

# 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

Índice de Contenidos .....	1
1. Introducción .....	2
2. Desarrollo de los contenidos.....	3
PRÁCTICA 1 .....	3
Despliegue del entorno.....	3
Pruebas de Verificación.....	3
Finalización.....	4
PRÁCTICA 2 .....	4
Despliegue del entorno.....	4
Pruebas de Verificación.....	5
Finalización.....	6
PRÁCTICA 3 .....	6
Despliegue del entorno.....	6
Pruebas de Verificación.....	7
Visualización de Logs .....	8
Verificación de persistencia del bastionado (Capa 1).....	9
Finalización.....	9
PRÁCTICA 4 .....	9
Despliegue del entorno.....	10
Pruebas de Verificación.....	10
Verificación de persistencia de capas de seguridad (P1, P2 y P3) .....	12
Finalización.....	13
PRÁCTICA CERTIFICADOS.....	13
Despliegue del entorno.....	13
Pruebas de Verificación.....	14
Verificación de persistencia de capas de seguridad (WAF + DoS + Hardening) .....	17
Finalización.....	17
PRÁCTICA GOLD IMAGE .....	18
Despliegue del entorno.....	18
Pruebas de Verificación.....	18
Finalización.....	20
3. Bibliografía y otras fuentes consultadas .....	20

## BASTIONADO DE APACHE

4. Conclusión .....	21
---------------------	----

## 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
1456f7dcfd501cdedf0c82873d3bd8da30b89fc8882fa37ed11421b159437bcc
root@vera-umate:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
1456f7dcfd50   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:

## BASTIONADO DE APACHE

```
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.

## BASTIONADO DE APACHE

### 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-pr2:latest
latest: Pulling from pps10549287/pps-pr2
Digest: sha256:0e0ff6617f0183763d2a0f854d8caaaf69d4cc33eb8dd981e2ac1cbfe8e051e7
Status: Image is up to date for pps10549287/pps-pr2:latest
docker.io/pps10549287/pps-pr2:latest
```

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:

```
root@vera-umate:~# docker run -d --name pps-pr2-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr2:latest
d8ae299f470a4fce71098b83273757cb06e0bb300ceecf6ae303422e477950cb
root@vera-umate:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
d8ae299f470a        pps10549287/pps-pr2:latest   "apache2ctl -D FORE..."   2 seconds ago    Up 1 second     0.0.0.0:8080->80/tcp, [::]:8080->80/tcp, 0.0.0
.0:8081->443/tcp, [::]:8081->443/tcp   pps-pr2-javlluapa
```

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:

```
root@vera-umate:~# curl -I "http://localhost:8080/index.php?exec=/bin/bash"
HTTP/1.1 403 Forbidden
Date: Wed, 28 Jan 2026 20:46:18 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

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:

## BASTIONADO DE APACHE

```
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 domain name  
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:

## BASTIONADO DE APACHE

```
root@vera-umate:~# docker pull pps10549287/pps-pr3:latest
latest: Pulling from pps10549287/pps-pr3
509bb4fe8a0a: Pull complete
14093b42a048: Pull complete
1be6e46f6bcb: Pull complete
489bbf2d020f: Download complete
Digest: sha256:694ccc3d7abad968db3e6a25b06ca46300e2dacb5cf897e81eae13ede8255e4b
Status: Downloaded newer image for pps10549287/pps-pr3:latest
docker.io/pps10549287/pps-pr3:latest
```

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:

```
root@vera-umate:~# docker run -d --name pps-pr3-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr3:latest
7f1bb6872949c51d82c658e394cc83b989947feb87187bdd070e4060c80cbf43
root@vera-umate:~# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7f1bb6872949 pps10549287/pps-pr3: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-pr3-javlluapa
```

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:

```
root@vera-umate:~# curl -I "http://localhost:8080/?exec=../../../../"
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 19:27:54 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

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:

## BASTIONADO DE APACHE

