IES El Caminás

Ciclo de Especialización en Ciberseguridad

Módulo: Incidentes de Ciberseguridad

Título del trabajo: PR_RA3_1_JAVLLUAPA_CIBERSEGURIDAD

Autor(es): Javier Lluesma Aparici

Profesor: Pau Conejero Alberola

Fecha: 28/01/2025

Versión: 1.0

Índice de Contenidos

1. Introducción

El presente proyecto documenta el proceso de diseño, implementación y auditoría de una infraestructura web segura basada en el servidor Apache, utilizando una arquitectura de contenedores Docker. El objetivo fundamental es la creación de una Golden Image que consolide diversas capas de protección, siguiendo una metodología incremental de bastionado (hardening).

A lo largo de las distintas fases, se han integrado medidas de seguridad críticas que abarcan desde la reducción de la superficie de exposición y la gestión de cabeceras HTTP, hasta la implementación de un WAF (Web Application Firewall) con reglas OWASP, la mitigación de ataques de Denegación de Servicio (DoS) y el cifrado de comunicaciones mediante SSL/TLS. Cada etapa hereda las configuraciones de la anterior, garantizando así un sistema robusto, resiliente y alineado con los estándares de seguridad de la industria.

2. Desarrollo de los contenidos

PRÁCTICA 1

Esta primera práctica supone la configuración base del servidor Apache. El objetivo principal es aplicar medidas de bastionado (*hardening*) orientadas a reducir la superficie de exposición y mejorar la seguridad en las comunicaciones desde el despliegue inicial.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

```
root@vera-umate:~# docker pull pps10549287/pps-pr1:latest
latest: Pulling from pps10549287/pps-pr1
Digest: sha256:450f216b3f68c5108d8120da7885e702830a1651635776c06b346cbf19a61217
Status: Image is up to date for pps10549287/pps-pr1:latest
docker.io/pps10549287/pps-pr1:latest
```

Imagen 1. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

```
root@vera-umate:~# docker run -d --name pps-pr1-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr1:latest
1456f7dced501dcedf0c82873d3bd8da30b89fc8882fa37ed11421b159437bcc
root@vera-umate:~# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
                                                                                                                NAMES
1456f7dced50   pps10549287/pps-pr1:latest   "apache2ctl -D FOREG…"   2 seconds ago   Up 2 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp, 0.0.
0.0:8081->443/tcp, [::]:8081->443/tcp   pps-pr1-javlluapa
```

Imagen 2. Lanzar el contenedor y comprobar estado.

Pruebas de Verificación

Tras el despliegue, validamos la configuración de las cabeceras de seguridad en el puerto 8080 mediante curl:

```
root@vera-umate:~# curl -I http://localhost:8080

HTTP/1.1 200 OK
Date: Wed, 28 Jan 2026 20:43:03 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Sun, 25 Jan 2026 11:47:35 GMT
ETag: "29cd-64934f4d523c0"
Accept-Ranges: bytes
Content-Length: 10701
Vary: Accept-Encoding
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src 'self';
Content-Type: text/html
```

Imagen 3. Comprobar cabeceras en el puerto 8080.

Como se observa en la evidencia, el campo **Server** ha sido anonimizado (no revela versiones del software ni del SO) y se confirma la correcta implementación de la política **Content-Security-Policy (CSP)**.

Para la validación del puerto seguro, comprobamos la persistencia de la cabecera **HSTS** en el puerto 8081. Se utiliza el flag -k para omitir la validación de confianza, dado que en esta fase se emplean certificados temporales:

```
root@vera-umate:~# curl -Ik https://localhost:8081
HTTP/1.1 200 OK
Date: Wed, 28 Jan 2026 20:43:33 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Sun, 25 Jan 2026 11:47:35 GMT
ETag: "29cd-64934f4d523c0"
Accept-Ranges: bytes
Content-Length: 10701
Vary: Accept-Encoding
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src 'self';
Content-Type: text/html
```

Imagen 4. Comprobar puerto seguro (HSTS) en puerto 8081.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

```
root@vera-umate:~# docker stop pps-pr1-javlluapa && docker rm pps-pr1-javlluapa
pps-pr1-javlluapa
pps-pr1-javlluapa
root@vera-umate:~#
```

Imagen 5. Limpieza del contendor.

PRÁCTICA 2

Esta imagen hereda la configuración de la P1 (Hardening de Apache) y añade una capa de defensa activa mediante un WAF (Web Application Firewall).

Disponemos de un servidor robusto por fuera (gracias al Hardening de la P1), pero además ahora instalamos un WAF (Web Application Firewall) para que analice el tráfico en tiempo real.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

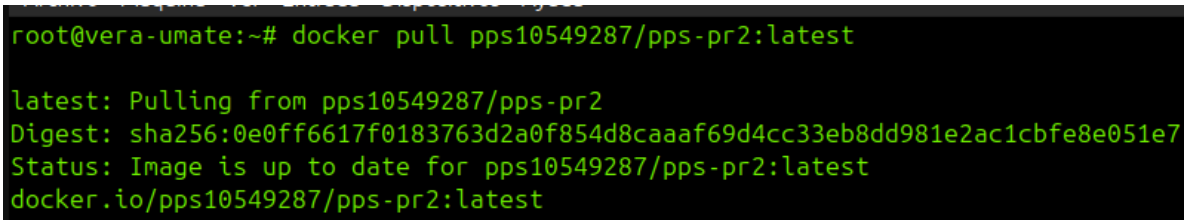

Imagen 6. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

Imagen 7. Lanzar el contenedor y comprobar estado.

Pruebas de Verificación

Tras el despliegue, verificamos que el WAF está operativo y bloqueando amenazas, para ello realizamos pruebas de ataque simulado.

En primer lugar, hacemos una comprobación de bloqueo con curl:

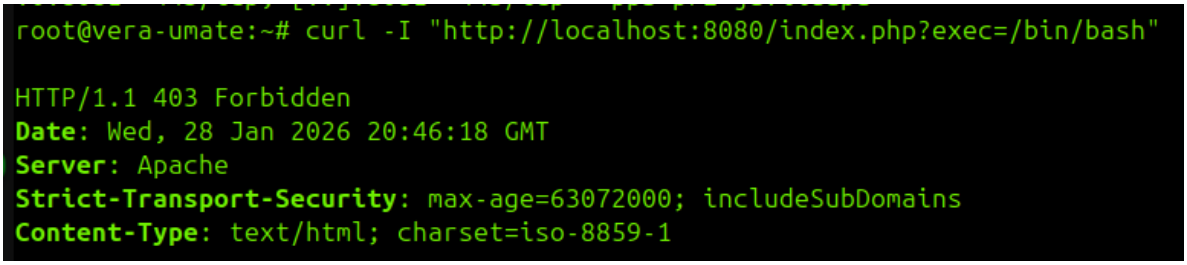

Imagen 8. Comprobar bloqueo (Path Traversal).

