Informe Laboratorio 2

Sección 1

Bruno Figueroa
e-mail: bruno.figueroa@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 instalar Docker en Ubuntu se pueden utilizar los siguientes comandos:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh ./get-docker.sh
```

Eso entrega una versión limpia de Docker sin nada, lista para acceder desde la terminal.

Luego, instala Docker Compose: `sudo apt install docker-compose`; esto entrega la versión más reciente de `compose` disponible.

Descarga el repositorio con DVWA desde `https://github.com/digininja/DVWA`. Este cuenta con un `Dockerfile` y `docker-compose` ya creados. Para desplegarlo, en una terminal ejecuta:

```
docker-compose up --build
```

Automáticamente se crearán las imágenes del repositorio y se desplegarán los contenedores.

Como se ve en la figura 1, se descarga y ejecuta el contenedor de DVWA, este se encuentra corriendo en su puerto por defecto: 80, pero este se cambia al puerto 4280 en el siguiente punto.

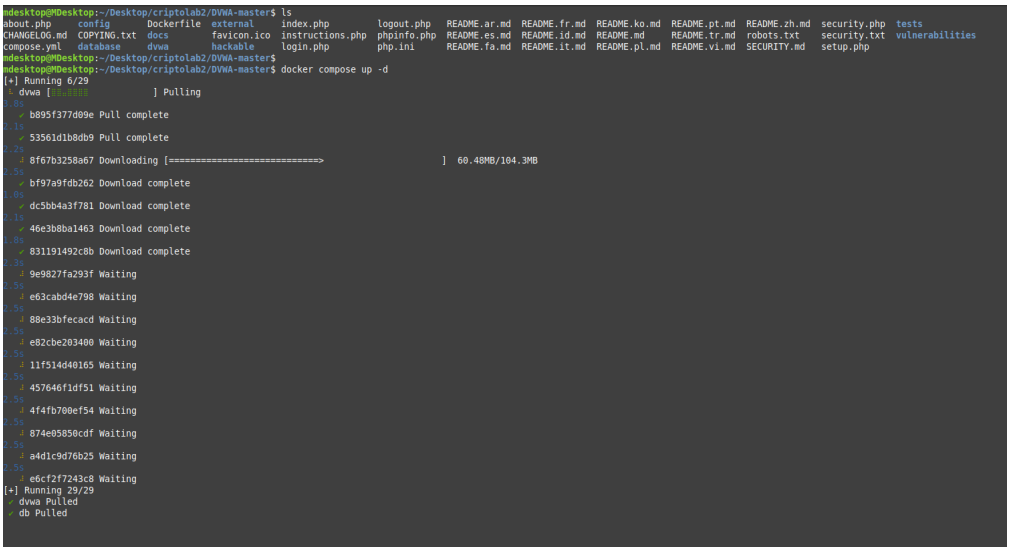

Figura 1: Creación de contenedor de DVWA utilizando docker compose

A continuación, se ingresa a `http://localhost:4280` e inicia sesión con el usuario por defecto `admin;admin`. Para luego hacer clic en el botón *Create / Reset Database* para terminar de configurar la aplicación, en la figura 2, se observa el botón que se debe apretar, y el feedback que entrega la página al hacerlo.

