Biggest Web Security Risks:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. XSS
8. Insecure Deserialization
9. Componentes with Known Vulnerabilities
10. Insufficient Logging & Monitoring

0.1 SQL Injection

Consider this query:

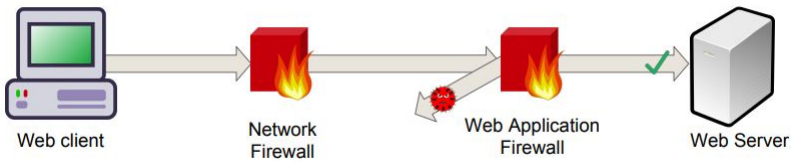
```
db.query("SELECT * FROM users WHERE email = '" +
inputEmail + "' AND password = '" + inputPassword + "'");
```

What happens when **inputPassword** has the following value? `' OR '1'='1`

```
SELECT * FROM users WHERE email = 'admin@fe.up.pt'
AND password = '' OR '1'='1'
```

Prevention:

- Never trust the user
- **Prepared statements**
- Stored procedures
- Hide error messages
  - Blind injection is way harder to perform
- ~~Regular expressions~~
- Web Application Firewall



0.2 Authentication

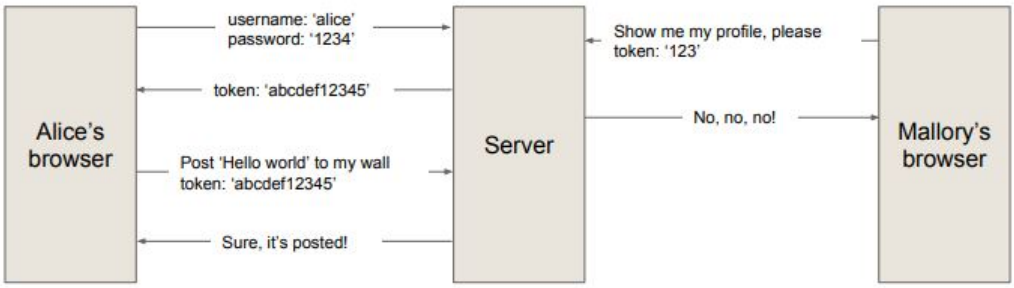
How to build a secure authentication system?

Challanges:

- Password encryption
- Block "easy" password
- Limit failed attempts
- CSRF, Session forgery, ...
- Forgot my password
- Forgot my ID
- Multi-factor
- SSO
- Revoking sessions
- Reset all passwords after breaches

Sessions

Typical workflow with sessions:



Session Tokens are usually stored in cookies.

Cookies

- **Secure:** A secure cookie is only sent to the server with an encrypted request over the HTTPS protocol.
- **HttpOnly:** To prevent cross-site (XSS) attacks, HttpOnly cookies are inaccessible via Javascript.
- **Domain:** Domain specifies allowed hosts that will receive the cookie.
- **Path:** Path indicates a URL path that must exist in the requested URL in order to send the Cookie header.

Session Hijacking

Sesion Hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system.

Main Methods:

- Cross-site scripting
- Session fixation
- Non-secure communications
- Malware

Session Fixation

Consider this scenario:

1. Bruno knows that `https://unsafe.example.com` accepts session tokens from query strings.
2. Bruno tells Chico to visit `https://unsafe.example.com?SID=123456789`.
3. Chico clicks the URL and logs in with her credentials for `https://unsafe.example.com`.

Chico is now using the session token **123456789**, which was provided by Bruno, so Bruno can use the same token to access Chico's account.

Prevention:

- Reject session identifiers from GET/POST variables.
- Regenerate the session token in every request.
  - Not always possible
- Regenerate the session ID when users log in.

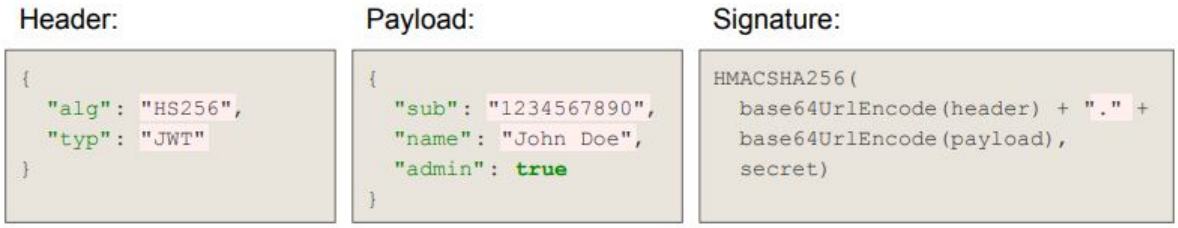
JWT

JSON web token is a standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

Use cases:

- **Authorization:** Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. It is also commonly used in SSO.
- **Information Exchange:** JWT is a good way of securely transmitting information between parties.