La anterior imagen donde se visualiza el código 403 Forbidden recibido indica que el WAF ha detectado un intento de acceso a binarios del sistema a través de la URL y lo ha bloqueado automáticamente.

Seguido, se hace una comprobación de protección en formularios de nuevo con curl:

```
root@vera-umate:~# curl -i -X POST http://localhost:8080/post.php -d "campo=<script>alert(1)</script>"

HTTP/1.1 403 Forbidden
Date: Wed, 28 Jan 2026 20:46:46 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Length: 239
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
```

Imagen 9. Comprobar protección en formularios (POST-XSS).

En la anterior imagen vemos como el servidor bloquea el intento de inyección de script con un código 403, protegiendo la integridad de la aplicación, indicando además que no tenemos acceso a este recurso.

Finalmente, se hace una verificación de que el módulo está cargado con grep:

```
root@vera-umate:~# docker exec pps-pr2-javlluapa apache2ctl -M | grep security

AH00558: apache2: Could not reliably determine the server's fully qualified domai
y to suppress this message
 security2_module (shared)
```

Imagen 10. Módulo security2 cargado.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

```
root@vera-umate:~# docker stop pps-pr2-javlluapa && docker rm pps-pr2-javlluapa
pps-pr2-javlluapa
pps-pr2-javlluapa
root@vera-umate:~#
```

Imagen 11. Limpieza del contenedor.

PRÁCTICA 3

Con esta imagen aplicamos un nivel más de seguridad en la serie, heredando la configuración de la P2 (WAF Base) e integrando el conjunto de reglas más reconocido de la industria.

En la fase anterior instalamos el motor del firewall (ModSecurity), en esta lo configuramos haciendo uso de las reglas estándar de la industria.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

Imagen 12. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

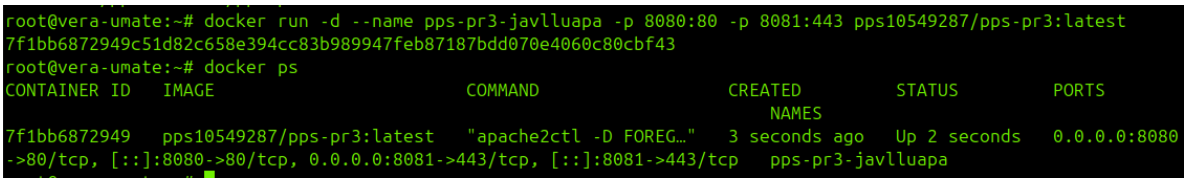

Imagen 13. Lanzar el contenedor y comprobar estado.

Pruebas de Verificación

Tras el despliegue, realizamos pruebas de intrusión en el servidor para verificar la respuesta del firewall y las reglas configuradas.

En primer lugar, ejecutamos un ataque avanzado (Path Traversal), para verificar el bloqueo por parte del WAF con las reglas configuradas:

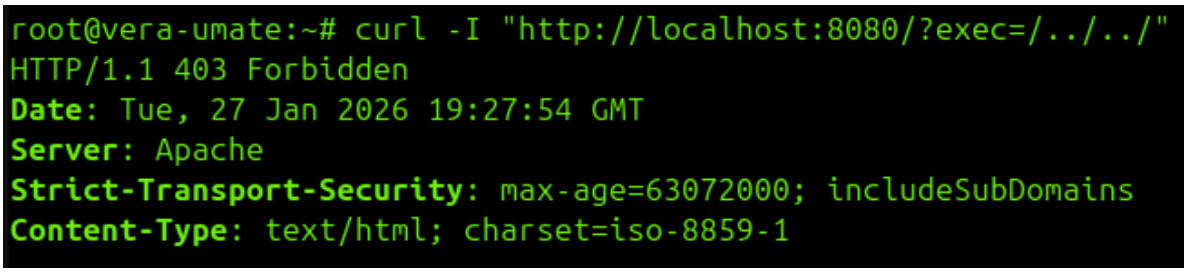

Imagen 14. Comprobar bloqueo (Path Traversal).

Tal y como se visualiza en la anterior imagen, recibimos el código 403 Forbidden confirmando que las reglas de OWASP han identificado el patrón de navegación prohibida por directorios.

De nuevo, ejecutamos otro ataque avanzado (Command Injection), para verificar el bloqueo por parte del WAF con las reglas configuradas:

```
root@vera-umate:~# curl -I "http://localhost:8080/?exec=/bin/bash"
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 19:29:56 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

Imagen 15. Comprobar bloqueo (Command Injection).

En la anterior imagen, se confirma de nuevo que las reglas de OWASP han identificado el patrón de navegación prohibida por directorios.

Visualización de Logs

Ahora procedemos a visualizar con "tail" el fichero "error.log" para ver como el servidor captura ambos ataques en tiempo real:

```
root@vera-umate:~# docker exec pps-pr3-javlluapa tail -f /var/log/apache2/error.log
[Tue Jan 27 19:27:54.054011 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Pattern match "(?i)(?:[/\\\\x5c]|%(?:2(?:f|5(?:2f|5c|c(?:1%259c|0%25af))|%46)|5c|c(?:0%(?:[2aq]f|5c|9v)|1%(?:[19p]c|8s|af))|(?:bg%q|(?:e|f(?:8%8)?0%8)0%80%a)f|u(?:221[56]|EFC8|F025|002f)|%3(?:2(?:%(?:%6|4)6|F)|5%%63)|1u)|0x(?:2f|5c))(?:\\\\.(?:%0[01]|\\\\?)?|\\\\?\\\\.?|%(?:2( ..." at ARGS:exec. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "54"] [id "930100"] [msg "Path Traversal Attack (/../) or (/.../)"] [data "Matched Data: /../ found within ARGS:exec: /../../"] [severity "CRITICAL"] [ver "OWASP_CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-lfi"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/ATTACK-LFI"] [tag "capec/1000/255/153/126"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.054105 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Pattern match "(?:(?:^|[\\\\x5c/;])\\\\.{2,3}[\\\\x5c/;]|[\\\\x5c/;]\\\\.{2,3}[\\\\x5c/;])" at REQUEST_URI_RAW. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "88"] [id "930110"] [msg "Path Traversal Attack (/../) or (/.../)"] [data "Matched Data: /../ found within REQUEST_URI_RAW: /?exec=/../../"] [severity "CRITICAL"] [ver "OWASP_CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-lfi"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/ATTACK-LFI"] [tag "capec/1000/255/153/126"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.054178 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Pattern match "(?:(?:^|[\\\\x5c/;])\\\\.{2,3}[\\\\x5c/;]|[\\\\x5c/;]\\\\.{2,3}[\\\\x5c/;])" at REQUEST_URI_RAW. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "88"] [id "930110"] [msg "Path Traversal Attack (/../) or (/.../)"] [data "Matched Data: /../ found within REQUEST_URI_RAW: /?exec=/../../"] [severity "CRITICAL"] [ver "OWASP_CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-lfi"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/ATTACK-LFI"] [tag "capec/1000/255/153/126"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.054211 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Pattern match "(?:(?:^|[\\\\x5c/;])\\\\.{2,3}[\\\\x5c/;]|[\\\\x5c/;]\\\\.{2,3}[\\\\x5c/;])" at ARGS:exec. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "88"] [id "930110"] [msg "Path Traversal Attack (/../) or (/.../)"] [data "Matched Data: /../ found within ARGS:exec: /../../"] [severity "CRITICAL"] [ver "OWASP_CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-lfi"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/ATTACK-LFI"] [tag "capec/1000/255/153/126"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.054239 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Pattern match "(?:(?:^|[\\\\x5c/;])\\\\.{2,3}[\\\\x5c/;]|[\\\\x5c/;]\\\\.{2,3}[\\\\x5c/;])" at ARGS:exec. [file "/etc/modsecurity/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "88"] [id "930110"] [msg "Path Traversal Attack (/../) or (/.../)"] [data "Matched Data: /../ found within ARGS:exec: /../../"] [severity "CRITICAL"] [ver "OWASP_CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-lfi"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/ATTACK-LFI"] [tag "capec/1000/255/153/126"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.055337 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:blocking_inbound_anomaly_score. [file "/etc/modsecurity/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "233"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 30)"] [ver "OWASP_CRS/4.23.0-dev"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
[Tue Jan 27 19:27:54.063980 2026] [:error] [pid 9:tid 12] [client 172.17.0.1:42982] [client 172.17.0.1] ModSecurity: Warning. Unconditional match in SecAction. [file "/etc/modsecurity/rules/RESPONSE-980-CORRELATION.conf"] [line "98"] [id "980170"] [msg "Anomaly Scores: (Inbound Scores: blocking=30, detection=30, per_pl=30-0-0-0, threshold=5) - (Outbound Scores: blocking=0, detection=0, per_pl=0-0-0-0, threshold=4) - (SQLI=0, XSS=0, RFI=0, LFI=30, RCE=0, PHPI=0, HTTP=0, SESS=0, COMBINED_SCORE=30)"] [ver "OWASP_CRS/4.23.0-dev"] [tag "reporting"] [tag "OWASP_CRS"] [hostname "localhost"] [uri "/"] [unique_id "aXkRumDd34HsLfEgt5REJwAAAAA"]
```

Imagen 16. Logs generados por el primer ataque.

En esta primera imagen, se visualizan los logs generados con relación al primer ataque, podemos ver el ID de la regla de OWASP que ha sido activada y la descripción en detalle del ataque bloqueado.

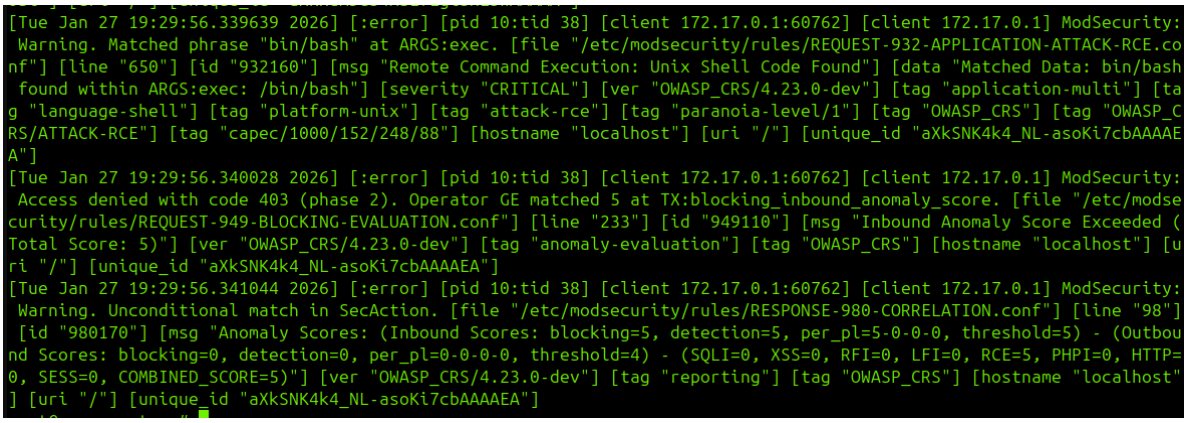

Imagen 17. Logs generados por el segundo ataque.

En la segunda imagen, se visualizan los logs generados esta vez relacionados con el segundo ataque realizado, donde de nuevopodemos ver el ID de la regla de OWASP que ha sido activada y la descripción en detalle del ataque bloqueado.

Verificación de persistencia del bastionado (Capa 1)

Tras verificar el funcionamiento de las reglas configuradas en el WAF, se comprueba la persistencia del bastionado aplicado en la primera práctica con curl:

Imagen 18. Comprobar cabeceras en el puerto 8080.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

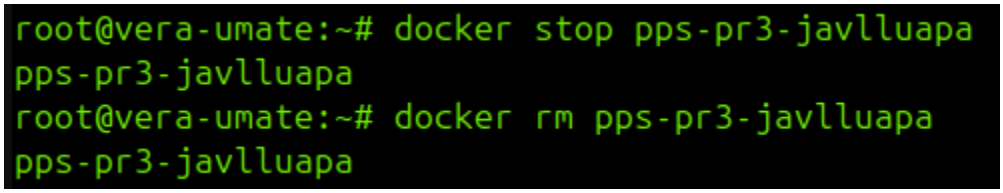

Imagen 19. Limpieza del contenedor.

PRÁCTICA 4

Esta imagen hereda la robustez de la P3 (OWASP CRS) y añade protección contra ataques de Denegación de Servicio (DoS) y fuerza bruta.

En esta etapa, reforzamos la seguridad del servidor contra ataques de denegación de servicio (DoS) y ataques de fuerza bruta. Es decir, preparamos al servidor para reconocer cuándo alguien está intentando saturarlo.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

```
root@vera-umate:~# docker pull pps10549287/pps-pr4:latest
latest: Pulling from pps10549287/pps-pr4
4f4fb700ef54: Pull complete
14093b42a048: Pull complete
1be6e46f6bcb: Pull complete
509bb4fe8a0a: Pull complete
91a50e85643e: Pull complete
5fdb9ec5a2db: Pull complete
613102ef622d: Pull complete
ddc41fd90675: Pull complete
ec97595f112d: Download complete
Digest: sha256:f7444b8ece4d276369b0a2bfc05f0014a8ab39d3dae85a0762b948326c2a0092
Status: Downloaded newer image for pps10549287/pps-pr4:latest
docker.io/pps10549287/pps-pr4:latest
```

Imagen 20. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

