**OWASP Top Ten (Core)**

**Learning Objectives:**

* Research current security vulnerabilities.
* Determine the most appropriate mitigation to a security vulnerability.

Research the OWASP top 10 that are current. After you have this, research the impact of each of the 10 vulnerabilities. Make sure to use APA citations for your sources (yes including the OWASP website). Each vulnerability should have **at least 1 paragraph** explaining the impacts and how the vulnerability can be mitigated.

I logged on unto the “ <https://owasp.org/>” website.

A screenshot of a computer

Description automatically generated

I did a search for the “**top 10**” in the search field.

A screenshot of a computer

Description automatically generated

I chose the first item “[[**OWASP Top 10:2021**](https://owasp.org/Top10/)](https://owasp.org/www-project-top-ten/)”

A screenshot of a computer

Description automatically generated

## This is the most up-to-date Top 10

A screenshot of a computer

Description automatically generated

I am going to break down all top 10 web application risks of each one by one. This breakdown will include the following: (**Short summary, Impact, and Mitigation**)

1. **Broken Access Control**

Broken Access Control is a security vulnerability that occurs when a system or application fails to properly enforce restrictions on who can access certain resources or perform specific actions. It occurs when users can access unauthorized areas, manipulate data, or execute functions they shouldn't have permission for. This can lead to data breaches, unauthorized actions, and compromised system integrity.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Violation of the principle of least privilege or denial 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 to modify 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.

CORS misconfiguration allows API access from unauthorized/untrusted origins.

Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

**Mitigation**

**These are the listed mitigation I was able to find:**

Enforce least privileges: Assign users the minimum privileges needed to complete their function.

Deny by default: For security purposes, even when no access control rules are explicitly matched, an application should be configured to deny access by default.

Validate permissions on every request: Correctly validate permissions on every request, including those initiated by AJAX script, server-side, or any other source.

Take time to thoroughly review the authorization logic of chosen tools and technology and implement custom logic when necessary. Test configurations all configurations.

Prefer feature and attribute-based controls over role-based.

Ensure lookup IDs cannot be accessible (even when guessed) and cannot be tampered with.

Ensure that static resources are authorized and incorporated into access control policies.

Authorized checks should be performed at the right location. Never rely on client-side access control checks.

Exit safely when authorization checks fail.

Unit and integration test authorization logic.

Article link: https://hackernoon.com/what-is-broken-access-control-and-why-should-you-care

A computer screen shot of a computer

Description automatically generated

1. **Cryptographic Failures**

A cryptographic failure is a critical web application security vulnerability that exposes sensitive application data on a weak or non-existent cryptographic algorithm. Those can be passwords, patient health records, business secrets, credit card information, email addresses, or other personal user information.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Transmitting secret data in plain text

Use of old/less-secure algorithm

Use of a hard-coded password in config files

Improper cryptographic key management

Insufficient randomness for cryptographic functions

Missing encryption

Insecure implementation of certificate validation

Use of deprecated hash functions

Use of outdated padding methods

Presence of sensitive data in source control

Use of insecure initialization vectors

Use of passwords as crypto keys without a password-based key derivation function

Exploitable side-channel information or cryptographic error messages

**Mitigation**

**These are the listed mitigation I was able to find:**

Discard Unused Data

Disable Caching for Responses with Sensitive Data

Use Appropriate Initialization Vectors

Use Updated and Established Cryptographic Functions, Algorithms, and Protocols

Enforce Key Rotation

Use Authenticated Encryption Instead of Plain Encryption

Article link: <https://crashtest-security.com/owasp-cryptographic-failures/>

A screenshot of a computer

Description automatically generated

1. **Injection**

Injection is an attacker's attempt to send data to an application in a way that will change the meaning of commands being sent to an interpreter. For example, the most common example is SQL injection, where an attacker sends “101 OR 1=1” instead of just “101”

**Impact**

**These are the listed vulnerabilities I was able to find:**

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.

**Mitigation**

**These are the listed mitigation I was able to find:**

The preferred option is to use a safe API, which avoids using the interpreter entirely, provides a parameterized interface, or migrates to Object Relational Mapping Tools (ORMs).

Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data or executes hostile data with EXECUTE IMMEDIATE or exec ().

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.

Article link: <https://owasp.org/Top10/A03_2021-Injection/>

A screenshot of a computer

Description automatically generated

1. **Insecure Design**

At its core, insecure design is the lack of security controls being integrated into the application throughout the development cycle. This can have wide ranging and deep-rooted security consequences as the application itself is not designed with security in mind.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Missing user input bounds can lead to issues like buffer overflows.