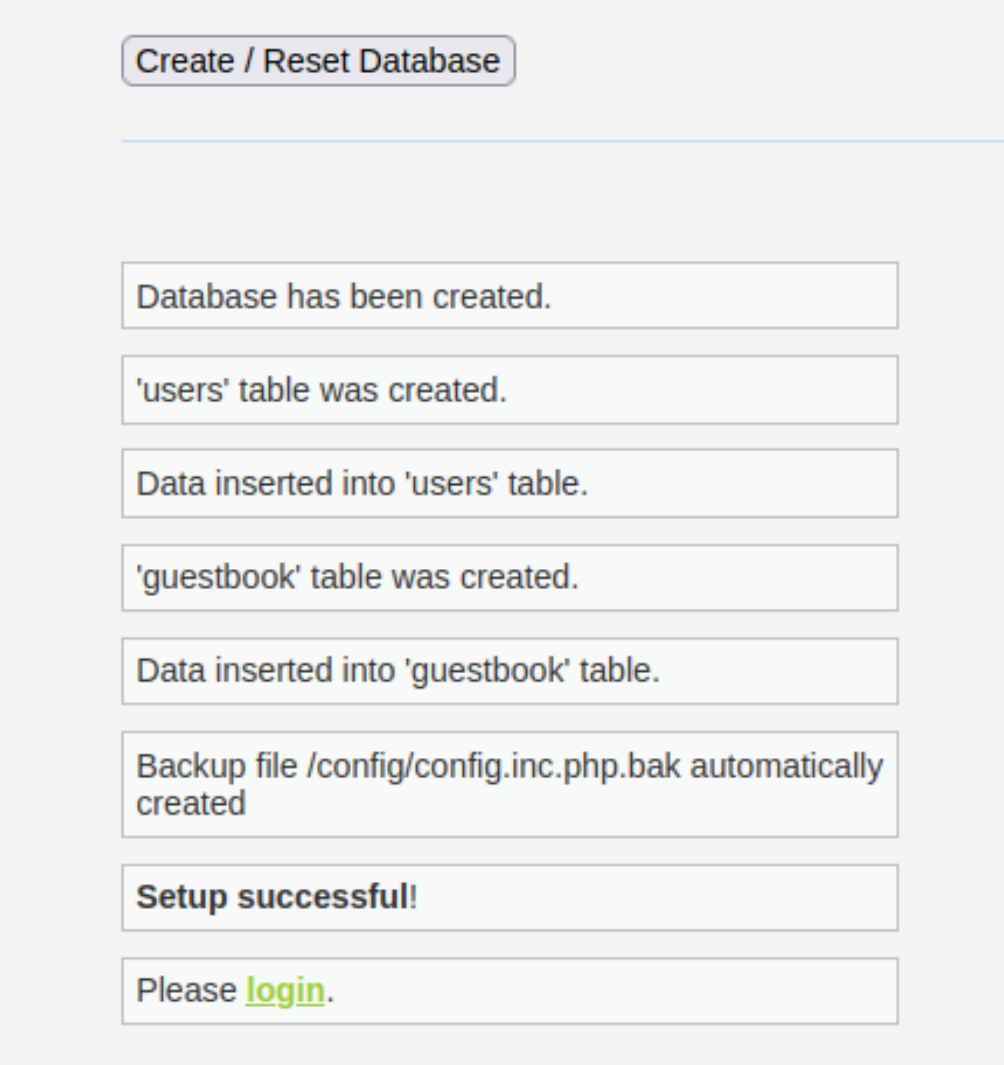

Figura 2: Creación de base de datos

2.2. Redirección de puertos en docker (dvwa)

El puerto por defecto funciona sin ninguna complicación, dado que este no entra en conflicto con ninguna otra app actual, pero a continuación se muestra cómo realizar el redireccionamiento de puertos en caso de existir una colisión.

Entrando al compose.yml, en el servicio dvwa en la línea que dice ports: cambiar de 80:80 a 4280:80, en la figura 3 se muestra la línea exacta que se debe cambiar, como fue indicado anteriormente.

```
Run All Services
services:
  Run Service
  dvwa:
    build: .
    image: ghcr.io/digininja/dvwa:latest
    # Change `always` to `build` to build from local source
    pull_policy: always
    environment:
      - DB_SERVER=db
    depends_on:
      - db
    # Uncomment the next 2 lines to serve local source
    # volumes:
    #   - ./:/var/www/html
    networks:
      - dvwa
    ports:
      - 127.0.0.1:4280:80
    restart: unless-stopped
```

Figura 3: Configuracion de puerto en docker compose

Esto cambia el puerto donde se ejecuta la aplicación de 80 a 4280, luego, para aplicar el cambio, es necesario reiniciar la app utilizando docker-compose down, y luego docker-compose up.

2.3. Obtención de consulta a replicar (burp)

Luego es necesario instalar Burp Suite Community desde `https://portswigger.net/burp/communitydownload` y ya instalado, abrir la pestaña *Proxy/Intercept*, para finalmente, abrir el navegador de Burp y acceder a DVWA en http://locahost:4280

En el nuevo navegador, y en la pagina web, se debe acceder con `admin;password`, para luego entrar a *Brute force* (`http://localhost:4280/vulnerabilities/brute/`) y activar la interceptación de paquetes en burp suite.

Ahora, se debe enviar un intento de login cualquiera (por ejemplo `12345;1234`) mientras se tiene activa la intercepción. Ese paquete será el mensaje base para el ataque.

En la figura 4 se muestra un ejemplo de un paquete interceptado, luego de mandar el mensaje admin;password, este es un ejemplo del tipo de paquetes que se utilizaran en los siguientes puntos.

