containers & Kubernetes
Deploy securely at the speed of cloud native.

SEC588: Cloud Penetration Testing
Aim your arrows to the sky and penetrate the cloud.

MGT516: Managing Security Vulnerabilities: Enterprise and Cloud
Stop treating the symptoms. Cure the disease.

SHORT COURSES

SEC534: Secure DevOps: A Practical Introduction
Principles! Practices! Tools! Oh my. Start your journey on the DevSecOps road here.

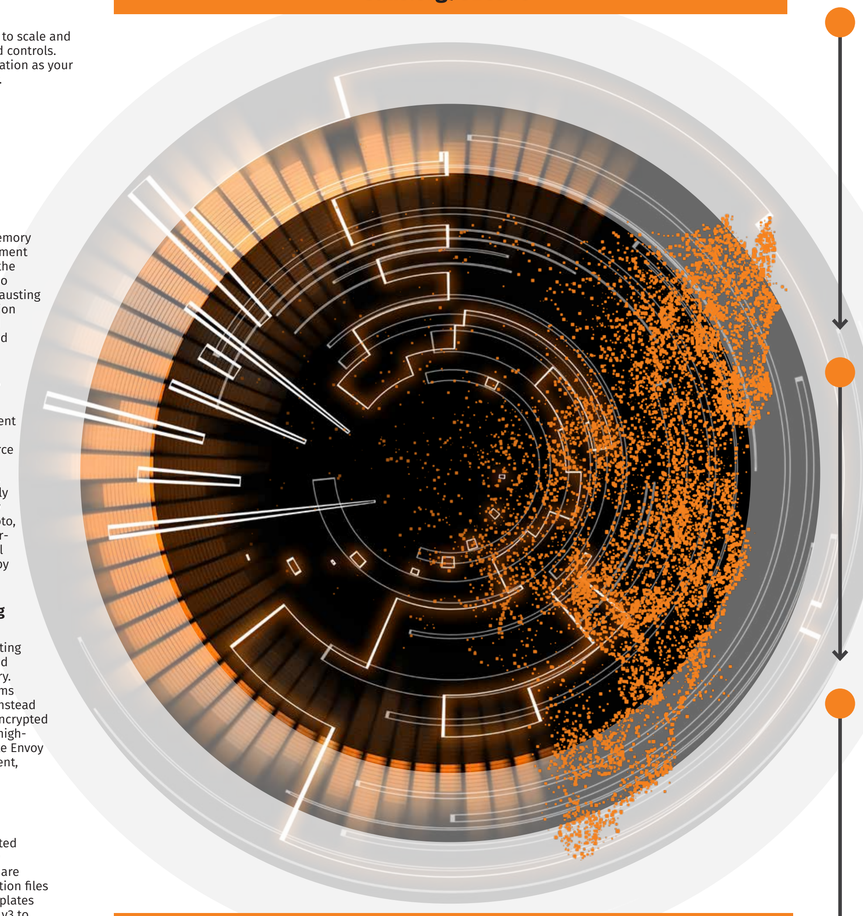
SEC541: Cloud Security Monitoring and Threat Detection
Attackers can run but not hide. Our radar sees all threats.

MGT520: Leading Cloud Security Design & Implementation
Building and leading a cloud security program.

Cloud Security and DevSecOps Best Practices

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

SEC540: Cloud Security and DevOps Automation
sans.org/SEC540



Cloud Security Top 10

1 Insecure Use of Developer Credentials
Developer credentials allow your team and integrations access to your account. They should be stored and used securely to ensure that only authorized individuals and use-cases have access. When possible, consider tracking and auto-expiring credentials after a set period of time or inactivity.

2 Publicly Accessible Storage
Cloud providers have several different methods of storing objects and data. Regularly review your configurations to ensure that only the intended components are publicly accessible.

3 Improper Use of Default Configurations
Cloud providers pre-configure common access control policies. These can be convenient, but often introduce risk as a provider's service offerings change. Pre-configured rules often change to introduce access to new services outside the context of what is actually needed or being used.