Usage of unsafe APIs or functions can result in compromise: think of using random numbers without any seed for instance or extracting an archive without taking into consideration the absolute or relative paths that the embedded files can have.

Applications making use of elevated privileges than required.

**Mitigation**

**These are the listed mitigation I was able to find:**

Follow best practices, wherever possible

Build things with a security-centric mindset.

Secure design principles must be followed and adhered to for the lifetime of the application/services

Code reviews must be done thoroughly to avoid any bad code going into the production

Regular code audits and pentests are a good way to ensure the security of your products. It’s a general advice that would help detect and mitigate most of the trivial issues.

Article link: <https://medium.com/@shivam_bathla/a04-2021-insecure-design-9e16449f29ef>

A screenshot of a computer

Description automatically generated

1. **Security Misconfiguration**

A security misconfiguration is when security options are not defined in a way that maximizes security, or when services are deployed with insecure default settings. This can happen in any computing system, software application, as well as in cloud and network infrastructure.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Default accounts / passwords are enabled—Using vendor-supplied defaults for system accounts and passwords is a common security misconfiguration and may allow attackers to gain unauthorized access to the system.

Secure password policy is not implemented Failure to implement a password policy may allow attackers to gain unauthorized access to the system by methods such as using lists of common username and passwords to brute force a username and/or password field until successful authentication.

Software is out of date and flaws are unpatched—Failure to update software patches as part of the software management process may allow attackers to use techniques such as code injection to inject malicious code that the application then executes.

Files and directories are unprotected—Leaving files and directories unprotected may allow attackers to use techniques such as forceful browsing to gain access to restricted files or areas in the server directory.

Unused features are enabled or installed Failure to remove unnecessary features, components, documentation, and samples makes the application susceptible to misconfiguration vulnerabilities, and may allow attackers to use techniques such as code injection to inject malicious code that the application then executes.

Security features not maintained or configured properly—Failure to properly configure and maintain security features makes the application vulnerable to misconfiguration attacks.

Unpublished URLs are not blocked from receiving traffic from ordinary users—Unpublished URLs, accessed by those who maintain applications, are not intended to receive traffic from ordinary users. Failure to block these URLs can pose a significant risk when attackers scan for them.

Improper / poor application coding practices—Improper coding practices can lead to security misconfiguration attacks. For example, the lack of proper input/output data validation may lead to code injection attacks which work by injecting code that the application executes.

Directory traversal—allows an attacker to access directories, files, and commands that are outside the root directory. Armed with access to application source code or configuration and critical system files, a cybercriminal can change a URL in such a way that the application could execute or display the contents of arbitrary files on the server. Any device or application that reveals an HTTP-based interface is possibly vulnerable to a directory traversal attack.

**Mitigation**

**These are the listed mitigation I was able to find:**

Establish a hardening process that is repeatable, so that it’s fast and simple to deploy correctly configured new environments. The production, development, and QA environments must all be configured in the same way, but with distinct passwords used in every environment. Automate this process to easily establish a secure environment.

Install patches and software updates regularly and in a timely way in every environment. You can also patch a golden image and deploy the image into your environment.

Develop an application architecture that offers effective and secure separation of elements.

Run scans and audits often and periodically to identify missing patches or potential security misconfigurations.

Ensure a well-maintained and structured development cycle. This will facilitate the security testing of the application in the development phase.

Train and educate your employees on the significance of security configurations and how they can affect the general organization’s security.

Encrypt data-at-rest to prevent data from exploitation.

Apply genuine access controls to both files and directories. This will help offset the vulnerabilities of files and directories that are unprotected.

If using custom code, utilize a static code security scanner before you integrate the code into the production environment. Security professionals must also perform manual reviews and dynamic testing.

Utilize a minimal platform free from excess features, documentation, samples and components. Don’t install or remove unused features or insecure frameworks.

Review cloud storage permissions, including S3 bucket permissions. Incorporate updates and reviews of all security configurations for all updates, security patches and notes into your patch management process.

Put in place an automated process. This makes certain that security configurations are applied to all environments.

Article link: https://brightsec.com/blog/security-misconfiguration/

A screenshot of a computer screen

Description automatically generated

1. **Vulnerable and Outdated Components**

Vulnerable and outdated components refer to third-party libraries or frameworks used in web applications that have known vulnerabilities or are no longer supported by their developers. These components can be exploited by attackers to gain unauthorized access to sensitive data or take control of the system.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Vulnerable components (OS or software packages, applications, runtime environments) in the client and server-side code.

Insecure software configuration

Old/unpatched dependencies in the dependency chain of the components being used.

**Mitigation**

**These are the listed mitigation I was able to find:**

Maintain an inventory of components you are using and ensure that they are kept up to date.