```
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] ModSecurity: Warning. Pattern match "(?i)(?:[\\\\\\x5c]|%{2:(?:f|5c|c(%:1%259c|0%25af)|%46)|5c|c(%:0%(?:[2aq]f|5c|9v)|1%:[19p]c|8s)af))|(?:bg%q|(?e|f(%:8%80%80%80)f|u(%:221[56]|EFC8|F025|002f)|%3(%:2(%:6|4)6|F)|5%63)|1u)|0x(?:2f|5c))(?:\\\\(.?:%[0j]\\\\\\\\)?\\\\\\\\.?:%(?:(?: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, PHP=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.

## BASTIONADO DE APACHE

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.

```
[Tue Jan 27 19:29:56.339639 2026] [:error] [pid 10:tid 38] [client 172.17.0.1:60762] [client 172.17.0.1] ModSecurity: Warning. Matched phrase "bin/bash" at ARGS:exec. [file "/etc/modsecurity/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf"] [line "650"] [id "932160"] [msg "Remote Command Execution: Unix Shell Code Found"] [data "Matched Data: bin/bash found within ARGS:exec: /bin/bash"] [severity "CRITICAL"] [ver "OWASP CRS/4.23.0-dev"] [tag "application-multi"] [tag "language-shell"] [tag "platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/ATTACK-RCE"] [tag "capec/1000/152/248/88"] [hostname "localhost"] [uri "/"] [unique_id "aXkSNK4k4_NL-asoKi7cbAAAEAA"]  
[Tue Jan 27 19:29:56.340028 2026] [:error] [pid 10:tid 38] [client 172.17.0.1:60762] [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: 5)"] [ver "OWASP CRS/4.23.0-dev"] [tag "anomaly-evaluation"] [tag "OWASP CRS"] [hostname "localhost"] [uri "/"] [unique_id "aXkSNK4k4_NL-asoKi7cbAAAEAA"]  
[Tue Jan 27 19:29:56.341044 2026] [:error] [pid 10:tid 38] [client 172.17.0.1:60762] [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=5, detection=5, per_pl=5-0-0-0, threshold=5) - (Outbound Scores: blocking=0, detection=0, per_pl=0-0-0-0, threshold=4) - (SQLi=0, XSS=0, RFI=0, LFI=0, RCE=5, PHPPI=0, HTTP=0, SESS=0, COMBINED_SCORE=5)"] [ver "OWASP CRS/4.23.0-dev"] [tag "reporting"] [tag "OWASP CRS"] [hostname "localhost"] [uri "/"] [unique_id "aXkSNK4k4_NL-asoKi7cbAAAEAA"]  
[Tue Jan 27 19:29:56.341044 2026] [:error] [pid 10:tid 38] [client 172.17.0.1:60762] [client 172.17.0.1] ModSecurity:
```

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:

```
root@vera-umate:~# curl -I http://localhost:8080  
HTTP/1.1 200 OK  
Date: Tue, 27 Jan 2026 19:38:02 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 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.

```
root@vera-umate:~# docker stop pps-pr3-javlluapa  
pps-pr3-javlluapa  
root@vera-umate:~# docker rm pps-pr3-javlluapa  
pps-pr3-javlluapa
```

Imagen 19. Limpieza del contenedor.

## BASTIONADO DE APACHE

### 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:

## BASTIONADO DE APACHE

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

## BASTIONADO DE APACHE

```
root@vera-umate:~# ab -n 100 -c 5 http://localhost:8080/
This is ApacheBench, Version 2.3 <$Revision: 1903618 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:        Apache
Server Hostname:        localhost
Server Port:            8080

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      5
Time taken for tests:   0.122 seconds
Complete requests:      100
Failed requests:         88
    (Connect: 0, Receive: 0, Length: 88, Exceptions: 0)
Non-2xx responses:      88
Total transferred:       174680 bytes
HTML transferred:        149444 bytes
Requests per second:    818.16 [#/sec] (mean)
Time per request:        6.111 [ms] (mean)
Time per request:        1.222 [ms] (mean, across all concurrent requests)
Transfer rate:           1395.67 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0  0.3      0     2
Processing:     1    6  5.5      3    24
Waiting:        0    2  3.7      1    17
Total:          1    6  5.7      3    26

