Contenido

[A01:2021 – Broken Access Control 5.1 Management direction for information security 2](#_Toc152610698)

[Avoid: 3](#_Toc152610699)

[Suggestions: 3](#_Toc152610700)

[Decision: 3](#_Toc152610701)

[A02:2021 – Cryptographic Failures 3](#_Toc152610702)

[Avoid: 4](#_Toc152610703)

[Suggestions: 4](#_Toc152610704)

[Decision: 5](#_Toc152610705)

[A03:2021 – Injection 5](#_Toc152610706)

[Avoid: 5](#_Toc152610707)

[Suggestions: 5](#_Toc152610708)

[Decision: 6](#_Toc152610709)

[A04:2021 – Insecure Design 6](#_Toc152610710)

[Suggestions: 6](#_Toc152610711)

[Decision: 6](#_Toc152610712)

[A05:2021 – Security Misconfiguration 6](#_Toc152610713)

[Avoid: 6](#_Toc152610714)

[Suggestions: 7](#_Toc152610715)

[Decision: 7](#_Toc152610716)

[A06:2021 – Vulnerable and Outdated Components 7](#_Toc152610717)

[Concerns: 7](#_Toc152610718)

[Suggestions: 8](#_Toc152610719)

[Decision: 8](#_Toc152610720)

[A07:2021 – Identification and Authentication Failures 8](#_Toc152610721)

[Avoid: 8](#_Toc152610722)

[Suggestions: 8](#_Toc152610723)

[Decision: 9](#_Toc152610724)

[A08:2021 – Software and Data Integrity Failures 9](#_Toc152610725)

[Concerns: 9](#_Toc152610726)

[Suggestions: 9](#_Toc152610727)

[Decision: 10](#_Toc152610728)

[A09:2021 – Security Logging and Monitoring Failures 10](#_Toc152610729)

[Avoid: 10](#_Toc152610730)

[Suggestions: 10](#_Toc152610731)

[Decision: 10](#_Toc152610732)

[A10:2021 – Server-Side Request Forgery (SSRF) 10](#_Toc152610733)

[Suggestions: 11](#_Toc152610734)

[Decision: 11](#_Toc152610735)

# A01:2021 – Broken Access Control 5.1 Management direction for information security

Notable Common Weakness Enumerations (CWEs) included are CWE-200: Exposure of Sensitive Information to an Unauthorized Actor, CWE-201: Insertion of Sensitive Information Into Sent Data, and CWE-352: Cross-Site Request Forgery.

## Avoid:

* Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
* Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
* Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
* Accessing API with missing access controls for POST, PUT and DELETE.
* Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.
* Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
* Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

## Suggestions:

* Except for public resources, deny by default.
* Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
* Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
* Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
* Log access control failures, alert admins when appropriate (e.g., repeated failures).
* Rate limit API and controller access to minimize the harm from automated attack tooling.
* Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

## Decision:

It applies

# A02:2021 – Cryptographic Failures

The focus is on failures related to cryptography (or lack thereof). Which often lead to exposure of sensitive data. Notable Common Weakness Enumerations (CWEs) included are CWE-259: Use of Hard-coded Password, CWE-327: Broken or Risky Crypto Algorithm, and CWE-331 Insufficient Entropy.

## Avoid:

* Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, FTP also using TLS upgrades like STARTTLS. External internet traffic is hazardous. Verify all internal traffic, e.g., between load balancers, web servers, or back-end systems.
* Are any old or weak cryptographic algorithms or protocols used either by default or in older code?
* Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing? Are crypto keys checked into source code repositories?
* Is encryption not enforced, e.g., are any HTTP headers (browser) security directives or headers missing?
* Is the received server certificate and the trust chain properly validated?
* Are initialization vectors ignored, reused, or not generated sufficiently secure for the cryptographic mode of operation? Is an insecure mode of operation such as ECB in use? Is encryption used when authenticated encryption is more appropriate?
* Are passwords being used as cryptographic keys in absence of a password base key derivation function?
* Is randomness used for cryptographic purposes that was not designed to meet cryptographic requirements? Even if the correct function is chosen, does it need to be seeded by the developer, and if not, has the developer over-written the strong seeding functionality built into it with a seed that lacks sufficient entropy/unpredictability?
* Are deprecated hash functions such as MD5 or SHA1 in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?
* Are deprecated cryptographic padding methods such as PKCS number 1 v1.5 in use?
* Are cryptographic error messages or side channel information exploitable, for example in the form of padding oracle attacks?

## Suggestions:

* Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
* Make sure to encrypt all sensitive data at rest.
* Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
* Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
* Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
* Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
* Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5 .

## Decision:

It applies

# A03:2021 – Injection

Notable Common Weakness Enumerations (CWEs) included are CWE-79: Cross-site Scripting, CWE-89: SQL Injection, and CWE-73: External Control of File Name or Path. Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections. Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged. Organizations can include static (SAST), dynamic (DAST), and interactive (IAST) application security testing tools into the CI/CD pipeline to identify introduced injection flaws before production deployment.

## Avoid:

* User-supplied data is not validated, filtered, or sanitized by the application.
* Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
* Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
* Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

## Suggestions:

* Use positive server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
* For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

Note: SQL structures such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.

* Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

## Decision:

It applies

# A04:2021 – Insecure Design

Focuses on risks related to design and architectural flaws, with a call for more use of threat modeling, secure design patterns, and reference architectures. Notable Common Weakness Enumerations (CWEs) include CWE-209: Generation of Error Message Containing Sensitive Information, CWE-256: Unprotected Storage of Credentials, CWE-501: Trust Boundary Violation, and CWE-522: Insufficiently Protected Credentials.

## Suggestions:

* Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
* Segregate tier layers on the system and network layers depending on the exposure and protection needs
* Segregate tenants robustly by design throughout all tiers
* Limit resource consumption by user or service.

## Decision:

It applies

# A05:2021 – Security Misconfiguration

Applications must be tested for some form of misconfiguration.

## Avoid:

* Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
* Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
* Default accounts and their passwords are still enabled and unchanged.
* Error handling reveals stack traces or other overly informative error messages to users.
* For upgraded systems, the latest security features are disabled or not configured securely.
* The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
* The server does not send security headers or directives, or they are not set to secure values.
* The software is out of date or vulnerable

## Suggestions:

* A repeatable hardening process makes it fast and easy to deploy another environment that is appropriately locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to set up a new secure environment.
* A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
* A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see A06:2021-Vulnerable and Outdated Components). Review cloud storage permissions (e.g., S3 bucket permissions).
* A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
* Sending security directives to clients, e.g., Security Headers.
* An automated process to verify the effectiveness of the configurations and settings in all environments

## Decision:

It applies

# A06:2021 – Vulnerable and Outdated Components

Notable CWEs included are CWE-1104: Use of Unmaintained Third-Party Components and the two CWEs from Top 10 2013 and 2017. Vulnerable Components are a known issue that we struggle to test and assess risk.

## Concerns:

* If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
* If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
* If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
* If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
* If software developers do not test the compatibility of updated, upgraded, or patched libraries.
* If you do not secure the components’ configurations

## Suggestions:

* Remove unused dependencies, unnecessary features, components, files, and documentation.
* Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, OWASP Dependency Check, retire.js, etc. Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
* Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component (See A08:2021-Software and Data Integrity Failures).
* Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

## Decision:

It applies

# A07:2021 – Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. Notable CWEs included are CWE-297: Improper Validation of Certificate with Host Mismatch, CWE-287: Improper Authentication, and CWE-384: Session Fixation

## Avoid:

* Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
* Permits brute force or other automated attacks.
* Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
* Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
* Uses plain text, encrypted, or weakly hashed passwords data stores (see A02:2021-Cryptographic Failures).
* Has missing or ineffective multi-factor authentication.
* Exposes session identifier in the URL.
* Reuse session identifier after successful login.
* Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

## Suggestions:

* Where possible, implement multi-factor authentication to prevent automated credential stuffing, brute force, and stolen credential reuse attacks.
* Do not ship or deploy with any default credentials, particularly for admin users.
* Implement weak password checks, such as testing new or changed passwords against the top 10,000 worst passwords list.
* Align password length, complexity, and rotation policies with National Institute of Standards and Technology (NIST) 800-63b's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.
* Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
* Limit or increasingly delay failed login attempts, but be careful not to create a denial-of-service scenario. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
* Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.

## Decision:

It applies

# A08:2021 – Software and Data Integrity Failures

Focuses on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. Notable Common Weakness Enumerations (CWEs) include CWE-829: Inclusion of Functionality from Untrusted Control Sphere, CWE-494: Download of Code Without Integrity Check, and CWE-502: Deserialization of Untrusted Data.

## Concerns:

Attackers could potentially upload their own updates to be distributed and run on all installations.

## Suggestions:

* Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
* Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories. If you have a higher risk profile, consider hosting an internal known-good repository that's vetted.
* Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities
* Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline.
* Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.
* Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data

## Decision:

It applies

# A09:2021 – Security Logging and Monitoring Failures

It can be very impactful for accountability, visibility, incident alerting, and forensics. This category expands beyond CWE-778 Insufficient Logging to include CWE-117 Improper Output Neutralization for Logs, CWE-223 Omission of Security-relevant Information, and CWE-532 Insertion of Sensitive Information into Log File. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time

## Avoid:

* Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
* Warnings and errors generate no, inadequate, or unclear log messages.
* Logs of applications and APIs are not monitored for suspicious activity.
* Logs are only stored locally.
* Appropriate alerting thresholds and response escalation processes are not in place or effective.
* Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
* The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

## Suggestions:

* Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
* Ensure that logs are generated in a format that log management solutions can easily consume.
* Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.

## Decision:

It applies

# A10:2021 – Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures

## Suggestions:

1. Implement From Network layer for:

* Segment remote resource access functionality in separate networks to reduce the impact of SSRF
* Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.

Hints:

~ Establish an ownership and a lifecycle for firewall rules based on applications.

~ Log all accepted and blocked network flows on firewalls

1. Implement From Application layer for:

* Sanitize and validate all client-supplied input data
* Enforce the URL schema, port, and destination with a positive allow list
* Do not send raw responses to clients
* Disable HTTP redirections
* Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions

1. Do not mitigate SSRF via the use of a deny list or regular expression. Attackers have payload lists, tools, and skills to bypass deny lists

## Decision:

It applies