4 Broken Access Control
Principles of least privilege should be followed when architecting access to cloud services. Consider the granularity of access to services, systems, and the network. Regularly or automatically review this access to ensure that least privilege is being followed.

5 Misconfigured Network Constructs
Most cloud providers have sophisticated methods to control network access beyond simple IP address-based rules. Consider using these constructs for controlling access at a granular level, and using cloud-provider-based network components to segment traffic thoughtfully.

6 Inadequate Monitoring and Logging
Turn on and regularly monitor API access logging. Consider a risk-based logging strategy for services that are not logged by way of these core logging services.

7 Lack of Inventory Management
API-based access solves a lot of inventory management problems. Consider strategies to enrich your environment with additional information around ownership, use-case, and sensitivity.

8 Domain Hijacking
Transitive-trust often exists between cloud services and DNS entries. Regularly review your DNS and cloud configurations to prevent take-over situations.

9 Lack of a Disaster Recovery Plan
Cloud environments do not automatically solve disaster recovery (DR) concerns. Consider what level of investment is appropriate for catastrophic events within your cloud environment. Design a DR program to recover from outside accounts, providers, or locales.

10 Manual Account Configuration
Doing things by hand limits your ability to scale and leverage cloud-native security tools and controls. Consider "security-as-code" and automation as your best friends within cloud environments.

Top 12 Kubernetes Threats

1 Public-Facing API or etcd Instances
Do not host administrative endpoints on the public Internet and secure networks as one would do with other virtualized or bare metal infrastructure. If creating your own cluster, ensure etcd is deployed separately from the Master nodes and firewalled from the rest of the cluster.

2 Cluster Recency and Certificate Authority Expiration
Provision Kubernetes clusters with new key material or expiration checks on reused certificates and authorities. Replace clusters regularly to keep keys and control plane versions updated. Develop a rigorous update procedure using Blue/Green or Canary clusters to automate deployments. This is necessary because Kubernetes has a major release every three months with minor releases being maintained for nine months.

3 Insecure Workload Configuration
Running privileged containers as the root user or without security contexts increases the risk to that workload and the rest of the cluster. Use Kubernetes Security Policies to enforce Network Policy, Pod Security Policy (or better still OPA) and utilize SecurityContexts on all workloads. These features reduce the likelihood of pivots across the network, enforce the use of security profiles, and prevent the use of the privileged flag, containers running as the root user, and the sharing of host network, process, or IPC namespaces.

4 Improper Namespace Use
Namespaces are a logical grouping for API server entities and should be used to coordinate security and availability features. Many security features are namespace-bound (PodSecurityPolicies, NetworkPolicy) as are denial of service prevention features (LimitRanges, ResourceQuota). Use namespaces first for security and availability features and second for grouping of logical components.

5 Unrestricted and Unaudited Users
RBAC permissions can be difficult to test, so Kubernetes provides a SubjectAccessReview to validate RBAC decisions. Utilize this in a non-production environment to instill confidence that changes to RBAC roles and bindings do not have accidental wider-reaching implications to cluster security. Federate identity to third-parties with 2FA enabled. Enable audit logs, consolidate log data, proactively monitor events, and generate alerts.

6 Configuration Drift
To ensure consistent cluster state, leverage GitOps principles with an in-cluster operator such as Weave Flux that continually re-asserts the state of a cluster or namespace, ensuring that cluster configurations match the state of a Git repository holding YAML configuration files.

7 Cloud Metadata APIs
Cloud instance identity is a point of escalation from Server Side Request Forgeries. In Kubernetes many pods are running on a single node, and if they are able to make calls to the cloud provider's metadata API then they can assume the identity of the underlying host. To remedy this, use workload identity (exchanging the pod's ServiceAccount for a narrowly scoped cloud provider API token) or metadata concealment (blocking the metadata API from the CNI network). For clusters without the required cloud provider support, projects such as Kiam provide an alternative mechanism for workload identity.