Percentage of the requests served within a certain time (ms)
  50%    3
  66%    5
  75%   10
  80%   10
  90%   15
  95%   18
  98%   24
  99%   26
100%  26 (longest request)
```

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:

## BASTIONADO DE APACHE

```
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 implementado 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:

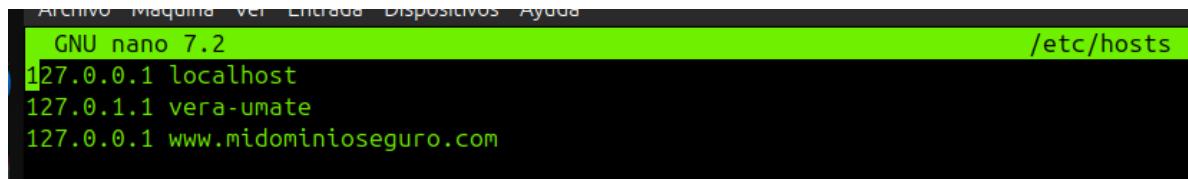
## BASTIONADO DE APACHE

```
root@vera-umate:~# docker run -d --name pps-pr-cert-javlluapa -p 8080:80 -p 8081:443 pps10549287/pps-pr-cert:latest
73977cdfdc656cbcef4257b265932414880666ee291lee8e27139fb70b94e90
root@vera-umate:~# ^C
root@vera-umate:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
73977cdfdc6      pps10549287/pps-pr-cert:latest   "apache2ctl -D FOREG..."   2 minutes ago    Up 2 minutes     0.0.0.0:8080->80/tcp, [::]:8081->443/tcp, [::]:8081->443/tcp   pps-pr-cert-javlluapa

```

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:



```
Archivo Maquina Ver Entrada Dispositivos Ayuda
GNU nano 7.2                                         /etc/hosts
127.0.0.1 localhost
127.0.1.1 vera-umate
127.0.0.1 www.midominioseguro.com
```

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:

```
root@vera-umate:~# curl -I http://localhost:8080
HTTP/1.1 301 Moved Permanently
Date: Tue, 27 Jan 2026 20:18:48 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Location: https://www.midominioseguro.com/
Content-Type: text/html; charset=iso-8859-1
```

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.).

```
root@vera-umate:~# curl -Iv -k https://localhost:8081 2>&1 | grep "issuer"
* issuer: C=ES; ST=Castellon; L=Castellon; O=Seguridad; CN=www.midominioseguro.com
```

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.

## BASTIONADO DE APACHE

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:

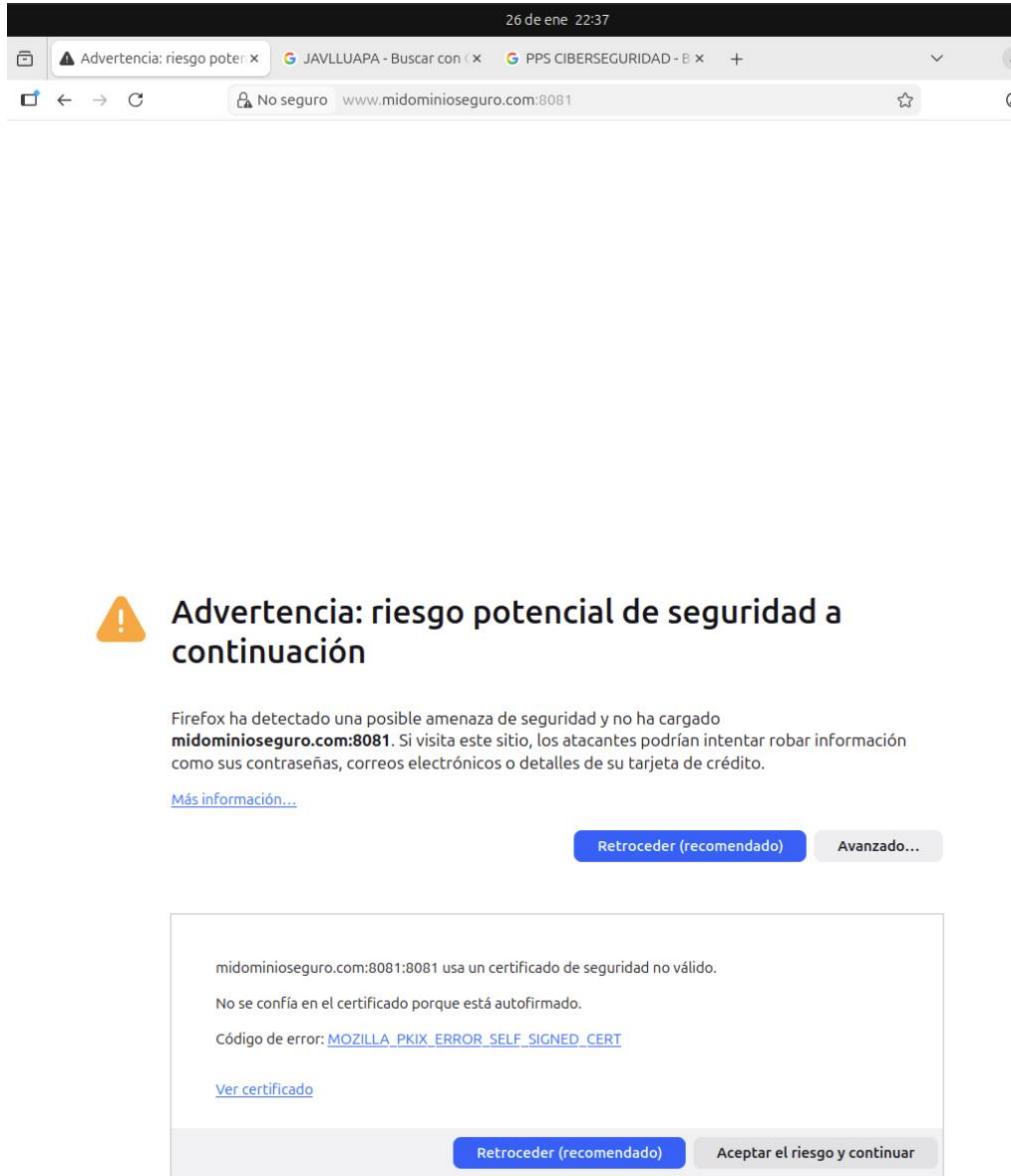


Imagen 32. Aviso de seguridad por certificado autofirmado.

## BASTIONADO DE APACHE

The screenshot shows a Firefox browser window with the title bar "Certificate for www.midominioseguro.com". The main content area displays the certificate details:

**Subject Name**

Country	ES
State/Province	Castellon
Locality	Castellon
Organization	Seguridad
Common Name	www.midominioseguro.com

**Issuer Name**

Country	ES
State/Province	Castellon
Locality	Castellon
Organization	Seguridad
Common Name	www.midominioseguro.com