```
root@vera-umate:~# docker run -d --name pps-pr4-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr4:latest
a9131f5e0d1ca2f2e4087b05231d1f2b5ba1c6a33c86823f81c4d9b475f3c107
root@vera-umate:~# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
                                                                       NAMES
a9131f5e0d1c   pps10549287/pps-pr4:latest   "apache2ctl -D FOREG…"   3 seconds ago   Up 2 seconds   0.0.0.0:8080->80/
tcp, [::]:8080->80/tcp, 0.0.0.0:8081->443/tcp, [::]:8081->443/tcp   pps-pr4-javlluapa
```

Imagen 21. Lanzar el contenedor y comprobar estado.

Pruebas de Verificación

Para demostrar la efectividad de la protección anti-DoS y la persistencia de las capas anteriores, realizamos las siguientes pruebas:

En primer lugar, utilizamos el script de prueba localizado en /root/test.pl que simula una inundación de peticiones HTTP/1.1 con perl:

```
root@vera-umate:~# docker exec pps-pr4-javlluapa perl /root/test.pl
Iniciando test de inundación (DoS)...
Peticion 0: HTTP/1.1 200 OK
Peticion 1: HTTP/1.1 200 OK
Peticion 2: HTTP/1.1 200 OK
Peticion 3: HTTP/1.1 200 OK
Peticion 4: HTTP/1.1 200 OK
Peticion 5: HTTP/1.1 200 OK
Peticion 6: HTTP/1.1 200 OK
Peticion 7: HTTP/1.1 200 OK
Peticion 8: HTTP/1.1 403 Forbidden
Peticion 9: HTTP/1.1 403 Forbidden
Peticion 10: HTTP/1.1 403 Forbidden
Peticion 11: HTTP/1.1 200 OK
Peticion 12: HTTP/1.1 200 OK
Peticion 13: HTTP/1.1 403 Forbidden
Peticion 14: HTTP/1.1 403 Forbidden
Peticion 15: HTTP/1.1 403 Forbidden
Peticion 16: HTTP/1.1 200 OK
Peticion 17: HTTP/1.1 403 Forbidden
Peticion 18: HTTP/1.1 403 Forbidden
Peticion 19: HTTP/1.1 403 Forbidden
```

Imagen 22. Ejecución del script de prueba de carga con perl.

En la anterior imagen se observa cómo las primeras peticiones devuelven 200 OK y, tras superar el umbral de 5 peticiones, el servidor comienza a responder con 403 Forbidden, confirmando el bloqueo activo de la IP.

Seguidamente, se verifican los registros de bloqueo, comprobando que el módulo ha creado un registro físico de la IP bloqueada en el directorio de logs configurado:

```
root@vera-umate:~# docker exec pps-pr4-javlluapa ls -l /var/log/mod_evasive
total 4
-rw-r--r-- 1 www-data www-data 2 Jan 27 19:50 dos-127.0.0.1
```

Imagen 23. Verificar registros de bloqueo.

Tal como se visualiza en la anterior imagen, la existencia del archivo dos-127.0.0.1 con el propietario www-data confirma que el motor de evasión tiene permisos de escritura y está registrando los ataques detectados.

Ahora, se procede con una prueba de carga masiva con la herramienta apache bench, lanzando 100 peticiones concurrentes desde el host y generando un informe técnico de denegación de servicio:

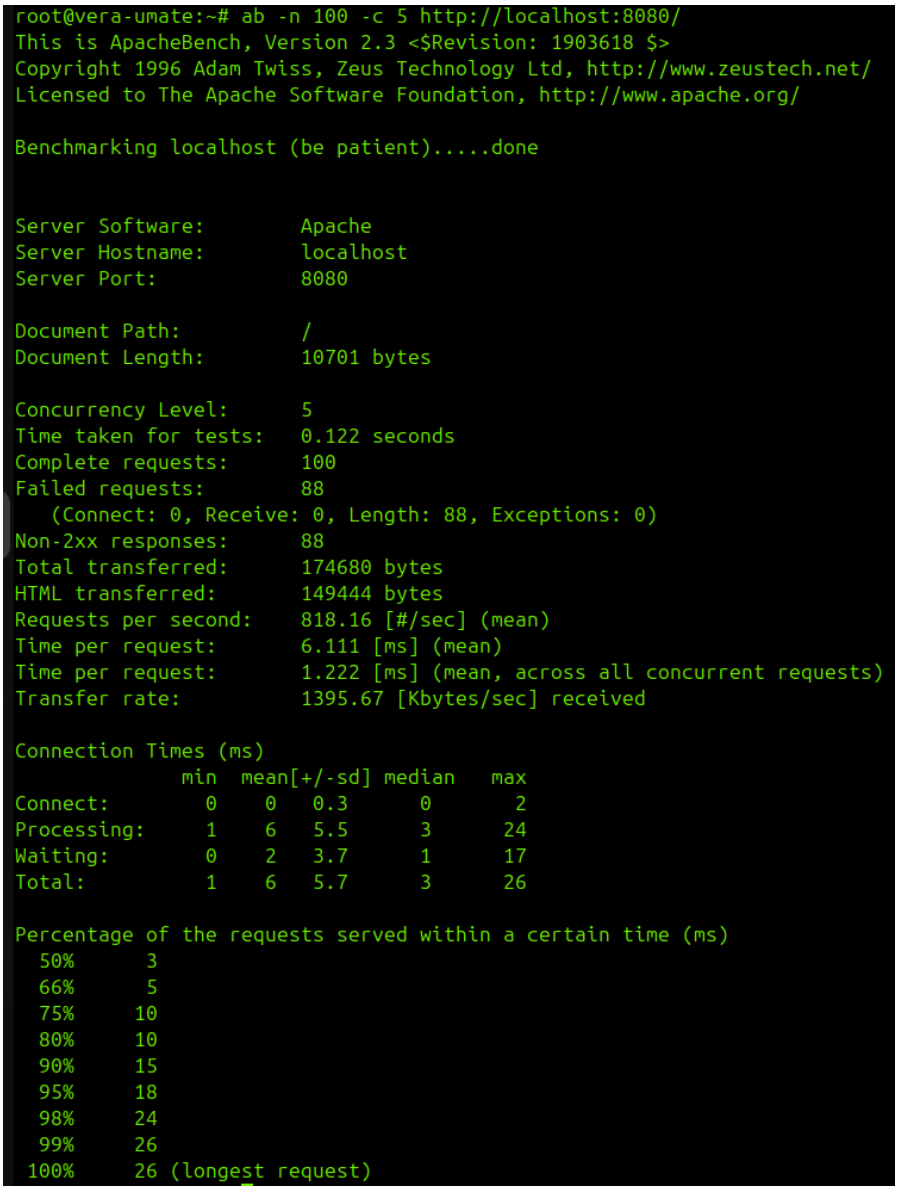

Imagen 24. Informe generado tras la carga masiva con apache bench.

En el informe visualizado en la anterior imagen, se observa que, de las 100 peticiones realizadas, 88 fueron rechazadas (Non-2xx responses). Esto demuestra que, tras las primeras peticiones exitosas, el módulo mod_evasive identificó el exceso de tráfico y activó el bloqueo 403, limitando el consumo de recursos del servidor de forma efectiva.

Verificación de persistencia de capas de seguridad (P1, P2 y P3)

Comprobamos que las reglas de OWASP siguen filtrando ataques de inyección (XSS) a pesar de la nueva capa de protección DoS:

```
root@vera-umate:~# curl -I "http://localhost:8080/?test=<script>alert(1)</script>"
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 19:59:02 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

Imagen 25. Persistencia de capas de seguridad.