Remove unused dependencies and components to reduce the attack surface and your liabilities (yes, code is indeed a great liability!)

Install the components via trusted channels and make sure to validate their integrity. Also, it’s better to use signed packages (if available).

Be on the lookout for any security patches for the components you are relying on. If the packages you use are not maintained, then either make sure you apply patches yourself or use some alternate component that is well maintained and has a big user-base and supporting community. If possible, then this should actually be done right from the start — choose your dependencies and components wisely!

Article link: <https://medium.com/@shivam_bathla/a06-2021-vulnerable-and-outdated-components-a5d96017049c>

A screenshot of a computer

Description automatically generated

1. **Identification and Authentication Failures**

While identification is the ability to identify a user uniquely, authentication is the ability of an application to prove the authenticity of the user as they claim to be. The failure of a system to identify and/or authenticate leaves the application susceptible to attacks and leaves user accounts/data at risk.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Credential Stuffing

Brute Force Attacks

Session Identifier Exposed in the URL

Session Fixation

No validation of weak passwords

Weak credential recovery and forgot password processes

Using plain text or weakly hashed passwords data stores.

Ineffective multi-factor authentication

**Mitigation**

**These are the listed mitigation I was able to find:**

Requiring Strong Passwords

Securing Password Data Stores

Personal Password Vaults

Securing the Logout/Login Function

Securing Password Resets

Multi-Factor Authentication

Preventing User Enumeration

CAPTCHA

Multi-Step Login Processes

Article link: <https://cyolo.io/blog/identification-and-authentication-failures-and-how-to-prevent-them>

A black rectangular object with a red lock on it

Description automatically generated

1. **Software and Data Integrity Failures**

Software and data integrity failures occur when an attacker can modify or delete data in an unauthorized manner. This can happen due to vulnerabilities in the software or poor coding practices. Attackers can exploit these vulnerabilities to gain access to sensitive information or cause damage to the system.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Faulty assumptions of the server-side and client-side components in use

Outdated or unsupported third-party software

Insufficient vulnerability scanning

Erroneous input validation across the pipeline

Missing framework/platform patches

Missing unit tests

Insecure component configurations

**Mitigation**

**These are the listed mitigation I was able to find:**

Use Digital Signatures for Software Component Verification

Use Secure Repositories for Libraries and Dependencies

Enforce Secure Code Reviews

Implement a Software Supply Chain Security Program

Deserialization of Untrusted Data

Article link: <https://crashtest-security.com/owasp-software-data-integrity-failures/>

A screenshot of a computer

Description automatically generated

1. **Security Logging and Monitoring Failures**

Logging and monitoring will help you to identify patterns of activity on your networks, which in turn provide indicators of compromise. In the event of incidents, logging data can help to more effectively identify the source and the extent of compromise.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Login and failed attempts not being logged.

Logs not backed up or being stored locally.

Improper logs that do not provide any valuable information.

Lack of monitoring systems in real time

Missing monitoring and alerting systems

Logs not protected for integrity.

**Mitigation**

**These are the listed mitigation I was able to find:**

Ensure login, access control, and server-side input validation is logged.

Ensure logs contain enough context to identify suspicious behavior and enable in-depth forensic analysis.

Take measures to prevent attackers from tampering with log data

Where should I start?

Determine which logs to generate and backup.

Decide how to retain logs by looking at the log sources and seeing how these should be properly collected, stored, and secured.

Implement log storage and tooling for analysis. If you are using a central solution, you need to think about the large volumes of traffic moving across your network and ensure it’s transported securely.

Have the correct tools installed to perform analysis.

Validate that logging is working as expected. If you make changes within your network or to infrastructure, think about how those logs are connected. As a precaution the logging strategy should be reviewed periodically as a precaution.

Article link: <https://foresite.com/blog/owasp-top-ten-9-security-logging-and-monitoring-failures/>

A screenshot of a computer

Description automatically generated

1. **Server-Side Request Forgery**

A Server-Side Request Forgery (SSRF) attack involves an attacker abusing server functionality to access or modify resources. The attacker targets an application that supports data imports from URLs or allows them to read data from URLs.

**Impact**

**These are the listed vulnerabilities I was able to find:**

Attackers can use a compromised server as an IP scanner or port to gather the information from front and back-end systems.

If internal protocols like Gopher are enabled on targeted servers, then hackers can easily make a path for further reconnaissance possibilities.

A successful SSRF cyber-attack permits threat actors to find the hidden IP addresses that are running at the back of reverse proxy.