A JWT typically looks as: `<header>.<payload>.<signature>`



- Header

  - Base64Url encoded
- Payload
  - Base64Url encoded
- Signature
  - Signed using HMAC256
  - Message: `Base64(header) + "." + Base64(payload)`
  - Key: secret key

0.3 Cross-site scripting (XSS)

Cross-site scripting is a type of attack where malicious code is injected in web pages trusted and viwed by other users.

Prevention:

Never trust user input...

- Anywhere in your HTML document
- In CSS
- In a script
- ...

Make sure you always **encode** untrusted data.

0.4 Sensitive Data Exposure

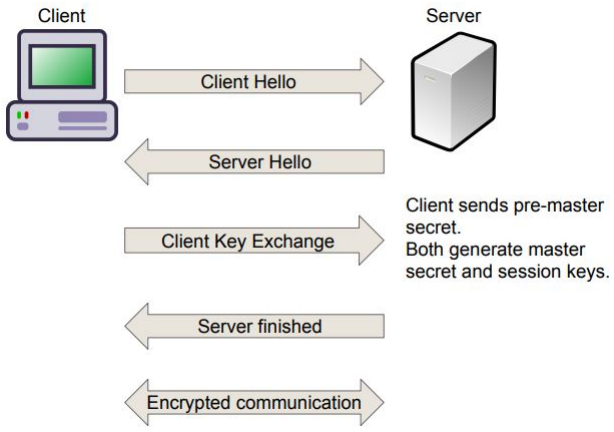
Public-key cryptography:

Public-key cryptography or asymmetric cryptography is an encryption scheme that uses two keys: a public key and a prive key. It is used to block **man-in-the-middle** attacks.

The private key is used to decrypt messages encrypted with the public key.

HTTPS:

HTTPS uses public-key cryptography to establish secure connections. The protocols works on top of TLS, which is an updated version of SLL.



HSTS:

HTTP Strict Transport Security is a web server directive that informs user agents and web browsers how to handle its connection.

```
Strict-Transport-Security: max-age=31536000
```

HSTS headers tell the browser to:

1. Always convert `http://` to `https://`.
2. Block the connection if a secure connection cannot be established.

Certificate Authorities:

A **Certificate Authority (CA)** is an entity that issues digital certificates. Web browsers trust a predefined list of root CAs, that is shipped with the browser.

Handling passwords:

A cryptographic hash function should have these properties:

- Deterministic

- Fast to compute
- Non-invertible
- Changing a single byte in the message generates a totally different output
- Very hard to find collisions

If an intruder gains access to a database containing hashed passwords, there is no easy way of obtaining the original passwords.

However, he may take advantage of a **rainbow table**.

Salts help fighting rainbow tables and make password cracking more difficult.

0.5 Security Misconfiguration

- Keeping default accounts
- Directory listing enabled
- Too much information in errors (such as stack traces)
- Out-of-date software
- Not disabling debug features
  - If a debug feature is necessary then restrict access by user type and IP address.
- ...

0.6 CSRF

Cross-Site Request Forgery attacks stem from the capability that a site has to issue a request to another site. Imagine a malicious website contains the following form:

```
<form action="https://vulnerable-bank.com/transfer" method="POST">
        <input type="hidden" name="to_account" value="14278935">
        <input type="hidden" name="amount" value="£1,000">
</form>
```

The malicious website can use Javascript to submit that form for you. Since you have previously performed login in `https://vulnerable-bank.com`,your money will be transferred.

Prevention:

- Anti-CSRF tokens

```
<form action="https://vulnerable-bank.com/transfer" method="POST">
      <input type="hidden" name="to_account" value="14278935">
      <input type="hidden" name="amount" value="£1,000">
      <input type="hidden" name="csrf_token"
value="12345678987654321">
</form>
```

- Same-site Cookies

```
Set-Cookie: sess=abc123; path=/; SameSite=lax
```

  - Two modes: `strict` or `lax`

Reverse Tabnabbing

The following HTML is vulnerable to **Reverse Tabnabbing**:

```
<a href="bad.example.com" target="_blank">
```

Prevention is simple - add `rel="noopener"`:

```
<a href="bad.example.com" target="_blank" rel="noopener">
```