8 Externalized Certificate Authorities
Kubernetes operates a zero-trust mutual TLS authentication flow across most components and supports many certificate authorities (CA) that don't have to share the same root of trust. It supports TLS node bootstrapping and key rotation, in addition to projects like Istio adding their own PKI for applications. An enterprise may decide to create CAs from the organization's root of trust, which means there are at least two places that the cluster's network security can be compromised. Instead, self-sign a CA and maintain key material on the cluster to ensure that the only compromise of the key material must happen after the masters have been compromised, at which point the key material is already useless.

9 Resource Exhaustion
Workloads should have their expected ("requests") and maximum ("limits") memory and compute values set in their deployment configuration. This information assists the scheduler in placing workloads, and also prevents monolithic JVM apps from exhausting resources during JIT bytecode compilation startup. These values can be controlled by admission controllers LimitRange and ResourceQuota.

10 Application and Infrastructure Supply Chain
Create reference application development pipelines for your teams that leverage deployment tooling from a trusted source and incorporates static code analysis, container image vulnerability scanning, and dynamic security tooling. Potentially create a cryptographic chain of custody between components with Notary, in-toto, and Git commit signing. Run a container-based IDS solution that detects unusual behavior or network activity exhibited by a running container.

11 Application Developers Writing Security Functionality
Within a multi-tenanted cluster, mandating Encryption in Transit, Authentication and Authorization between Pods is necessary. However, do not have development teams create this functionality on their own. Instead use a service mesh, which is a web of encrypted persistent connections made between high-performance "sidecar" proxy servers like Envoy and Linkerd that adds traffic management, monitoring, and policy – all without microservice changes.

12 Complexity
Maintaining and deploying multi-tenanted Kubernetes clusters can be particularly complex, as within these clusters there are potentially hundreds of YAML configuration files to write and maintain. Create YAML templates with tools such as Kustomize and Helm v3 to simplify configuration management, reduce maintenance burden, and mitigate the risk of unique implementations by giving developers and administrators a secure template.

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 of 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

Shift Security Left

- Start security testing as early in development as possible
- Add self-service security testing into all stages of the pipeline
- Don't slow delivery down: Focus on fast, simple, and clear feedback
- Negotiate windows with DevOps teams for security testing and remediation
  - Maximum duration of security testing feedback delays
  - Maximum lag in fixing vulnerabilities
- Make vulnerability test and fix metrics transparent

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

SANS CLOUD SECURITY Curriculum

LONG COURSES

SEC488: Cloud Security Essentials
License to learn cloud security.

SEC510: Public Cloud Security: AWS, Azure, and GCP
Multiple clouds require multiple solutions.

SEC522: Defending Web Applications Security Essentials
Not a matter of "if" but "when." Be prepared for a web attack. We'll teach you how.

SEC540: Cloud Security and DevOps Automation
The cloud moves fast. Automate to keep up.

SEC545: Cloud Security Architecture and Operations
In the cloud, no one can hear you scream. Architect it properly and you won't have to.

SEC584: Cloud Native Security: Defending Containers & Kubernetes
Deploy securely at the speed of cloud native.

SEC557: Continuous Automation for Enterprise and Cloud Compliance
Using Cloud Security and DevOps Tools to Measure Security and Compliance

SEC588: Cloud Penetration Testing
Aim your arrows to the sky and penetrate the cloud.

MGT516: Managing Security Vulnerabilities: Enterprise and Cloud
Stop treating the symptoms. Cure the disease.

SHORT COURSES

SEC534: Secure DevOps: A Practical Introduction
Principles! Practices! Tools! Oh my. Start your journey on the DevSecOps road here.

SEC541: Cloud Security Monitoring and Threat Detection
Attackers can run but not hide. Our radar sees all threats.