El código 403 Forbidden y la presencia de cabeceras de seguridad de la P1 visto en la anterior imagen demuestran que el Hardening y el WAF siguen protegiendo el servidor.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

```
root@vera-umate:~# docker stop pps-pr4-javlluapa
pps-pr4-javlluapa
root@vera-umate:~# docker rm pps-pr4-javlluapa
pps-pr4-javlluapa
```

Imagen 26. Limpieza del contenedor.

PRÁCTICA CERTIFICADOS

En esta práctica se ha implementa la capa de transporte seguro (SSL/TLS) sobre la infraestructura ya blindada en las prácticas anteriores:

En este punto, tomando de nuevo la base de la práctica anterior (que ya traía el WAF y el anti-DDoS) nos centramos en asegurar que toda la comunicación sea privada y esté cifrada.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

```
root@vera-umate:~# docker pull pps10549287/pps-pr-cert:latest
latest: Pulling from pps10549287/pps-pr-cert
526553ce5528: Pull complete
4d0c063c2788: Pull complete
c19cc886f900: Pull complete
55bd148ee710: Download complete
Digest: sha256:678cc91a75eb2008771b163374436f780f06988d3ab4dd2c9dc382b6824efb7e
Status: Downloaded newer image for pps10549287/pps-pr-cert:latest
docker.io/pps10549287/pps-pr-cert:latest
```

Imagen 27. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

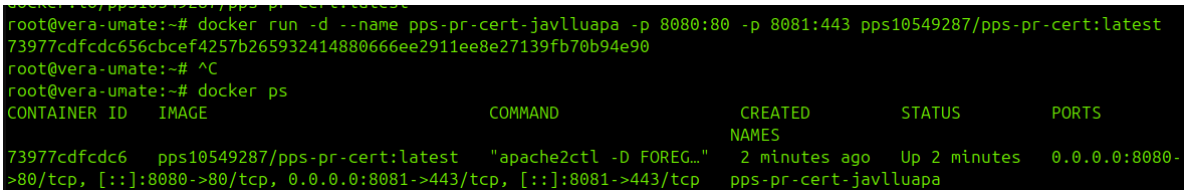

Imagen 28. Lanzar el contenedor y comprobar estado.

A partir de aquí es importante también configurar el entorno local, para que el navegador y las herramientas de test reconozcan el dominio. Para ello es necesario mapear la IP local en el archivo /etc/hosts de la máquina anfitriona:

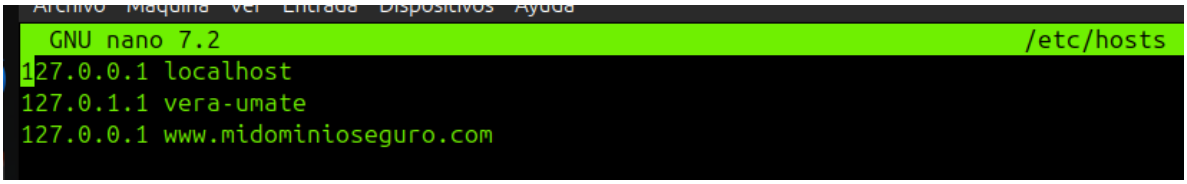

Imagen 29. Configurar entorno local.

Pruebas de Verificación

Para demostrar la redirección de HTTP a HTTPS, comprobamos que el servidor rechaza conexiones inseguras y fuerza el salto al puerto seguro con curl:

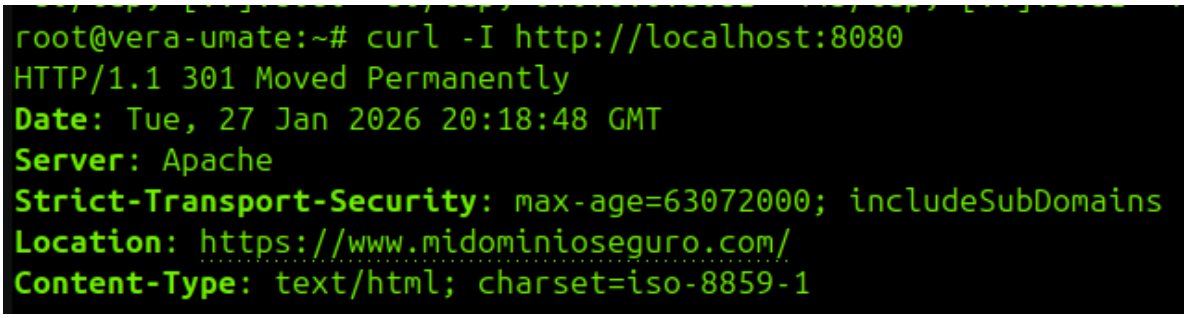

Imagen 30. Redirección de http a https.

Como se visualiza en la anterior imagen, el código 301 Moved Permanently hacia https://www.midominioseguro.com/ confirma la política de transporte seguro.

Tras esto, se procede con la inspección técnica del certificado con curl y grep para validar que el certificado contiene los datos de identidad configurados durante la construcción (Castellón, Seguridad, etc.).

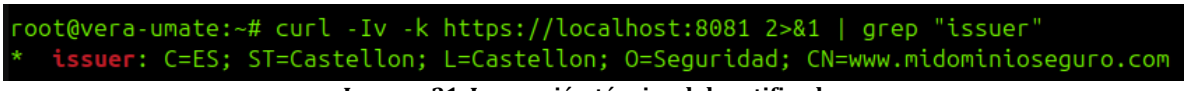

Imagen 31. Inspección técnica del certificado.

En la imagen anterior, la línea issuer: C=ES; ST=Castellon; L=Castellon; O=Seguridad; CN=www.midominioseguro.com confirma la autoría del certificado.