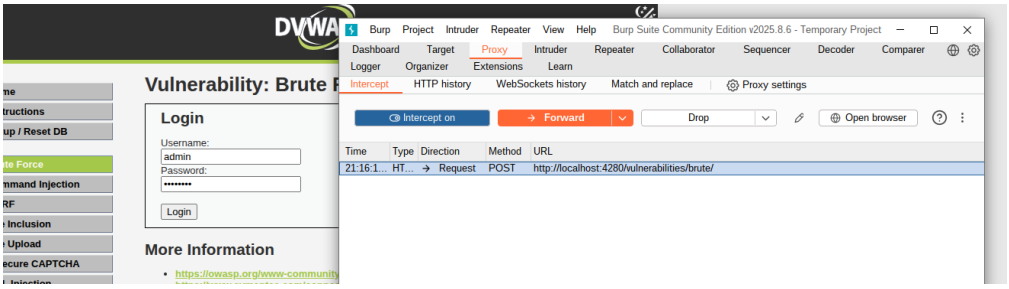

Figura 4: Paquete interceptado en burpsuite

2.4. Identificación de campos a modificar (burp)

Ahora, para continuar, se debe dar click derecho sobre el paquete capturado y seleccionar *Send to Intruder*. En la pestaña *Intruder* se puede observar el request que se envío. Limpia los payloads existentes, selecciona el campo `username` (12345) y el campo `password` (1234) y marca cada uno como posición para payloads separados.

En la figura 5 se muestra un paquete de ejemplo, el cual tiene marcados los cambios username y password utilizando el símbolo de posición, como se pueden observar destacados en naranja y azul respectivamente.

Figura 5: Campos para payload en burpsuite

2.5. Obtención de diccionarios para el ataque (burp)

En Internet hay múltiples diccionarios para ataques de fuerza bruta. En este caso se utilizarán los siguientes:

- Usernames: `https://gist.github.com/giper45/414c7adf883f113142c2dde1106c0c4c`
- Passwords: `https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/darkweb2017_top-100.txt`

Descarga los archivos `.txt` e impórtalos en los payloads de Intruder (selecciona el payload y pulsa *Load...*).

2.6. Obtención de al menos 2 pares (burp)

Con el target (`http://localhost:4280`), el payload 1 (usernames) y el payload 2 (passwords) cargados, pulsa *Start attack*. En la pestaña de resultados verás el progreso por payload.

Un request de fracaso tiene longitud 5055 bytes, mientras que un request de éxito tiene longitud 5092 bytes, lo que permite distinguirlos rápidamente.

Pares obtenidos: `admin:password`, `pablo:letmein` y `smithy:password`.

A continuación se muestra la figura 6 y 7 con 2 resultados de pares exitosos, con su mensaje de éxito, primero smithy, siendo solo el HTML y segundo pablo, siendo la visualización web.

```
  </form>
  <p>
    Welcome to the password protected area smithy
  </p>
  <img src="/hackable/users/smithy.jpg" />
</div>
```

Figura 6: HTML de ingreso exitoso para Smithy

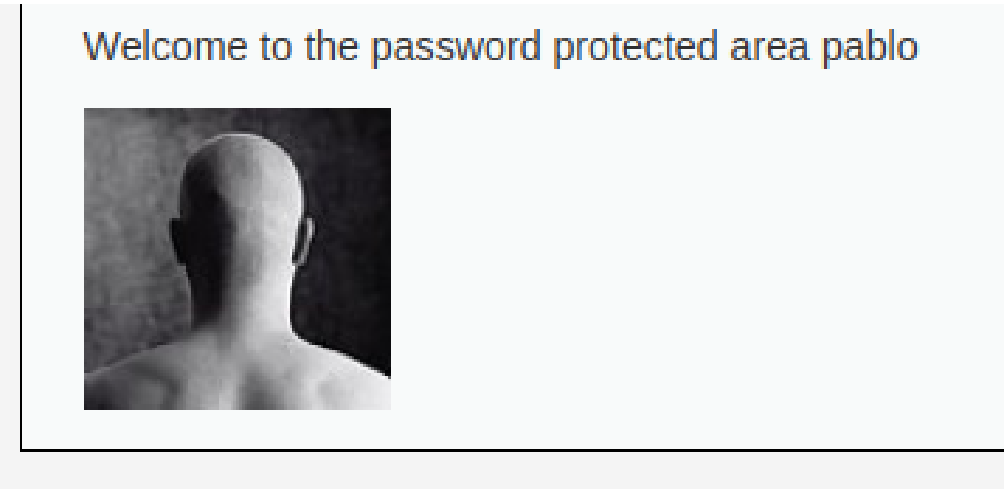

Figura 7: Pagina web con mensaje de ingreso exitoso para Pablo

2.7. Obtención de código de inspect element (curl)

En el navegador, pestaña *Network*, se captura el request GET usado para el login con `pablo;letmein` y se exporta como `curl`. También fue descargado el HTML resultante para comparaciones.

