



# WEB SECURITY

---

# 01 – INTRODUZIONE

---



# WORLD WIDE WEB

---

- Servizio utilizzato per la **condivisione di informazioni** (pagine web, immagini, etc...)
- Componenti principali
  - **HTTP** (HyperText Transfer Protocol)
  - **HTML** (HyperText Markup Language)
  - **Client**
  - **Server**

Si basa sull'architettura client-server!





# HTTP

---

- Utilizza **TCP** (Transmission Control Protocol)
- Il principale protocollo utilizzato per lo scambio di informazioni nel web.
- Ha diverse versioni
- Definisce il modo in cui il client e il server comunicano tra di loro.



# RICHIESTA HTTP

```
1 GET / HTTP/1.1 \r \n
```

Request line

```
2 Host: example.com \r \n
```

```
3 User-Agent: Mozilla/5.0 (Windows NT 10.0;  
Chrome/121.0.6167.160 Safari/537.36 \r \n
```

```
4 Connection: close \r \n
```

Headers

```
5 \r \n
```

Body

```
6
```

# RICHIESTA HTTP

```
1 POST / HTTP/2 \r \n
2 Host: example.com \r \n
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win6
  Chrome/121.0.6167.160 Safari/537.36 \r \r \n
4 Content-Length: 14 \r \n
5 \r \n
6 username=admin
```

A cosa serve? 🍌

Body

# RICHIESTA HTTP

GET	/helloooo.html	HTTP/2 \r \n
Metodo	Uniform Resource Locator	Versione HTTP

**Metodi HTTP:** <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

**Versioni HTTP:** [https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/Evolution\\_of\\_HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP)

# RISPOSTA HTTP

```
HTTP/2 200 HTTP/2 \r \n
Content-Type: text/html; charset=UTF-8 \r \n
Server: ECS (dce/26A0) \r \n
Content-Length: 1256 \r \n
\r \n
<!doctype html> \r \n
<html> \r \n
<head> \r \n
    <title>Example Domain</title>|
```



# RISPOSTA HTTP

HTTP/2	200	OK	\r	\n
Versione HTTP	Status Code	Status Message		

Tutti gli status code: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>



# PAGINE WEB

---

Le pagine web possono includere vari tipi di risorse!

- Immagini
- Video
- CSS (per definire lo stile della pagina)
- JavaScript (per rendere una pagina dinamica)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Esempio</title>
  </head>
  <body>
    <h1>Un titolo</h1>
    <p style="color: red;">
      Un paragrafo rosso
    </p>
    <script>
      alert("Hello world");
    </script>
    <script src="/main.js"></script>
  </body>
</html>
```



# JAVASCRIPT

---

Un linguaggio di programmazione dinamicamente tipizzato. Nato per essere usato nelle pagine web ma (al giorno d'oggi) viene utilizzato anche altrove.

The logo for JavaScript, featuring the letters 'J' and 'S' in a bold, black, sans-serif font. The 'J' has a thick vertical stem and a curved bottom. The 'S' is also bold and rounded. They are centered within a bright yellow square, which is itself centered on a dark gray background.



# COOKIES

---

- Utilizzati per identificare uno specifico utente
- Vengono gestiti tramite gli header **Cookie:** (Client) e **Set-Cookie:** (Server)



# PRIMA DI CONTINUARE...

---

Risoluzione delle seguenti challenge

Web 01, Web 02, Web 03, Web 04, Web 08

Web 05, Web 06, Web09

# STRUMENTI

---



# STRUMENTI

---

- DevTools (Set di strumenti presenti di default su tutti i browser)
- Requests (Libreria di python)
- **Burp Suite**



# BURP SUITE

---



<https://portswigger.net/burp/communitydownload>

FINE PARTE 1



# PRIMA DI CONTINUARE...

---

Risoluzione delle seguenti challenge

Web 01, Web 02, Web 03, Web 04, Web 08

Web 05, Web 06, Web09

**Cookie Monster Army**

**Gabibbo Says – Training Camp 1**

**MEME Shop – Training Camp 2**

**solo una convenzione – Training Camp 3**

**Gabibbo's Friend**



# COSA C'È CHE NON VA?

---

```
from flask import Flask, request

app = Flask(__name__)

#? è una post o una get?
@app.get("/")
def index():
    #! Args è di tipo 'dict' -> /?arg1=hello&arg2=world
    #! "Hello World" in questo caso è il valore di default
    data = request.args.get("query", "Hello World")

    # Rispondo al client con il contenuto di data
    #> Content-Type: text/html
    return data

if __name__ == "__main__":
    # Vado ad avviare il server
    app.run("0.0.0.0", 1337)
```

# COSA C'È CHE NON VA?

---

```
import subprocess
from flask import Flask, request

app = Flask(__name__)

@app.get("/")
def index():
    host = request.args.get("host")

    if host:
        #!/ :think:
        return subprocess.check_output(f"ping {host}", shell=True)
    .decode()
    else:
        return "Hello World"

if __name__ == "__main__":
    app.run("0.0.0.0", 1337)
```

# COSA C'È CHE NON VA?

---