Finalmente se hace una validación visual en el navegador, accediendo a https://www.midominioseguro.com:8081, donde se verifica el aviso de seguridad por certificado autofirmado y se inspeccionan los detalles:

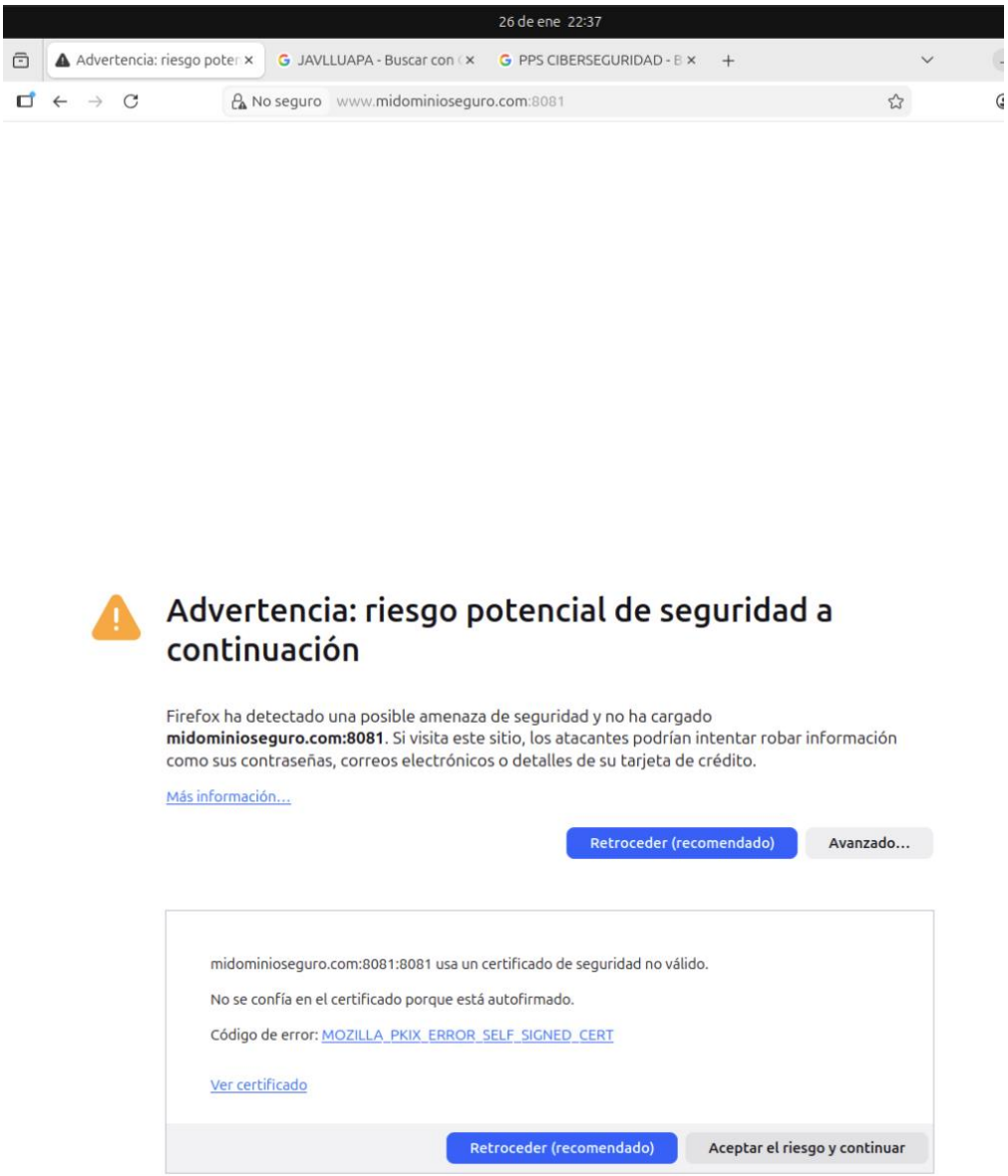

Imagen 32. Aviso de seguridad por certificado autofirmado.

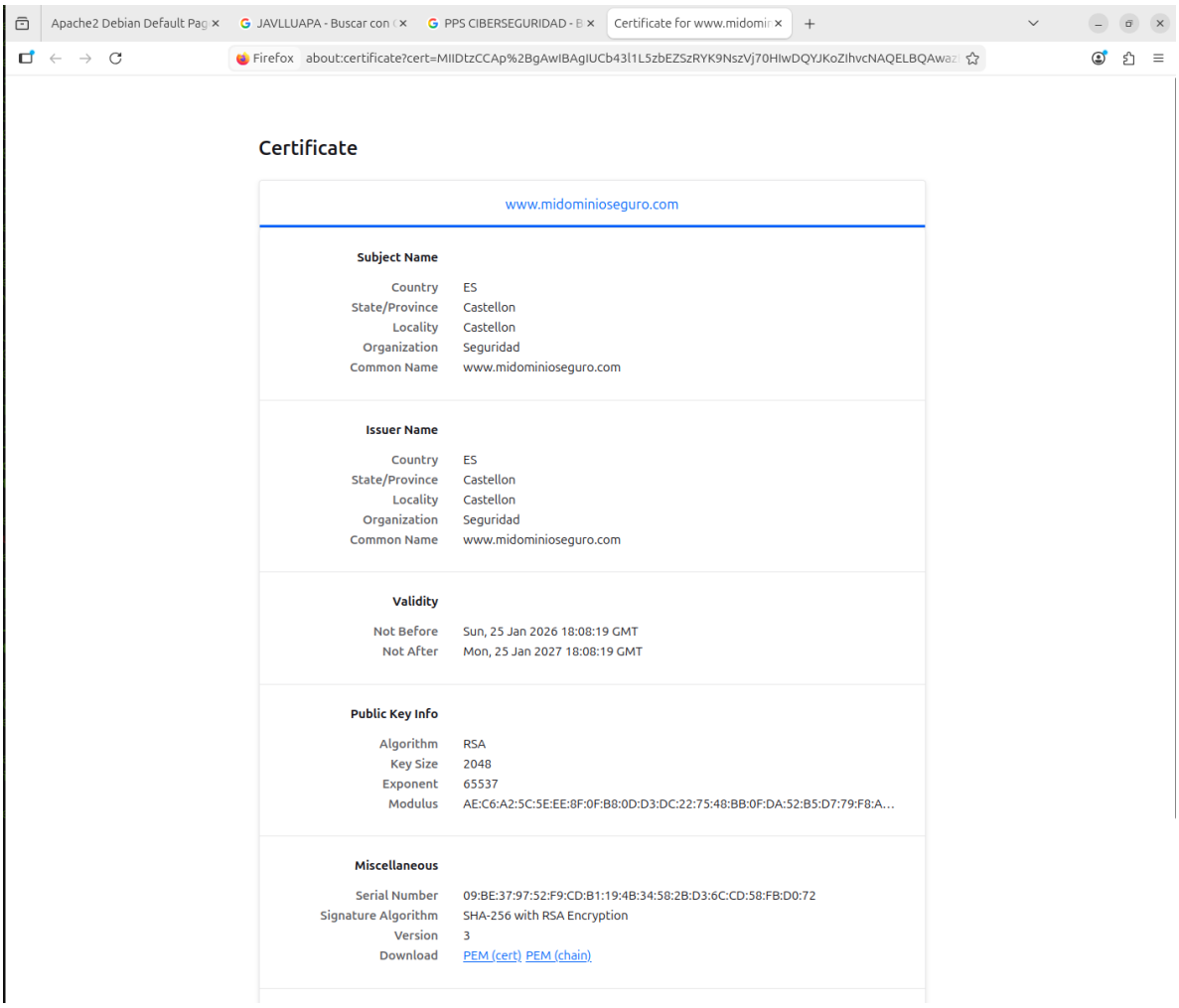

Imagen 33. Detalles del certificado autofirmado.