En la figura 8 se muestra el paquete GET obtenido en la pestaña Network en azul, luego de haber ingresado con el usuario pablo.

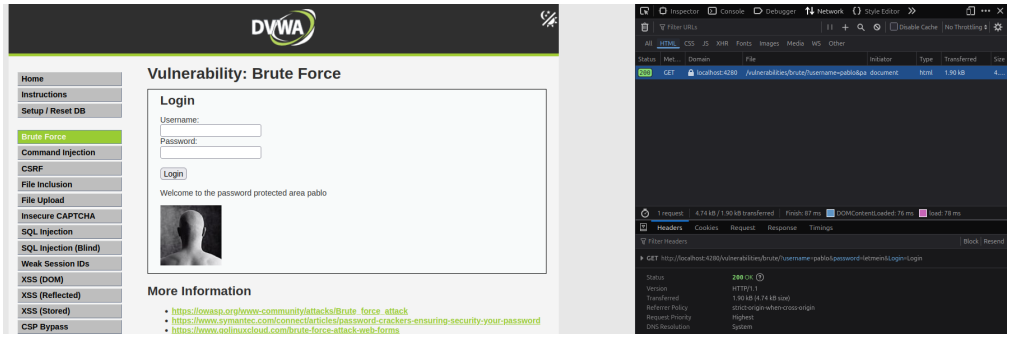

Figura 8: Mensaje GET enviado al ingresar con pablo

2.8. Utilización de curl por terminal (curl)

Se utilizaron los siguientes comandos cURL por terminal para enviar los 3 siguientes paquetes:

Acceso válido (`pablo:letmein`):

```
curl -i -s -L --compressed \
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
  -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
  -H 'Referer: http://localhost:4280/vulnerabilities/brute/' \
  -b 'security=low; PHPSESSID=6e6ee3e831fd666000c17b55935d95f1' \
  'http://localhost:4280/vulnerabilities/brute/?username=pablo&password=letmein&Login=
  -o body_pablo.html -D headers_pablo.txt
```

Acceso válido (`smithy:password`):

```
curl -i -s -L --compressed \
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
  -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
  -H 'Referer: http://localhost:4280/vulnerabilities/brute/' \
  -b 'security=low; PHPSESSID=6e6ee3e831fd666000c17b55935d95f1' \
  'http://localhost:4280/vulnerabilities/brute/?username=smithy&password=password&Logi
  -o body_smithy.html -D headers_smithy.txt
```

Acceso inválido (`invalido:no`):

```
curl -i -s -L --compressed \
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
  -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
  -H 'Referer: http://localhost:4280/vulnerabilities/brute/' \
  -b 'security=low; PHPSESSID=6e6ee3e831fd666000c17b55935d95f1' \
  'http://localhost:4280/vulnerabilities/brute/?username=invalido&password=no&Login=Lo
  -o body_invalid.html -D headers_invalid.txt
```

Estos paquetes incluyen la cabecera de respuesta en la salida, se envían en modo silencioso, y siguen redirecciones HTTP, los demás parámetros son la url de destino y los parámetros de acceso como username, password y cookies. En la figura 9 se muestra la ejecución de estos comandos en la terminal.

