# OWASP Top 10 - 2021

This room breaks each OWASP topic down and includes details on the vulnerabilities, how they occur, and how you can exploit them. You will put the theory into practice by completing supporting challenges.

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging & Monitoring Failures
10. Server-Side Request Forgery (SSRF)

# Broken Access Control

Websites have pages that are protected from regular visitors. For example, only the site's admin user should be able to access a page to manage other users. If a website visitor can access protected pages they are not meant to see, then the access controls are broken.

Simply put, broken access control allows attackers to bypass **authorization**, allowing them to view sensitive data or perform tasks they aren't supposed to.

# Cryptographic Failures

A **cryptographic failure** refers to any vulnerability arising from the misuse (or lack of use) of cryptographic algorithms for protecting sensitive information. Web applications require cryptography to provide confidentiality for their users at many levels.

# Injection

Injection flaws are very common in applications today. These flaws occur because the application interprets user-controlled input as commands or parameters. Injection attacks depend on what technologies are used and how these technologies interpret the input. Some common examples include:

* **SQL Injection:** This occurs when user-controlled input is passed to SQL queries. As a result, an attacker can pass in SQL queries to manipulate the outcome of such queries. This could potentially allow the attacker to access, modify and delete information in a database when this input is passed into database queries. This would mean an attacker could steal sensitive information such as personal details and credentials.
* **Command Injection:** This occurs when user input is passed to system commands. As a result, an attacker can execute arbitrary system commands on application servers, potentially allowing them to access users' systems.

The main defence for preventing injection attacks is ensuring that user-controlled input is not interpreted as queries or commands. There are different ways of doing this:

* **Using an allow list:** when input is sent to the server, this input is compared to a list of safe inputs or characters. If the input is marked as safe, then it is processed. Otherwise, it is rejected, and the application throws an error.
* **Stripping input:** If the input contains dangerous characters, these are removed before processing.

To execute inline commands, you only need to enclose them in the following format $(your\_command\_here). If the console detects an inline command, it will execute it first and then use the result as the parameter for the outer command. Look at the following example, which runs whoami as an inline command inside an echo command:

# Insecure Design

**Insecure design** refers to vulnerabilities which are inherent to the application's architecture. They are not vulnerabilities regarding bad implementations or configurations, but the idea behind the whole application (or a part of it) is flawed from the start. Most of the time, these vulnerabilities occur when an improper threat modelling is made during the planning phases of the application and propagate all the way up to your final app. Some other times, insecure design vulnerabilities may also be introduced by developers while adding some "shortcuts" around the code to make their testing easier. A developer could, for example, disable the OTP validation in the development phases to quickly test the rest of the app without manually inputting a code at each login but forget to re-enable it when sending the application to production.

# Security Misconfiguration

# Vulnerable and Outdated Components

# Identification and Authentication Failures

# Software and Data Integrity Failures

# Security Logging & Monitoring Failures

# Server-Side Request Forgery (SSRF)

# 