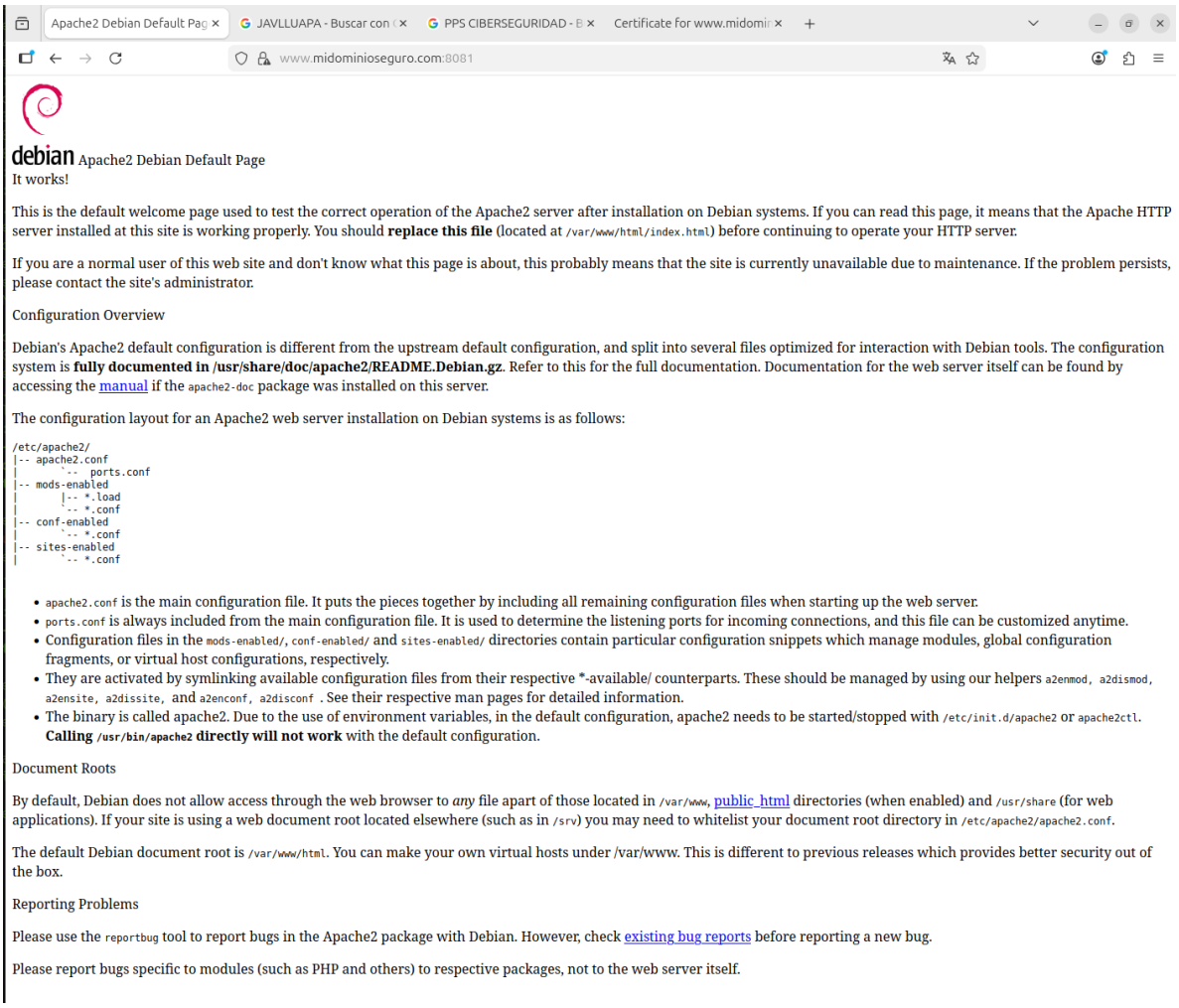

Imagen 34. Página web en funcionamiento con certificado autofirmado.

Verificación de persistencia de capas de seguridad (WAF + DoS + Hardening)

Verificamos que las capas de las prácticas anteriores siguen activas bajo el túnel SSL:

Imagen 35. Persistencia de seguridad.

El código 403 Forbidden de la anterior imagen demuestra que ModSecurity sigue inspeccionando el tráfico una vez descifrado.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

```
root@vera-umate:~# docker stop pps-pr-cert-javlluapa && docker rm -f pps-pr-cert-javlluapa
pps-pr-cert-javlluapa
pps-pr-cert-javlluapa
root@vera-umate:~#
```

Imagen 36. Limpieza del contenedor.

PRÁCTICA GOLD IMAGE

Se ha diseñado como una Gold Image, una imagen de referencia que consolida todas las capas de seguridad configuradas en las prácticas anteriores, sumando protecciones críticas a nivel de sistema operativo, usuario y protocolo.

Básicamente, lo que se ha montado aquí es un proceso de Hardening para asegurar que nuestra instancia de Apache no salga a producción con la configuración por defecto, que suele ser bastante insegura.

Despliegue del entorno

En primer lugar, realizamos el pull de la imagen desde el repositorio oficial en Docker Hub:

```
root@vera-umate:~# docker pull pps10549287/pps-pr-gold:latest
latest: Pulling from pps10549287/pps-pr-gold
31373cc2bedf: Pull complete
23a519f39210: Pull complete
97e2ab2f1475: Pull complete
ea6e338a1ede: Pull complete
21b360c126c6: Pull complete
5e74c7778b68: Download complete
Digest: sha256:3c474ca0892a7a701d3b32e218805e1d3bc28d4ca4e2c72d8670895bbcc15796
Status: Downloaded newer image for pps10549287/pps-pr-gold:latest
docker.io/pps10549287/pps-pr-gold:latest
```

Imagen 37. Descargar imagen del repositorio.

A continuación, desplegamos el contenedor, mapeando el puerto 8080 (HTTP) y 8081 (HTTPS), verificando posteriormente que el servicio se encuentra activo:

```
root@vera-umate:~# docker run -d --name pps-pr-gold-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr-gold:latest
461980525bb83aa27ecba40a2ac85acd86dc8c50f3443a37b3d8d3590382662f
root@vera-umate:~# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS
                                  NAMES
461980525bb8   pps10549287/pps-pr-gold:latest   "apache2ctl -D FOREG…"   2 seconds ago   Up 2 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp,
0.0.0.0:8081->443/tcp, [::]:8081->443/tcp   pps-pr-gold-javlluapa
```

Imagen 38. Lanzar el contenedor y comprobar estado.