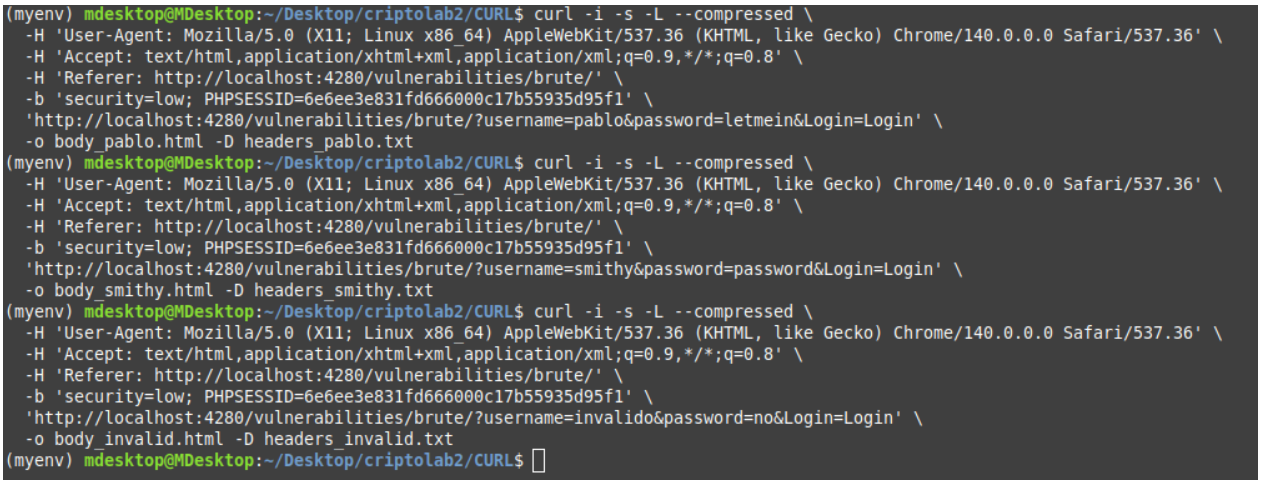

Figura 9: Mensaje GET enviado al ingresar con pablo

2.9. Demuestra 4 diferencias (curl)

Comparando el HTML/cabeceras entre acceso válido e inválido se observan 4 diferencias:

1. Diferencia en el cuerpo: el HTML válido contiene la línea `Welcome to the password protected area <username>`.
2. Imagen de usuario: El usuario correcto agrega una imagen de usuario al lado del mensaje de ingreso exitoso.
3. Longitud del response (Content-Length o tamaño del body): los responses exitosos suelen tener mayor tamaño (1520 vs 5055).
4. Fecha/tiempo de respuesta: los headers Date difieren (05:29:35 GMT para la válida vs 05:30:02 GMT para la inválida), lo que indica peticiones distintas en momentos distintos.

2.10. Instalación y versión a utilizar (hydra)

Hydra se instala con `sudo apt install hydra`. En el sistema usado para este trabajo la versión es 9.5, se puede comprobar utilizando el comando `hydra --version`.

2.11. Explicación de comando a utilizar (hydra)

A continuación se muestra el comando utilizado con hydra para realizar el ataque de fuerza bruta.

```
hydra -L username.txt -P password.txt -s 4280 -o hydra_found.txt -V \
 localhost http-get-form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Logi
```

Explicación de parámetros:

- `-L username.txt`: fichero con usuarios.
- `-P password.txt`: fichero con contraseñas.
- `-s 4280`: puerto del servicio en el host.
- `-o hydra`$_{f}ound.txt$: $ficherodesalidaconparesválidos$.
- `-V: modo verbose (muestra cada intento).`
- `http-get-form: modo para formularios GET; la sintaxis pide la ruta, parámetros con USER y PASS, la cabecera Cookie y la cadena que indica fallo (F=).`

A continuación se muestra la figura 10 donde se muestra parte del output de hydra en la terminal, hydra luego decide en base a estos outpus, cuales envia hacia hydra$_{f}ound.txtcomoposibles candi$

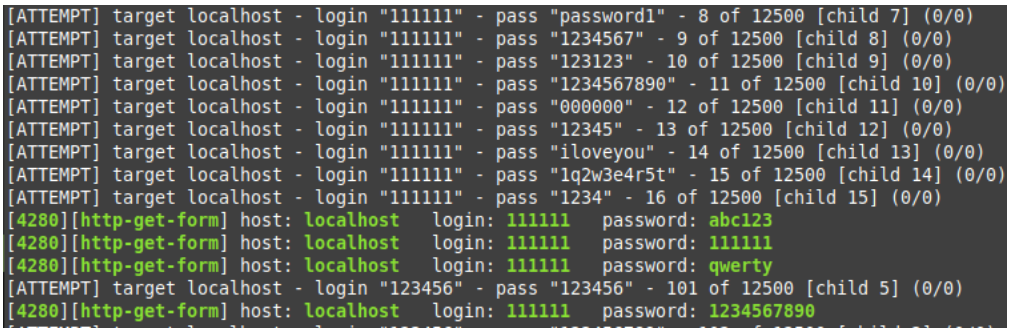

Figura 10: Ejecucion de hydra

2.12. Obtención de al menos 2 pares (hydra)

Se obtuvieron `admin:password`, `pablo:letmein` y `smithy:password`. Se observó que Hydra produjo una gran cantidad de falsos positivos que tuvieron que ser validados posteriormente con `curl`.