MGT520: Leading Cloud Security Design and Implementation
Building and leading a cloud security program.

sans.org/cloud-security | @SANSCloudSec | linkedin.com/showcase/sanscloudsec | https://youtube.com/c/SANSCloudSecurity

Secure DevOps Toolchain

Pre-Commit
Security activities before code is checked into version control

Threat Modeling/Attack Mapping:
- Attacker personas
- Evil user stories
- Raindance
- Mozilla Rapid Risk Assessment
- OWASP ThreatDragon
- SAFECode Tactical Threat Modeling
- Slack goSDL
- ThreatPlaybook

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

Manual and Peer Reviews:
- CODEOWNERS
- Code Review Description Templates
- Gerrit
- GitHub pull request
- GitLab merge request
- Review Board

Pre-Commit Security Hooks:
- detect-secrets
- git-hound
- git-secrets
- OWASP SEDATED
- pre-commit
- Repo-supervisor
- ThoughtWorks Talisman

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

Secure Coding Standards:
- CERT Secure Coding Standards
- OWASP Proactive Controls
- SAFECode Fundamental Practices for Secure Software Development

Commit (Continuous Integration)
Fast, automated security checks during the build and continuous integration steps

Static Analysis:
- Brakeman
- ESLint
- FindSecurityBugs
- NodeJsScan
- Phan

Kubernetes Security:
- kube-audit
- kube-bench
- kube-hunter
- kubeiscan
- kube-score
- kubesec.io

Infrastructure as Code Analysis:
- ansible-lint
- cfn_nag
- cookstyle
- flint
- Foodcritic
- puppet-lint
- Terrascan

Container Hardening:
- Bane
- CIS Benchmarks
- grsecurity

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

Security Unit Tests:
- JUnit
- Mocha
- xUnit

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

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

Security Testing:
- Arachni | sqlmap | ZAP
- Cloud Container Attack Tool
- ssh_scan | sslyze

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

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

Infrastructure Tests:
- CIS
- Serverspec
- Terratest
- Test Kitchen

Infrastructure Compliance Checks:
- conftest
- HubbleStack
- InSpec
- Open Policy Agent

Vulnerability Management:
- Archerysec
- DefectDojo
- JackHammer

Production (Continuous Deployment)
Security checks before, during, and after code is deployed to production

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

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

Configuration Safety Checks:
- AWS Config
- AWS Trusted Advisor
- Google Cloud Asset Inventory
- Microsoft Azure Advisor
- OSQuery

Cloud Security Testing:
- CloudSploit
- Nimbostratus

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

Serverless Protection:
- FunctionShield

Server Hardening:
- CIS
- dev-sec.io
- SIMP

Host Intrusion Detection System:
- fail2ban
- OSSEC
- Samhain
- Wazuh

Operations
Continuous security monitoring, testing, audit, and compliance checks

Fault Injection:
- Chaos Monkey
- Infection Monkey
- kube-monkey
- pumba

Cyber Simulations:
- Game day exercises
- Tabletop scenarios

Blameless Postmortems:
- Etsy Morgue

Cloud Monitoring:
- AWS Security Hub
- Azure Security Center
- Google Cloud Security Command Center

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

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

Cloud Compliance:
- CIS AWS Benchmark
- CIS Azure Benchmark
- Forseti Security
- Netflix Repokid
- OpenSCAP

Continuous Monitoring:
- alerta
- ElastAlert
- grafana/graphite
- MozDef
- prometheus
- sof-elk
- statsd
- 411

Continuous Scanning:
- Cloud Custodian
- CloudMapper
- Netflix Aardvark
- Prowler
- ScoutSuite
- vuls

Poster contributors:
- Ben Allen
- Will Bengston
- Jim Bird
- David Deatherage
- Mark Geeslin
- Ben Hagen
- Mark Hillick
- Eric Johnson
- Frank Kim
- Jason Lam
- Gregory Leonard
- Andrew Martin
- Dr. Johannes Ullrich
- Thomas Vachon
- Steve Woodrow

SANS