Pruebas de Verificación

Para validar que la Gold Image respeta la herencia de las prácticas anteriores y aplica correctamente el endurecimiento final, realizamos una búsqueda recursiva de directivas críticas en todo el árbol de configuración de Apache:

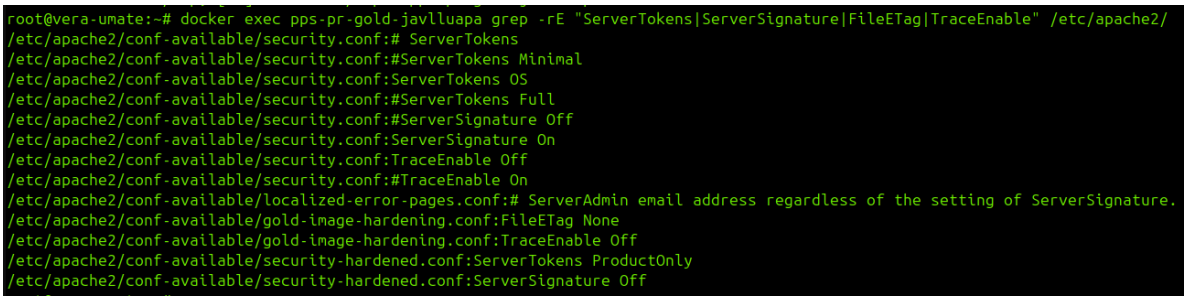

Imagen 39. Verificar herencia y prioridad de capas.

El resultado del escaneo visualizado en la anterior imagen confirma la arquitectura de Defensa en Profundidad mediante la coexistencia de todas las capas:

- Capa de Herencia (P1): Se detectan las directivas en security-hardened.conf (ServerTokens ProductOnly y ServerSignature Off), demostrando que el endurecimiento inicial persiste tras todas las fases del proyecto.
- Capa Gold Image: Se observan las nuevas restricciones en geekflare-hardening.conf (FileETag None y TraceEnable Off), introducidas específicamente en esta fase final.
- Prioridad de Carga: Al utilizar archivos .conf dentro de conf-available, Apache aplica los valores más restrictivos de las capas superiores sobre los valores por defecto del sistema, garantizando la robustez del servidor.

A continuación, se verifican la identidad y cabeceras globales, realizando una petición HTTPS para validar el stack completo de seguridad desde el punto de vista del cliente con curl:

Imagen 40. Verificar identidad y cabeceras globales.

Como se puede ver en la anterior imagen, el servidor responde con el banner oculto (Server: Apache), el mecanismo de transporte seguro activo (Strict-Transport-Security) y las nuevas protecciones contra Clickjacking y XSS. Se confirma que el servidor no responde ante intentos de reconocimiento de versión.

Ahora se verifica la configuración del usuario no privilegiado, auditando los procesos en ejecución para asegurar que un hilo de ejecución comprometido no otorgue acceso de root al atacante con ps y grep:

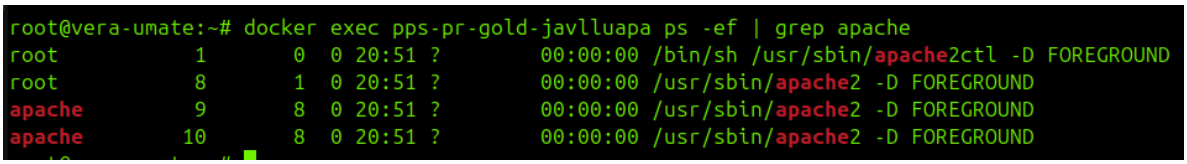

Imagen 41. Verficación de usuario no privilegiado.

En la imagen anterior, se observa cómo el proceso padre corre como root para gestionar los sockets de red, mientras que los procesos trabajadores (workers) que procesan el tráfico externo han cambiado su identidad al usuario apache, cumpliendo el principio de mínimo privilegio.

Finalmente se comprueba la persistencia de las herencias (WAF y SSL) validando que las defensas activas de las prácticas anteriores (ModSecurity) siguen operando correctamente tras el endurecimiento del sistema con curl:

Imagen 42. Persistencia de la herencia WAF y SSL.

Como se ve en la anterior imagen, El servidor devuelve un 403 Forbidden, confirmando que el motor ModSecurity (P2) y las reglas OWASP (P3) siguen inspeccionando el tráfico una vez descifrado por la capa SSL (P5), bloqueando el intento de intrusión.

Finalización

Una vez demostrada la integridad del servicio y el cumplimiento de los requisitos, procedemos a la parada y eliminación del contenedor para optimizar el uso de recursos del sistema.

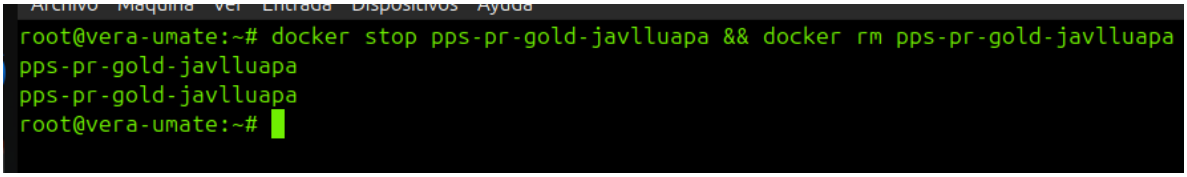

Imagen 43. Limpieza del contenedor.

3. Bibliografía y otras fuentes consultadas

Para el desarrollo de este proyecto se han utilizado recursos oficiales, documentación técnica y guías de referencia en ciberseguridad:

- Repositorio Oficial de Imágenes (Docker Hub):
  - Perfil de Usuario pps10549287 - Almacenamiento y registro de las imágenes generadas en cada práctica.
- Repositorio de Código Fuente (GitHub):

    - PPS JAVLLUAPA - Código y Dockerfiles - Documentación técnica y scripts de construcción.
- Documentación de Referencia (Hardening y Seguridad Web):
    - Segarra, P. (2021). Hardening de Servidor Web. Ciberseguridad-PePS
    - Segarra, P. (2020). Implementación de SSL/TLS. Ciberseguridad-PePS
    - Geekflare. (2023). Apache Web Server Hardening & Security Guide. Geekflare Cybersecurity
- Otras herramientas: Documentación oficial de Apache HTTP Server, ModSecurity Core Rule Set (CRS) y manuales de Docker Engine.

4. Conclusión

Este proyecto demuestra una arquitectura de Defensa en Profundidad real. Cada práctica ha añadido una capa de robustez que la Gold Image final ha consolidado, resultando en un servidor Apache optimizado para resistir ataques modernos.