2.13. Explicación paquete curl (tráfico)

En el ejemplo obtenido con `curl` aparece la cabecera HTTP completa seguida del HTML descomprimido. El HTML mantiene la serialización original del servidor (ej. `<br />`, atributos en mayúsculas como `AUTOCOMPLETE=.°ff"` en el código fuente mostrado).

Algunas características útiles para identificar tráfico generado por curl en un pcap o dump:

- Presencia de la **línea de estado HTTP** y campos de respuesta en el mismo paquete si se capturó la respuesta cruda.
- Cabeceras típicas que `curl` envía por defecto (`User-Agent: curl/...`) a menos que se sobrescriban.
- HTML con la serialización original del servidor (cierres con `/`, capitalización de atributos) indicando que lo que se ve es la respuesta tal cual la sirvió el servidor.

2.14. Explicación paquete burp (tráfico)

En los datos proporcionados Burp muestra sólo el HTML body sin la línea de estado ni las cabeceras HTTP de respuesta. Además el HTML presenta pequeñas diferencias de serialización: etiquetas `<br>` sin slash de autocierre, atributos en minúsculas (`autocomplete=.°ff"`) y entidades HTML escapadas (`&amp;`) en URLs internas.

Algunos detalles que facilitan la detección:

- Ausencia de la línea de estado y de cabeceras de respuesta en la captura del body sugiere que la fuente es un proxy o captura ya procesada por un cliente.
- HTML normalizado (minúsculas, amp; escapado, distinto formato de cierre de etiquetas) indica reserialización por proxy o por el motor HTML del navegador.
- Burp puede introducir pausas humanas visibles en el timing si se usa intercept o repeater manualmente.

2.15. Explicación paquete hydra (tráfico)

Hydra genera peticiones automáticas en alta cadencia. Cuando no se configura para mantener cookies ni para emular navegador, las respuestas suelen ser las mismas que las del servidor pero sin cabeceras de sesión ni elementos de navegación dependientes de cookie.

Patrones únicos de hydra en el HTML y el tráfico:

- Respuestas repetitivas e idénticas en contenido por cada intento, a menos que el servidor incluya tokens de sesión diferentes por intento.
- User-Agent genérico. Si no se establece, puede aparecer un UA simple o no aparecer la familia de cabeceras de navegador.
- Patrón temporal característico: ráfagas rápidas de peticiones con intervalos regulares.

2.16. Mención de las diferencias (tráfico)

Diferencias observadas entre Burp y `curl` en los HTML y en las cabeceras:

- **Cabeceras HTTP:** `curl` (con `-i`) muestra la línea de estado y campos completos (`Server`, `X-Powered-By`, `Content-Encoding`, `Content-Length`, etc.). Burp en el ejemplo proporcionado presenta sólo el body HTML sin esas cabeceras en la captura suministrada.
- **Serialización del HTML:** El HTML devuelto por el servidor y mostrado por `curl` conserva la serialización original (ej. `<br />`, mayúsculas en atributos). El HTML capturado vía Burp aparece normalizado: `<br>` sin slash, atributos en minúscula y entidades escapadas (`&amp;`) en URLs internas.
- **Cookies y estado:** En un flujo proxyado por Burp se ven normalmente las cookies de sesión en la petición. En ataques automáticos (Hydra) es común la ausencia de cookies.
- **Timing:** Burp/proxy con interacción manual muestra pausas y retransmisiones, Hydra muestra ráfagas regulares; `curl` suele ser ad-hoc por intento y no sigue un patrón de alta frecuencia.

2.17. Detección de SW (tráfico)

Algunos detalles que se pueden observar para distinguir el tipo de trafico son:

- Si la captura incluye la línea de estado y cabeceras de respuesta completas en el mismo flujo, es probable que provenga de curl (sobretodo si el User-Agent es curl/...).
- Si sólo aparece el body y este está normalizado, es un paquete de proxy o captura desde navegador/Burp.
- Si las peticiones aparecen en ráfagas regulares y sin cookies de sesión, puede que sea una herramienta de fuerza bruta automatizada como lo es hydra.

