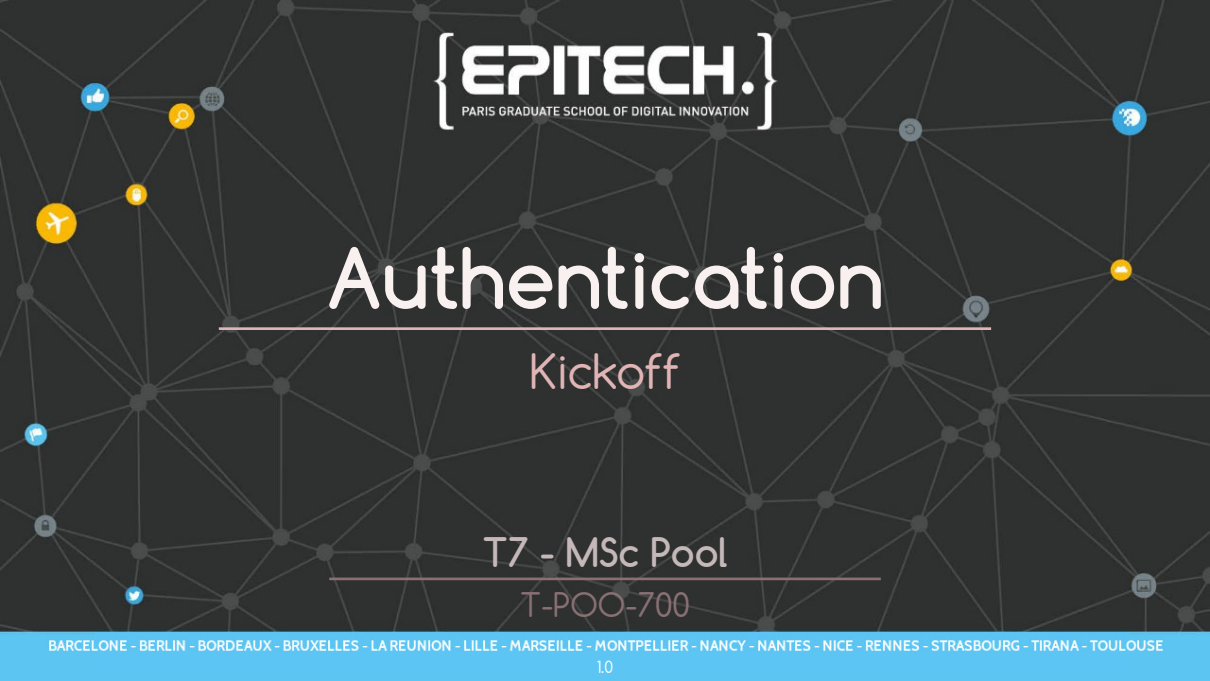



# Authentication

## Kickoff

T7 - MSc Pool

T-POO-700

# Security

For web servers, Internet security is managed at several levels:

- at the network level by filtering the flows having access to the server (via a firewall), which typically will only allow the http or https protocol;
- at the system level by updating basic software and software managing user access;
- at the application level by means of **authentication measures**, authorization, monitoring of access, traceability of the actions carried out, etc.

# Authentication

- makes it possible to identify oneself with reliability to an HTTP server
- give the proper access to a user, according to his/her rights level

# Basic and Digest

- **Basic Method**
  Simplest, but least secure
  Transmits a base-64-encoded password which is easily decoded
- **Digest Method**
  Does not transmit the password in plain text
  Requires it to be stored (or its SHA1 hash, which is sufficient to identify itself and can be considered as a password) in plain text
  Safer than the "Basic" method, but still susceptible to attacks

# JSON Web Token (JWT)

Open standard defined in RFC 75191
Secure exchange of tokens between multiple parties
It is composed of three parts :

- Header (*"Header"*) : contains token information (token type and hashing algorithm)
- The Payload (*"Payload"*) : contains the information you want to store in this token
- Digital Signature (*"Signature"*): allows verification of the authenticity of the token

# The XSS and CSRF flaws

**XSS** (Cross-Site Scripting) vulnerability:
involves injecting javascript code on a site in order to recover user data (like cookies).
To prevent cookie theft, the easiest way is to use cookies *"HttpOnly"* (transmitted in all your requests but not accessible by your javascript code).

**CSRF** (Cross-Site Request Forgery vulnerability:
involves sending a "dangerous" link (an account deletion link for example) to a conencted user. When the user clicks, the action will be executed even before the user knows what it is. To protect themselves, we usually use tokens called "XSRF tokens".

# Example of XSS attack

- I create an account and start writing an article
- I inject JavaScript code that will retrieve the user cookies in my article
- I publish my message on the site, my JavaScript code is then executed by all the users
- An administrator connects to the site : the JavaScript code sends me their cookies
- "I import" these in my browser : I am logged in as the administrator
- I can then delete all items from the site!

# Example of CSRF attack

*Context*: My online bank is not secure: I have a URL allowing me to make a transfer to an external account without verification.

- I create my URL with the IBAN and the amount of my choice
- I "dress up" my url using a tool such as "tinyurl" so as not to panic my target user
- I send my dressed up URL to a customer of this bank
- The user clicks on the link:
  if he/she is connected to the site: the transfer is instantaneous
  if not: the user gets connected thinking it is safe, and the transfer happens

# The SQL Injection

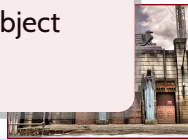
- attack that sends SQL code to a server so that it executes it and returns the result to the user
- often used to recover normally inaccessible data or to modify the database
- based on the fact that special characters (such as ' or " for example) are not escaped, which can be easily avoided, so this flaw is not widespread today

# SQL injection example

*Contect*: Special characters are not escaped on my webmail.

- I provide the pseudonym of another user
- I create a piece of SQL query to bypass the password check
- My injection is a success, I'm logged in as this other user
- I have access to all its messaging, and therefore to all the resulting actions

We remind you that such operations are prohibited by law and subject to a fine.



# Caution

The faults mentioned above are now well known, and often protected.
A lot of framework can also help you protect against these flaws, provided you use the right functions.

Nevertheless, there still exists sites where such flaws are exploitable. This can happen for example when the tools of a framework are misused; in these cases, it is the developer who "creates" flaws without their knowledge.

# Protect yourself effectively

- Using an XSRF token
- Using JWT
- Escape all special / interpretable characters from user inputs
- Store and send only essential information for the proper functioning of the site
- Use the HTTPS protocol (free SSL certificates exist !)

Where possible, you can also :

- Associate XSRF token and JWT
- Privilege cookies *"HttpOnly"*
- Watch out for the latest flaws discovered
- In the case of using a framework or external libraries, keep them up-to-date

# Any questions

?