```
cursor.execute(f"SELECT * FROM Users WHERE username={username!r}  
and password={password!r}")
```

# 02 – CLIENT SIDE

---



# TIPS

---

- La flag si trova in un posto accessibile solamente da un determinato utente (oppure nei cookie)
- L'utente viene simulato da uno script
- Se è presente quello script oppure un pulsante con scritto «Report» o «Visit», allora è una challenge client-side
- <https://webhook.site/> e ngrok vi tornerà utile





# XSS

---

- XSS: (Cross Site Scripting)
- **Una injection** causata dalla gestione dell'input fatta male.
- **Permette di eseguire script (js) arbitrari.**



# TIPI DI XSS

---



## REFLECTED

Lo script malevolo proviene dalla richiesta HTTP appena fatta.



## STORED

Lo script malevolo proviene dal database del server



## DOM-BASED

Vulnerabilità in uno script client-side



# REFLECTED XSS

---

```
https://example.com/status?message=<script>alert()</script>
```

Presente in uno dei precedenti esempi...

# DOM-BASED XSS

---

```
var search = document.getElementById('search').value;  
var results = document.getElementById('results');  
results.innerHTML = 'You searched for: ' + search;
```

DOM (Document Object Model): Rappresentazione degli elementi di una pagina.  
Mai passare ad element.innerHTML l'input dell'utente!

# PRIMA DI CONTINUARE...

---

Risoluzione delle seguenti challenge

Curious George  
EasyShop





# CSRF

---

- CSRF: (Cross Site Request Forgery)
- Permette di eseguire delle azioni per conto di un altro utente (ad esempio transazioni, pubblicare post, etc...)
- **Non serve nessun tipo di injection!**
- **Bypassa la SOP (Same Origin Policy)**



# SAME ORIGIN POLICY

---

Due URL hanno la stessa origin se il protocollo, la porta e l'host sono gli stessi.



# SAME ORIGIN POLICY

---

```
// In http://example.com/  
fetch("http://example.com/currentUser") // Apposto
```

STESSA  
ORIGIN



# SAME ORIGIN POLICY

---

```
// In http://attacker.com  
fetch("http://example.com/currentUser") // Errore!
```

ORIGIN  
DIVERSA



# COME PROTEGGERSI?

## CONTENT SECURITY POLICY (CSP)

Protezione da XSS e altre injection

## CSRF TOKEN

Rende impossibile fare CSRF

E MOLTO ALTRO....

FINE PARTE 2



# 03 – SERVER SIDE

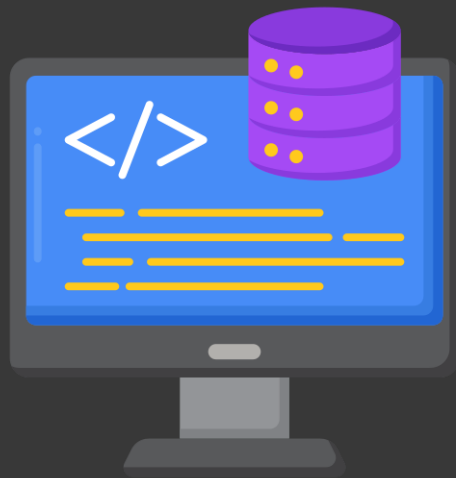
---



# INTRODUZIONE

---

- Di solito una **Web Application** è divisa in due (o più parti):
  - **Frontend** (Client side)
  - **Backend** (Server side)
- Il **backend** può essere scritto in vari **linguaggi** (Python, PHP, Java, etc...) e utilizzando vari **framework** (Flask, Spring, Django, etc..)
- Il **backend** si occupa di **elaborare e salvare i dati dell'utente** (login, register e altro)



# COME FACCIAMO A SALVARE I DATI DI UN UTENTE?

---

Sicuramente non dentro un .csv

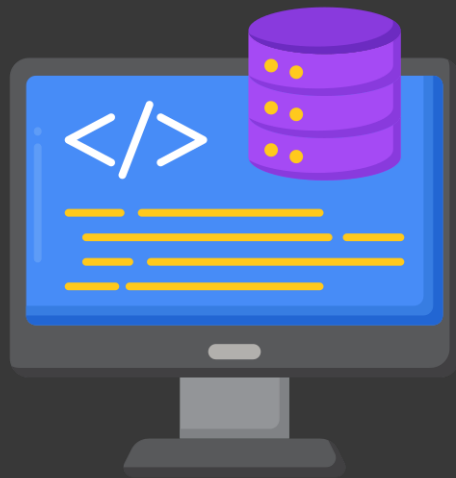




# DATABASE

---

- Ci permettono di salvare informazioni sugli utenti
- Ottimizzati per le operazioni di lettura e scrittura
- Le informazioni sono facili da recuperare

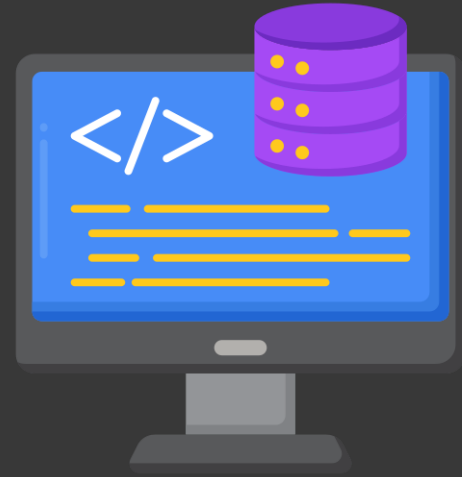
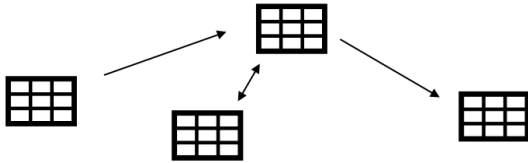




# DATABASE RELAZIONALI

---

- Permettono la creazione di relazioni fra tipi di dato
- Esempio: Database SQL (Structured Query Language)



# ESEMPIO

---

USER

<b>Username</b>	admin	salvatore
<b>Email</b>	admin@..	aranzulla@..
<b>Password</b>	12345	mario2019

POST

<b>Id</b>	1	2	3
<b>Titolo</b>	Come ..	Installar e...	Le basi di ...
<b>Testo</b>	Per fare	Oggi ...	I databse. .
<b>User</b>	admin	salvatore	admin



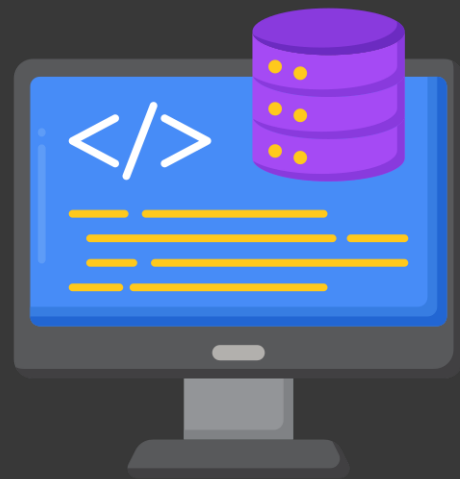
I Database SQL salvano le informazioni dentro a delle tabelle



# DATABASE RELAZIONALI

---

- Per contattare un database vengono utilizzate delle query
- Le query sono delle istruzioni inviate al database che fanno uso di un linguaggio specifico
- I database SQL utilizzano (appunto) SQL



# ESEMPIO

Recuperare la password dell'utente con username uguale ad «admin»