2.18. Interacción con el formulario (python)

A continuación se explica el script de python utilizado, el cual aplica cURL para realizar un ataque de fuerza bruta en DVWA.

En la función `try_login` se envían todos los campos requeridos por el login en la línea:

```
url = f"{base_url}?username={quote(user)}&password={quote(pw)}&Login=Login"
```

Al cambiar `user` y `pw` se prueban las combinaciones, realizando el ataque de fuerza bruta.

2.19. Cabeceras HTTP (python)

En la variable `cmd` se especifican los encabezados que se envían con `curl` (User-Agent, Accept, Referer y Cookie). Es equivalente a ejecutar el `curl` mostrado anteriormente, pero desde Python y con variables.

Se muestra en la figura 11 el snippet de código que se encarga tanto del formulario (url) y de el header curl (cmd), en donde cada parámetro necesario se ingresa como una variable, con valores clave como usuario, contraseña y cookie.

```python
def try_login(base_url, referer, cookie, user, pw, extra_headers=None, timeout=20):
    url = f"{base_url}?username={quote(user)}&password={quote(pw)}&Login=Login"
    cmd = [
        "curl",
        "-s",
        "-L",
        "--compressed",
        "-H", f"User-Agent: {DEFAULT_USER_AGENT}",
        "-H", f"Accept: {DEFAULT_ACCEPT}",
        "-H", f"Referer: {referer}",
        "-b", cookie,
        url
    ]
```

Figura 11: Snippet de código python

2.20. Obtención de al menos 2 pares (python)

El script en Python devolvió los mismos pares válidos ya obtenidos (`admin:password`, `pablo:letmein`, `smithy:password`) esto es debido a que es el mismo diccionario, demostrando como estos son finalmente los únicos 3 pares obtenibles con los diccionarios de usuarios y contraseñas utilizados.

Se puede observar en la figura 12 el resultado en terminal del script de python, como intenta 1 por 1 con cada combinación posible, entregando un OK, en los casos correctos y un fallo en los incorrectos.

```
(myenv) mdesktop@MDesktop:~/Desktop/criptolab2/python$ python3 brute.py --users username.txt --passwords password.txt --output valid_pairs.txt
[1] Probando admin:password ... OK
[2] Probando admin:123456 ... fallo
[3] Probando admin:123456789 ... fallo
[4] Probando admin:111111 ... fallo
[5] Probando admin:qwerty ... fallo
[6] Probando admin:abc123 ... fallo
[7] Probando admin:12345678 ... fallo
[8] Probando admin:password1 ... fallo
[9] Probando admin:1234567 ... fallo
[10] Probando admin:123123 ... fallo
[11] Probando admin:1234567890 ... fallo
[12] Probando admin:000000 ... fallo
[13] Probando admin:12345 ... fallo
[14] Probando admin:iloveyou ... fallo
```

Figura 12: Output en terminal por cURL python

Este script de python fue creado utilizando chatGPT y luego fue debugeado manualmente, en la Figura 13, se muestra la query utilizada, donde se indica el cURL que utilizara el código

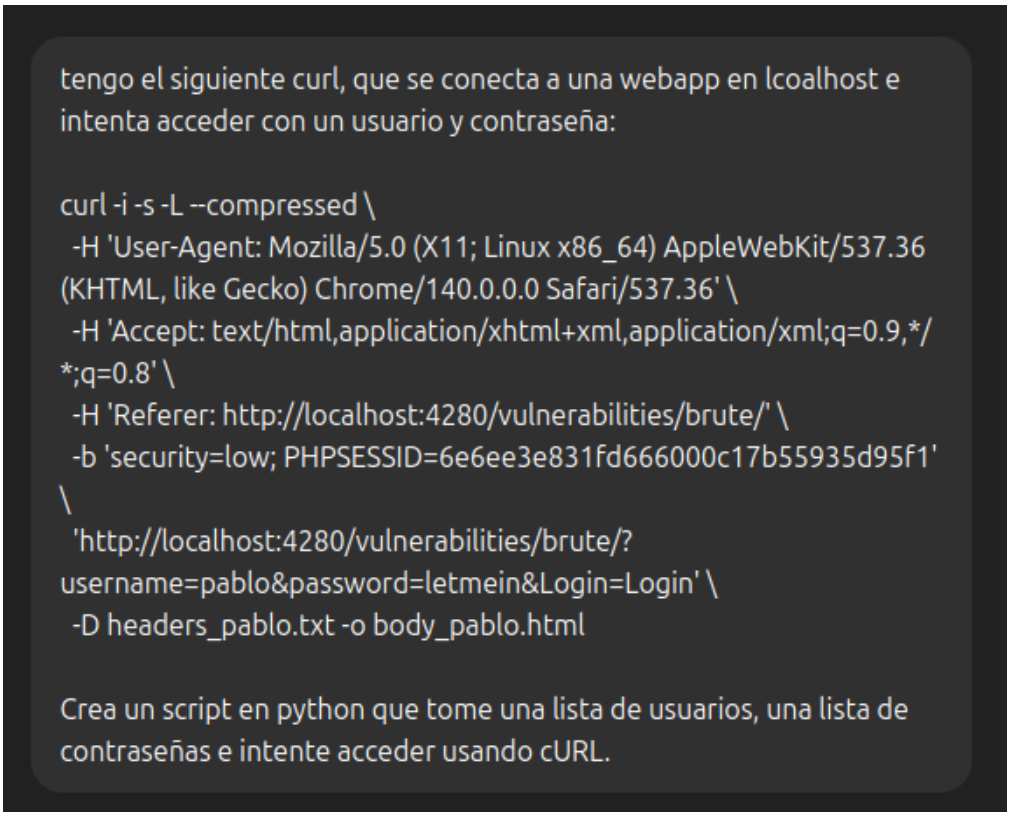

Figura 13: Query enviada a chatGPT para generación de código python