**Validity**

Not Before	Sun, 25 Jan 2026 18:08:19 GMT
Not After	Mon, 25 Jan 2027 18:08:19 GMT

**Public Key Info**

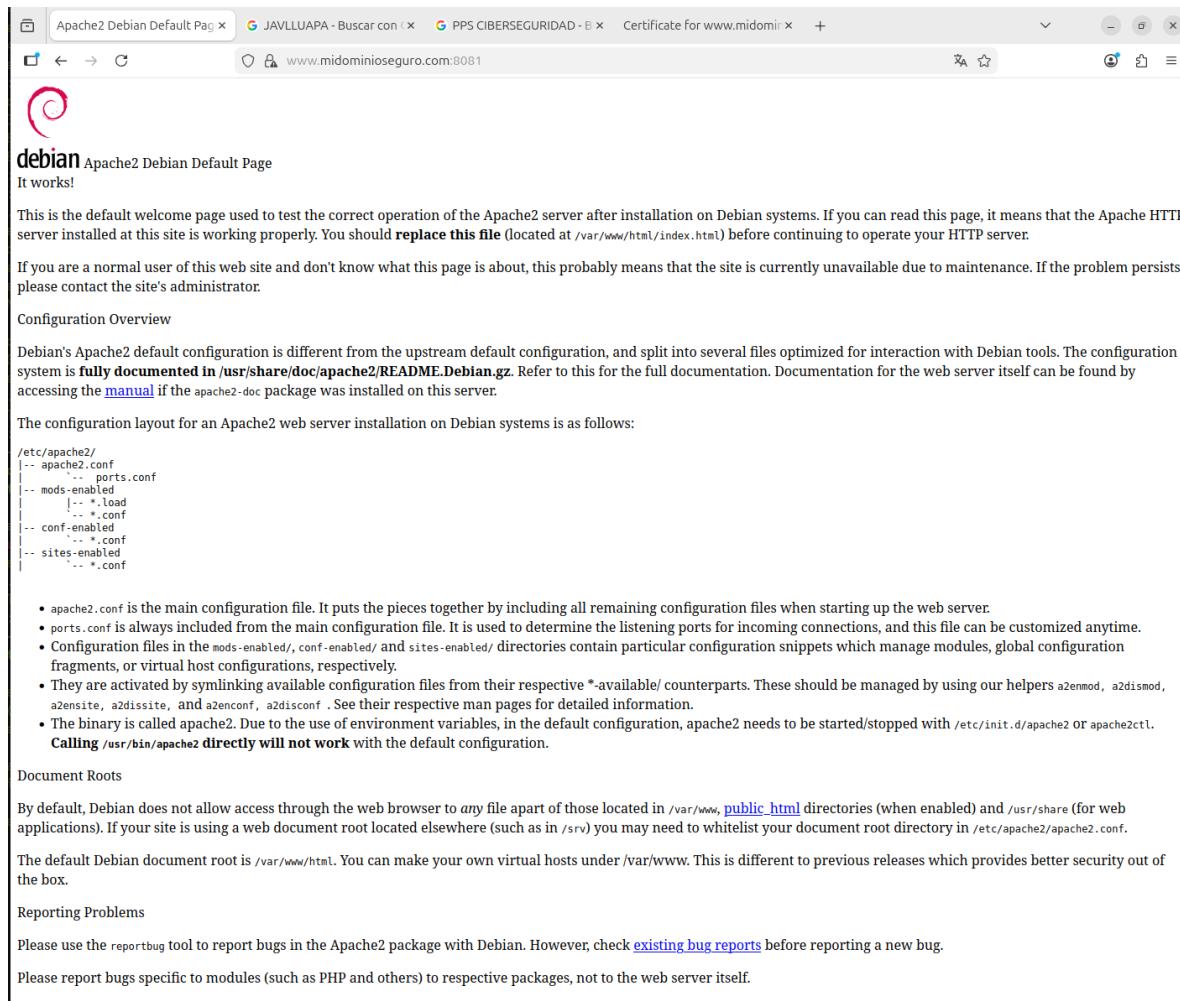
Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	AE:C6:A2:5C:5E:EE:8F:0F:B8:0D:D3:DC:22:75:48:BB:0F:DA:52:B5:D7:79:F8:A...

**Miscellaneous**

Serial Number	09:BE:37:97:52:F9:CD:B1:19:4B:34:58:2B:D3:6C:CD:58:FB:D0:72
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	<a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>

Imagen 33. Detalles del certificado autofirmado.

