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

SANS CLOUD SECURITY

Nine Key Cloud Security Concentrations

AND

Securing Web Application Technologies

(SWAT) CHECKLIST

The most trusted source of cloud security training, certification, and research.

sans.org/cloud-security

Data Protection

BEST PRACTICE	DESCRIPTION	CWE ID
Use HTTPS everywhere	Use HTTPS for all network data transfer for your application. The benefit of encrypting the data is huge, as it can protect the confidentiality and integrity of the transferred data. HTTPS is a pre-requisite for HTTP2 and HTTP3 protocol which offers better security and performance amongst other benefits. **EXAMPLE:** sslstrip	CWE-311 CWE-319 CWE-523
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, testssl.sh, SSLyze, sslscan	
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
Storing key material securely by using key management services	When keys or credentials are stored in your system they must be properly secured and only accessible to the appropriate staff on a need-to-know basis. The modern solution is to leverage a secret/key management solution. **EXAMPLE:** Hardware Security Modules (HSM), AWS KMS, Azure Key Vault, GCP Cloud Key Management	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**	CWE-324
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 and/or threat model	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 **https://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 corresponsing password is incorrect confirm to an attacker that the account does exist on the system.	
Use secret management solution to store API keys/credentials	Modern web apps often require network resource access, necessitating authentication through application-provided credentials. Safely storing these credentials is a major challenge, as embedding them in the app's code is a well-known security risk. Secrets management solutions address this issue by allowing apps to request credentials on-demand, without the need for storing them on disk. **EXAMPLE:** AWS Secrets Manager, Hashicorp Vault	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
Use Content Security Policy	Use the Content-Security-Policy header with configured security policy to enhance the security of the application. A properly configured policy can mitigate or reduce the risk of multiple very common exploited web security flaws such as Cross Site Scripting and Clickjacking.	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
Perform access control on static resources	Ensure that static application resources are incorporated into the access control system, this includes cloud based static resources. Use the same access control logic on the static resources where possible.	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

CSPS_SEC540_v2.4_10-23

NINE KEY CLOUD SECURITY CONCENTRATIONS

DevSecOps

CAPABILITY: CI/CD Security

AWS	AZURE	GCP	3RD PARTY	OPEN SOURCE
CodeBuild CodePipeline	Azure DevOps Azure Source Code Repositories Azure Branch Policies Azure Key Vault Secrets	Cloud Build	GitHub Branch Protections GitHub Actions GitHub Secrets GitHub Advanced Security GitLab Branch Protections GitLab Auto DevOps GitLab Vault Secrets	gitsecrets trufflehog gitleaks

CAPABILITY: Infrastructure as Code

AWS	AZURE	GCP	3RD PARTY
AWS CloudFormation	Azure Resource Manager BICEP	Cloud Resource Manager	HashiCorp Terraform

CAPABILITY: Configuration Management

AWS	AZURE	GCP	3RD PARTY		OPEN SOURCE
AWS OpsWorks	Configuration Manager on Azure Azure Automation Azure App Configuration	GCP – Anthos Config Management	Ansible Chef Puppet	Saltstack Vagrant Packer	inspec OPA

CAPABILITY: Content Delivery and Protection

AWS	AZURE	GCP	3RD PARTY	
AWS CloudFront	Azure Content Delivery Network (CDN)	GCP Cloud CDN	Akamai Cloudflare	Fastly

CAPABILITY: Microservice Security

AWS	AZURE	GCP	OPEN SOURCE
API Gateway App Mesh	API Management Open Service Mesh	API Gateway GKE Anthos Service Mesh	OPA

Infrastructure Security

CAPABILITY: Cloud Security Posture Management

AZURE	3RD PARTY	OPEN SOURCE
Microsoft Defender for Cloud	Palo Alto Prisma Cloud Palo Alto – Prisma CSPM Orca Wiz	Cloud Custodian GCP Config Validator GCP Terraform Validator Prowler

CAPABILITY: Network Security

AWS	AZURE	GCP
Virtual Private Cloud AWS Firewall Manager Network Access Control Lists Security Groups	Virtual Network Azure Firewall Manager Network Security Groups	Virtual Private Cloud GCP Firewall