de python, con la información necesaria para que pueda realizar el ataque.

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

En términos de velocidad pura, Hydra fue el más rápido (diseñado para fuerza bruta masiva) con alrededor de 20 intentos por segundo, seguido por `curl`/scripts en línea de comandos con el script en Python que hacían alrededor de 8-10 por segundo, y finalmente Burp Suite (más lento por diseño y controles de interfaz) tomando alrededor de 1 por segundo.

En detección, Hydra es más ruidoso (ráfagas), mientras que Burp/requests pueden configurarse para imitar tráfico humano y ser menos detectables.

2.22. Demuestra 4 métodos de mitigación (investigación)

- Límite de intentos: limitar el número de intentos permitidos por cuenta, bloqueando pasado un umbral.
- Retrasos: agregar tiempos de espera tras cada intento fallido para ralentizar ataques automáticos, por ejemplo 2 segundos por cada intento causaría que el obtener una contraseña por fuerza bruta tome horas, en vez de segundos.
- Autenticación multifactor: requerir un segundo factor (OTP, app o llave) además de la contraseña, esto imposibilita los ataques de fuerza bruta.
- Filtrado por IP: bloquear peticiones sospechosas según IP, velocidad o patrones de acceso.

Estos se pueden observar utilizando el mismo DVWA, ya que al cambiar el nivel de seguridad, se agregan varios de estas medidas, por ejemplo, un tiempo de espera por intento, y en dificultad imposible, un limite de 5 intentos antes de que la app te bloquee, demostrando como estos métodos mitigan e incluso eliminan la posibilidad de ataques de fuerza bruta.

Conclusiones y comentarios

En este trabajo se desplegó DVWA en docker, y se realizaron ataques de fuerza bruta utilizando diferentes aplicaciones, estos siendo, burpsuite, cURL con python y Hydra, cada uno de estos tenia sus propias fortalezas y debilidades, hydra teniendo mayor velocidad, burp entregando mayor control en el envío y recepción de mensajes, y cURL/python permitiendo una fácil replicación y verificación de resultados.

Adicionalmente se detectaron las diferencias claras en los mensajes de respuesta HTTPS, como mensajes de bienvenida/error, imagen de usuario, tamaño del body y marcas temporales, permitiendo automatizar la detección de accesos validos, demostrando lo simple que puede ser atacar una pagina relativamente desprotegida y como es vital el agregar métodos para reducir la capacidad de estos métodos, como limite de intentos, tiempo de espera, y de ser posible 2FA.

Adicionalmente, este laboratorio fue desarrollado siguiendo estos 2 videos como guias:
https://www.youtube.com/watch?v=pSBD9cgwgk0t=1s
https://www.youtube.com/watch?v=FAzRMqNGScst=183s
Y esta pagina web:
https://medium.com/@wyv3rn/creating-easy-proof-of-concept-scripts-with-python-and-curl-5dca489c596b

Finalmente, se utilizó https://www.correctoronline.es/ para correccion ortografica y gramatical, y chatGPT/Gemini para ayudar con la sintaxis de latex.