```
SELECT password FROM Users where username="admin";
```



# ESEMPIO

I backend utilizzano funzioni specifiche per interagire con i database...

```
cursor.execute(f"SELECT * FROM Users WHERE username={username!r}  
and password={password!r}")
```

Qua c'è qualcosa che non va....



# CODE INJECTION

---



# CODE INJECTION

---

- Causate dall'utilizzo di bad practices
- Capitano quando c'è di mezzo l'input di un utente
- Esistono TANTISSIMI tipi di injection



# SQL INJECTION- ESEMPIO

file sqli-example.py

Esempio di login

```
cursor.execute("SELECT * FROM users WHERE username='" + username +  
" ' and password='" + password + "'")
```



# SQL INJECTION

---

- Il miglior modo di trovarne una è inserire apici in tutti gli input processati dal backend
- Osservando la risposta, si può intuire (a volte) se il database ha restituito un errore per la query non corretta oppure no:
  - Pagina bianca
  - Status code 500 (Internal Server Error)
  - (*raramente*) il messaggio di errore del database
- In base al tipo di risposta, le SQL injection vengono chiamate in vari modi



# TIPI DI SQLI

---

## LOGIC SQLI

Cambiare la logica del backend

## BLIND SQLI

Non viene restituito nulla, in base agli errori si può capire se la query è andata a buon fine o no

## TIME BASED SQLI

Nessun tipo di output, utilizzando query particolari è possibile capire se la query è andata a buon fine basandoci sul tempo di risposta del server

## UNION SQLI

Recuperare informazioni da altre tabelle

*Molto noiose*

# PRIMA DI CONTINUARE...

---

Risoluzione delle seguenti challenge

Web 18, Web 19, Web 20




# COMMAND INJECTION

---

- Oltre ai database, i back-end hanno la necessità di interagire co altri programmi (per utilizzare funzioni di determinati programmi, ad esempio **ping**)

```
import os
os.system(f"ping {input()}")
```





# COMMAND INJECTION – ALTRO ESEMPIO

```
eval(input("Inserisci il calcolo da eseguire: "))
```



# PRIMA DI TERMINARE...

---

Esistono tantissime altre vulnerabilità (sia server side, sia client side).

Ci vorrebbe troppo tempo per vederle tutte.

# FINE

---

**Salvatore Abello, 5IB**

salvatore.abello2005@gmail.com  
<https://github.com/salvatore-abello>