CAPABILITY: Private Connectivity

AWS	AZURE	GCP
AWS Direct Connect AWS PrivateLink	Azure Express Route Azure Private Link	Cloud InterConnect Private Service Connect

CAPABILITY: Private Endpoints

AWS	AZURE	GCP
VPC Endpoints	Network Service Endpoints	VPC Service Controls

CAPABILITY: Remote Access

AWS	AZURE	GCP
Session Manager VPN	Serial Console VPN Gateway	Cloud SSH Cloud VPN

CAPABILITY: DDoS Protection

AWS	AZURE	GCP
AWS Shield	Azure DDoS Protection	Cloud Armor

SANS
CLOUD SECURITY

sans.org/cloud-security
@SANSCloudSec
linkedin.com/showcase/sanscloudsec
sansurl.com/cloud-discord

Identity and Access Management

CAPABILITY: Groups, Roles, Policies, Permissions

AWS	AZURE	GCP
AWS IAM	Microsoft Entra ID	Cloud IAM Resource Manager

CAPABILITY: Single Sign On (SSO)

AWS	AZURE	GCP	3RD PARTY
IAM Identity Center	Microsoft Entra ID	Cloud Identity	Okta Ping Sailpoint Oracle IDCS

CAPABILITY: Service Accounts

AWS	AZURE	GCP
Instance Profile	Managed Identity Service Principals User-Based Service Accounts	Service Account

CAPABILITY: Customer IAM (CIAM)

AWS	AZURE	GCP
Amazon Cognito User Pools	Entra ID for Customers	Cloud Identity for Customers and Partners Firebase Authentication Managed Service for Microsoft Active Directory

CAPABILITY: Least Privilege

AWS	AZURE	GCP
IAM Access Analyzer	Azure RBAC	Cloud IAM

Security Architecture

CAPABILITY: Best Practices

AWS	AZURE	GCP
AWS Well-Architected Framework	Azure Well-Architected Framework	Google Cloud Architecture Framework

CAPABILITY: Account Management

AWS	AZURE	GCP
AWS Organizations Service Control Policies Azure Bastion	Azure Tenants Subscriptions Azure Policy	Google Cloud Folders and Projects Organizational Policies

CAPABILITY: Hybrid Cloud

AWS	AZURE	OPEN SOURCE
AWS Outposts	Azure Arc	OpenStack

Threat Detection and Response

CAPABILITY: Logging & Monitoring

AWS	AZURE	GCP	OPEN SOURCE
AWS CloudTrail Amazon CloudWatch	Azure Monitor Azure Log Analytics Workspace Microsoft Sentinel	GCP StackDriver	Logstash OpenTelemetry fluentbit/fluentd

CAPABILITY: SIEM

AWS	AZURE	GCP	3RD PARTY	OPEN SOURCE
AWS SecurityHub	Microsoft Sentinel	Google Chronicle	Splunk Exabeam LogRhythm	AlienVault OSSIM SIEM Monster Wazuh

CAPABILITY: Network Flow

AWS	AZURE	GCP
VPC Flow Logs	NSG Flow Logs	VPC Flow Logs Firewall Logging

CAPABILITY: Threat Monitoring

AWS	AZURE	GCP
GuardDuty Detective	Microsoft Defender Microsoft Sentinel	Security Command Center

CAPABILITY: Vulnerability Scanning

AWS	AZURE	GCP
Amazon Inspector	Microsoft Defender for Cloud	Web Security Scanner

CAPABILITY: Security Cockpit

AWS	AZURE	GCP
AWS Security Hub	Microsoft Defender for Cloud	Security Command Center

CAPABILITY: Security Orchestration Automation & Response (SOAR)

AZURE	3RD PARTY
Microsoft Sentinel	Cortex XSOAR

Governance

CAPABILITY: Best Practices

AWS	AZURE	GCP
AWS Cloud Adoption Framework	Azure Cloud Adoption Framework	GCP Cloud Adoption Framework

