OWASP, the **Open Worldwide Application Security Project** (formerly Open Web Application Security Project), is an online community that publishes open-source information and resources on IoT, system software and web application security.[5] It is led by a non-profit called The OWASP Foundation.

The OWASP Top 10 serves as a widely recognized guide for developers and web application security professionals. It reflects a collective agreement on the most significant security threats facing web applications. Organizations are encouraged to implement this framework as a starting point to reduce these risks. Leveraging the OWASP Top 10 is an effective initial step toward fostering a software development culture focused on creating more secure code.

Top Ten Web Application Security Risks:

- OWASP publishes a list of the “OWASP Top Ten,” which is a regularly updated document highlighting the most critical web application security risks. This list helps organizations prioritize their efforts to address common vulnerabilities.

Guidelines and Best Practices:

- OWASP produces a variety of guides, cheat sheets, and best practices covering a wide range of security topics. These resources are intended to help developers, architects, and security professionals build secure applications.

Projects and Tools:

- OWASP sponsors and supports numerous open-source projects and tools related to web application security. These projects cover areas such as code analysis, penetration testing, security awareness, and more.

Conferences and Events:

- OWASP organizes conferences and events worldwide, bringing together security professionals, developers, and researchers to share knowledge and discuss the latest trends and challenges in application security.

Community Collaboration:

- OWASP is driven by a global community of volunteers and professionals who contribute to its projects, share expertise, and collaborate to improve web application security.

1. Broken Access Control

Access control refers a system that controls access to information or functionality. Broken access controls allow attackers to bypass authorization and perform tasks as though they were privileged users such as administrators. For example a web application could allow a user to change which account they are logged in as simply by changing part of a URL, without any other verification.

Access controls can be secured by ensuring that a web application uses authorization tokens* and sets tight controls on them.

**Many services issue authorization tokens when users log in. Every privileged request that a user makes will require that the authorization token be present. This is a secure way to ensure that the user is who they say they are, without having to constantly enter their login credentials.*

2. Cryptographic Failures

If web applications do not protect sensitive data such as financial information and passwords using encryption, attackers can gain access to that data and sell or utilize it for nefarious purposes. They can also steal sensitive information by using an on-path attack.

The risk of data exposure can be minimized by encrypting all sensitive data, authenticating all transmissions, and disabling the caching* of any sensitive information. Additionally, web application developers should take care to ensure that they are not unnecessarily storing any sensitive data.

**Caching is the practice of temporarily storing data for re-use. For example, web browsers will often cache webpages so that if a user revisits those pages within a fixed time span, the browser does not have to fetch the pages from the web.*

3. Injection

Injection attacks happen when untrusted data is sent to a code interpreter through a form input or some other data submission to a web application. For example, an attacker could enter SQL database code into a form that expects a plaintext username. If that form input is not properly secured, this would result in that SQL code being executed. This is known as an SQL injection attack.

The Injection category also includes cross-site scripting (XSS) attacks, previously their own category in the 2017 report. Mitigation strategies for cross-site scripting include escaping untrusted HTTP requests, as well as using modern web development frameworks like ReactJS and Ruby on Rails, which provide some built-in cross-site scripting protection.

In general, Injection attacks can be prevented by validating and/or sanitizing user-submitted data. (Validation means rejecting suspicious-looking data, while sanitization refers to cleaning up the suspicious-looking parts of the data.) In addition, a database admin can set controls to minimize the amount of information an injection attack can expose.

Learn more about how to prevent SQL injections.

4. Insecure Design

Insecure Design includes a range of weaknesses that can be emdedded in the architecture of an application. It focuses on the design of an application, not its implementation. OWASP lists the use of security questions (e.g. "What street did you grow up on?") for password recovery as one example of a workflow that is insecure by design. No matter how perfectly such a workflow is implemented by its developers, the application will still be vulnerable, because more than one person can know the answer to those security questions.

The use of threat modeling prior to an application's deployment can help mitigate these types of vulnerabilities.

5. Security Misconfiguration

Security misconfiguration is the most common vulnerability on the list, and is often the result of using default configurations or displaying excessively verbose errors. For instance, an application could show a user overly-descriptive errors which may reveal vulnerabilities in the application. This can be mitigated by removing any unused features in the code and ensuring that error messages are more general.

The Security Misconfiguration category includes the XML External Entities (XEE) attack — previously its own category in the 2017 report. This is an attack against a web application that parses XML* input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An 'external entity' in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker. The best ways to prevent XEE attacks are to have web applications accept a less complex type of data, such as JSON, or at the very least to patch XML parsers and disable the use of external entities in an XML application.

**XML or Extensible Markup Language is a markup language intended to be both human-readable and machine-readable. Due to its complexity and security vulnerabilities, it is now being phased out of use in many web applications*.

6. Vulnerable and Outdated Components

Many modern web developers use components such as libraries and frameworks in their web applications. These components are pieces of software that help developers avoid redundant work and provide needed functionality; common example include front-end frameworks like React and smaller libraries that used to add share icons or A/B testing. Some attackers look for vulnerabilities in these components which they can then use to orchestrate attacks. Some of the more popular components are used on hundreds of thousands of websites; an attacker finding a security hole in one of these components could leave hundreds of thousands of sites vulnerable to exploit.

Component developers often offer security patches and updates to plug up known vulnerabilities, but web application developers do not always have the patched or most-recent versions of components running on their applications. To minimize the risk of running components with known vulnerabilities, developers should remove unused components from their projects, as well as ensure that they are receiving components from a trusted source that are up to date.

7. Identification and Authentication Failures

Vulnerabilities in authentication (login) systems can give attackers access to user accounts and even the ability to compromise an entire system using an admin account. For example, an attacker can take a list containing thousands of known username/password combinations obtained during a data breach and use a script to try all those combinations on a login system to see if there are any that work.

Some strategies to mitigate authentication vulnerabilities are requiring two-factor authentication (2FA) as well as limiting or delaying repeated login attempts using rate limiting.

8. Software and Data Integrity Failures

Many applications today rely on third-party plugins and other external sources for their functionality, and they do not always make sure that updates and data from those sources have not been tampered with and originate from an expected location. For instance, an application that automatically accepts updates from an outside source could be vulnerable to an attacker uploading their own malicious updates, which would then be distributed to all installations of that application. This category also includes insecure deserialization exploits: these attacks are the result of deserializing data from untrusted sources, and they can result in serious consequences like DDoS attacks and remote code execution attacks.

To help ensure data and updates have not had their integrity violated, application developers should use digital signatures to verify updates, check their software supply chains, and ensure that continuous integration/continuous

deployment (CI/CD) pipelines have strong access control and are configured correctly.

9. Failures in Security Logging and Monitoring

Many web applications fail to take adequate measures to identify data breaches. On average, breaches go unnoticed for about 200 days, giving attackers ample time to inflict damage before any action is taken. OWASP advises developers to implement comprehensive logging, continuous monitoring, and incident response plans to quickly detect and respond to attacks on their applications.

10. Server-Side Request Forgery (SSRF)

Server-Side Request Forgery occurs when an attacker tricks a server into fetching a resource it shouldn't, even if that resource is normally protected. For instance, an attacker could request a URL like www.example.com/super-secret-data/ and gain access to confidential information from the server's response. To prevent SSRF attacks, it is crucial to validate all client-supplied URLs and ensure that invalid requests do not trigger direct responses from the server.