## BASTIONADO DE APACHE



This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

**Configuration Overview**

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the [manual](#) if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|   |-- mods-enabled
|       |   |-- *.load
|       |   |-- *.conf
|   |-- conf-enabled
|       |   |-- *.conf
|   |-- sites-enabled
|       |   |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` OR `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

**Document Roots**

By default, Debian does not allow access through the web browser to *any* file apart of those located in `/var/www`, [public\\_html](#) directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Debian document root is `/var/www/html`. You can make your own virtual hosts under `/var/www`. This is different to previous releases which provides better security out of the box.

**Reporting Problems**

Please use the `reportbug` tool to report bugs in the Apache2 package with Debian. However, check [existing bug reports](#) before reporting a new bug.

Please report bugs specific to modules (such as PHP and others) to respective packages, not to the web server itself.

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:

```
root@vera-umate:~# curl -I -k "https://localhost:8081/?exec=/bin/bash"
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 20:21:06 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

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.

## BASTIONADO DE APACHE

```
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:

## BASTIONADO DE APACHE

```
root@vera-umate:~# docker exec pps-pr-gold-javlluapa grep -rE "ServerTokens|ServerSignature|FileETag|TraceEnable" /etc/apache2/
/etc/apache2/conf-available/security.conf:# ServerTokens
/etc/apache2/conf-available/security.conf:#ServerTokens Minimal
/etc/apache2/conf-available/security.conf:#ServerTokens OS
/etc/apache2/conf-available/security.conf:#ServerTokens Full
/etc/apache2/conf-available/security.conf:#ServerSignature Off
/etc/apache2/conf-available/security.conf:ServerSignature On
/etc/apache2/conf-available/security.conf:TraceEnable Off
/etc/apache2/conf-available/security.conf:#TraceEnable On
/etc/apache2/conf-available/localized-error-pages.conf:# ServerAdmin email address regardless of the setting of ServerSignature.
/etc/apache2/conf-available/gold-image-hardening.conf:FileETag None
/etc/apache2/conf-available/gold-image-hardening.conf:TraceEnable Off
/etc/apache2/conf-available/security-hardened.conf:ServerTokens ProductOnly
/etc/apache2/conf-available/security-hardened.conf:ServerSignature Off
```

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:

```
root@vera-umate:~# curl -I -k https://www.midominioseguro.com:8081
HTTP/1.1 200 OK
Date: Tue, 27 Jan 2026 20:54:42 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Sun, 25 Jan 2026 11:47:35 GMT
Accept-Ranges: bytes
Content-Length: 10701
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src 'self';
Content-Type: text/html
```

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:

## BASTIONADO DE APACHE

```
root@vera-umate:~# docker exec pps-pr-gold-javlluapa ps -ef | grep apache
root      1      0  0 20:51 ?        00:00:00 /bin/sh /usr/sbin/apache2ctl -D FOREGROUND
root      8      1  0 20:51 ?        00:00:00 /usr/sbin/apache2 -D FOREGROUND
apache    9      8  0 20:51 ?        00:00:00 /usr/sbin/apache2 -D FOREGROUND
apache   10      8  0 20:51 ?        00:00:00 /usr/sbin/apache2 -D FOREGROUND
```

Imagen 41. Verificación 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:

```
root@vera-umate:~# curl -I -k "https://www.midominioseguro.com:8081/?exec=/bin/bash"
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 20:56:41 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

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.

```
Archivo Maquina Ver Entradas Dispositivos Ayuda
root@vera-umate:~# docker stop pps-pr-gold-javlluapa && docker rm pps-pr-gold-javlluapa
pps-pr-gold-javlluapa
pps-pr-gold-javlluapa
root@vera-umate:~#
```

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):
  - o [Perfil de Usuario pps10549287](#) - Almacenamiento y registro de las imágenes generadas en cada práctica.
- Repositorio de Código Fuente (GitHub):

## BASTIONADO DE APACHE

- [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.