By creating a possibility for XCE, SSRF creates a base for multiple cyber troubles as malware, viruses, ransomware, and other sorts of ill-intended components can be introduced.

**Mitigation**

**These are the listed mitigation I was able to find:**

Response handling

To make sure the server-side or client-side response, against a request, isn’t approachable by any unauthorized user, request-response handling measures should be adopted. For instance, under no circumstances, an unfamiliar response or a raw response body should be processed further.

Whitelists and DNS resolution

A better alternative to blacklisting, IP and DNS name whitelist works well as an SSRF preventive solution as it makes the application-accessible IP address/DNS names inaccessible. Also, it’s crucial to perform user input authentication wisely. Actions like not giving access to a private IP address, as it is non-routable, must be taken.

Disable unused URL schemas.

Always follow the HTTPS or HTTP URL schemas for request initiation and dispose of all the other redundant URL schemas and reduce the scope of SSRF attacks. The more URL schemas you follow or store, the higher the opportunities for the hacker to carry out an attack.

Keeping proper access-restrictions for internal services

In case you’re using services like MongoDB, Memcached, Redis, and MongoDB, it’s essential to enforce solid authentication. As these services lack default authentication, hackers can use them as a means to conduct the SSRF attack. When strong authentication measures are implemented on such internal services, they become safe to use.

Article link: <https://www.wallarm.com/what/server-side-request-forgery>

A screenshot of a computer

Description automatically generated

In summary, the OWASP Top 10 is important because it provides a list of the most critical web application security risks. It serves as a guide for developers, security professionals, and organizations to understand and prioritize the most prevalent and impactful vulnerabilities that can be exploited in web applications. By focusing on the OWASP Top 10, developers can ensure that their applications are designed, developed, and tested with security in mind. Addressing these vulnerabilities early in the software development lifecycle helps in reducing the risk of cyber-attacks, data breaches, and potential financial losses.

Impact and mitigation:

1. **Broken Access Control:**
   * **Impact:** Unauthorized access, data breaches, and compromised system integrity.
   * **Mitigation:** Enforce least privileges, deny by default, validate permissions on every request, and implement feature and attribute-based controls.
2. **Cryptographic Failures:**
   * **Impact:** Exposure of sensitive data due to weak or non-existent cryptographic algorithms.
   * **Mitigation:** Discard unused data, disable caching for sensitive responses, use appropriate initialization vectors, update cryptographic functions, and enforce key rotation.
3. **Injection:**
   * **Impact:** Execution of unauthorized commands through user-supplied data.
   * **Mitigation:** Use safe APIs, employ positive server-side input validation, escape special characters in residual dynamic queries, and implement LIMIT controls.
4. **Insecure Design:**
   * **Impact:** Security consequences due to missing user input bounds, unsafe APIs, or elevated privileges.
   * **Mitigation:** Follow security-centric best practices, conduct thorough code reviews, perform regular audits, and integrate security into the development lifecycle.
5. **Security Misconfiguration:**
   * **Impact:** Unauthorized access, data exposure, and exploitation of insecure default settings.
   * **Mitigation:** Establish a repeatable hardening process, install patches regularly, remove unused features, review cloud storage permissions, and implement access controls.
6. **Vulnerable and Outdated Components:**
   * **Impact:** Exploitation of known vulnerabilities in third-party components.
   * **Mitigation:** Maintain an inventory of components, update regularly, remove unused dependencies, install components via trusted channels, and choose well-maintained dependencies.
7. **Identification and Authentication Failures:**
   * **Impact:** Credential stuffing, brute force attacks, weak password validation.
   * **Mitigation:** Require strong passwords, secure password data stores, implement multi-factor authentication, and prevent user enumeration.
8. **Software and Data Integrity Failures:**
   * **Impact:** Unauthorized modification or deletion of data.
   * **Mitigation:** Use digital signatures, secure repositories, enforce secure code reviews, implement a software supply chain security program, and address deserialization vulnerabilities.
9. **Security Logging and Monitoring Failures:**
   * **Impact:** Lack of visibility into network activity and incidents.
   * **Mitigation:** Ensure proper login and access control logging, backup logs, implement monitoring systems in real-time, and protect logs for integrity.
10. **Server-Side Request Forgery:**
    * **Impact:** IP scanning, reconnaissance, introduction of malicious components.
    * **Mitigation:** Implement response handling, use whitelists and DNS resolution, disable unused URL schemas, and enforce access restrictions for internal services.

Incorporating these mitigation measures into the development lifecycle, conducting regular security assessments, and staying informed about emerging threats can significantly enhance the security posture of web applications. It's crucial for organizations to prioritize these measures to protect against potential cyber threats and safeguard sensitive data.