CAPABILITY: Oversight

AWS	AZURE	GCP
Security Hub	Microsoft Defender for Cloud	Security Command Center

CAPABILITY: Compliance

AWS	AZURE	GCP	3RD PARTY
AWS Audit Manager AWS Artifact AWS Config	Microsoft Defender for Cloud	Security Command Center	CIS Benchmarks

CAPABILITY: Asset Inventory

AWS
AWS Resource Explorer

CAPABILITY: Data Residency

Data Protection

CAPABILITY: Key Management

AWS	AZURE	GCP
Key Management Service (KMS) Cloud HSM	Azure Key Vault Azure Dedicated HSM	Cloud Key Management Service (KMS) Google Cloud HSM

CAPABILITY: Secrets Management

AWS	AZURE	GCP	3RD PARTY
AWS Secrets Manager Parameter Store	Azure Key Vault	Secret Manager	HashiCorp Vault

CAPABILITY: Encryption at Rest

AWS	AZURE	GCP
AWS KMS	Azure Key Vault	Google Cloud KMS

CAPABILITY: Encryption in Transit

AWS	AZURE	GCP
AWS Certificate Manager	Azure Key Vault	Certificate Manager

CAPABILITY: Certificate Management

AWS	AZURE	GCP
AWS Certificate Manager	Azure Key Vault	Certificate Authority Service Certificate Manager

CAPABILITY: Data Loss Prevention

AWS	AZURE	GCP
Amazon Macie	Microsoft Purview	Cloud Data Loss Prevention

CAPABILITY: Cloud Access Security Broker

AZURE
Microsoft Defender for Cloud Apps

CAPABILITY: Data Backup, Restore and Recovery

AWS	AZURE	GCP
AWS Backup	Azure Backup	GCP Backup & Disaster Recovery

Compute Security

CAPABILITY: Cloud Workload Protection Platform

AZURE	3RD PARTY
Defender for Cloud	Aqua Security Palo Alto Prisma Cloud

CAPABILITY: Container Orchestration

AWS	AZURE	GCP	3RD PARTY	OPEN SOURCE
AWS Elastic Container Service AWS Elastic Kubernetes Service AWS Fargate	Azure Kubernetes Service	Google Kubernetes Engine	Red Hat OpenShift	Kubernetes

CAPABILITY: Serverless Security

AWS	AZURE	GCP	OPEN SOURCE
AWS Lambda	Azure Functions	Cloud Functions	Security-Guard

CAPABILITY: Container Registry (CR)

AWS	AZURE	GCP	3RD PARTY
Amazon Elastic CR	Azure CR	GCP CR	DockerHub GitHub CR Gitlab CR

Application Security

CAPABILITY: Static Application Security Testing

>>> 3RD PARTY	OPEN SOURCE
Veracode Synopsys Checkmarx GitHub Advanced Security	semgrep Betterscan Horusec Automated Security Helper

CAPABILITY: Software Composition Analysis

>>> 3RD PARTY	OPEN SOURCE
Veracode Synopsys Checkmarx GitHub SCA Snyk	OWASP Dependency-Check retire.jds OSS Review Toolkit

CAPABILITY: Dynamic Application Security Testing

>>> 3RD PARTY	OPEN SOURCE
Veracode Synopsys Acunetix	ZAP w3af Nuclei

CAPABILITY: Web Application Firewall

AWS	AZURE	GCP	3RD PARTY	OPEN SOURCE
AWS WAF	Azure WAF	Cloud WAF	Akamai Cloudflare	ModSecurity Imperva

CAPABILITY: Web App and API Protection

AWS	AZURE	GCP	3RD PARTY	OPEN SOURCE
API Gateway	API Management Application Gateway	Web App API Protection	Akamai Cloudflare Imperva NoName Salt Security	Curiefence

CAPABILITY: Runtime Application Self-Protection

>>> 3RD PARTY	OPEN SOURCE
Imperva Signal Sciences Contrast Security	OpenRASP

CAPABILITY: Application Security Posture Management

>>> 3RD PARTY
Crowdstrike Snyk Synopsys

Supporting your journey to becoming a #SANSCloudAce