

Ciclo de Especialización Ciberseguridad°

RA3.1 Apache Hardening

Objetivos

- Detecta y corrige vulnerabilidades de aplicaciones web analizando su código fuente y configurando servidores web.
- Se han validado las entradas de los usuarios.
- Se han detectado riesgos de inyección tanto en el servidor como en el cliente.
- Se ha gestionado correctamente la sesión del usuario durante el uso de la aplicación.

Resumen

Autor: Izan Rubio Palau

01-2026

IES El Caminás

Obra publicada con [Licencia Creative Commons Reconocimiento Compartir igual 4.0](#)



Índice de Contenidos

1. Introducción	4
2. Actividades	5
2.1 Apache Hardening.....	5
2.1.1 Práctica 1: CSP	5
2.1.2 Práctica 2: Web Application Firewall	6
2.1.3 Práctica 3: OWASP	7
2.1.4 Práctica 4: Evitar ataques DoS.....	8
2.2 Certificados.....	9
2.2.1 Instalar un certificado digital en el servidor Apache y realizar la imagen Docker.....	9
2.3 Apache Hardening Best Practices.....	10
2.3.1 Securizar el servidor Apache y realizar la imagen Docker.....	10
4. Bibliografía	12
5. Conclusión	12

1. Introducción

En el siguiente documento se presentan los resultados obtenidos por el alumno Izan Rubio Palau de las prácticas de la actividad RA3_1 de Puesta en Producción Segura.

Los resultados se muestran conforme se han ido obteniendo junto con una miniguía para poder replicarlo.

2. Actividades

2.1 Apache Hardening

2.1.1 Práctica 1: CSP

Primero realizamos un pull de la imagen:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr1
Using default tag: latest
latest: Pulling from pps10711933/pr1
6c4f88051ada: Pulling fs layer
13a2dadbd3dba: Pulling fs layer
abe4c35235c9: Pulling fs layer
1dd86c8db7d0: Pulling fs layer
26721b9f38d4: Pulling fs layer
2f19ebf7baf5: Pulling fs layer
19182e008e30: Pulling fs layer
Digest: sha256:09d0bf7faa740bbfc2589a5cddaff43fa0fe313e116ff1868d5f64a74ac95b77
Status: Downloaded newer image for pps10711933/pr1:latest
docker.io/pps10711933/pr1:latest
```

Figura 1: Pull de la imagen pr1

Ahora ejecutamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 -p 8081:443 --name PR1 pps10711933/pr1
27ffab0907eb475275fde9893b471e6a0687f028570a18c7123327756d4e7d24
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED      STATUS      PORTS          NAMES
27ffab0907eb   pps10711933/pr1   "apachectl -D FOREGR..."   6 seconds ago   Up 5 seconds   0.0.0.0:8080->80/tcp, [::]:8081->443/tcp   PR1
```

Figura 2: Ejecución del contenedor

Y ahora comprobamos que todo está correcto:

```
izan@DESKTOP-7M6H6JS:~$ curl -Ik https://localhost:8081
HTTP/1.1 200 OK
Date: Tue, 27 Jan 2026 21:22:39 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Thu, 22 Jan 2026 22:04:45 GMT
ETag: "f0-649013a77f940"
Accept-Ranges: bytes
Content-Length: 240
Vary: Accept-Encoding
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com
Content-Type: text/html
```

Figura 3: Comprobar que está bien y se ha seguido la guía

2.1.2 Práctica 2: Web Application Firewall

Descargamos la imagen del contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr2
Using default tag: latest
latest: Pulling from pps10711933/pr2
c9da0a240e28: Pulling fs layer
939ad2bbc047: Pulling fs layer
22b26f03c256: Pulling fs layer
e184debbf8c5: Pulling fs layer
4f4fb700ef54: Pulling fs layer
05ae7da2c64e: Pulling fs layer
Digest: sha256:d90858f0df95993de2fcfe6fd6c34559e8adc5e1a8492f203b8d7fd19d147e8
Status: Downloaded newer image for pps10711933/pr2:latest
docker.io/pps10711933/pr2:latest
```

Figura 4: Descargar la imagen pr2

Ahora ejecutamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 --name PR2 pps10711933/pr2
c11e43f7c91c39df7832a5f2e8c35c4008413ec07a887e260b23152c58c11308
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS
PORTS          NAMES
c11e43f7c91c   pps10711933/pr2   "apachectl -D FOREGR..."   23 seconds ago   Up 23 seconds
0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   PR2
```

Figura 5: Ejecutar el contenedor

Y ahora comprobamos que funciona:

```
izan@DESKTOP-7M6H6JS:~$ curl -i -X POST --data-urlencode "comentario=<script>alert(1)</script>" http://localhost:8080/post.php
HTTP/1.1 403 Forbidden
Date: Tue, 27 Jan 2026 21:33:26 GMT
Server: Apache
Content-Length: 199
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
```

Figura 6: Comprobación de que funciona el WAF

También podemos revisar los logs para ver el aviso:

```
izan@DESKTOP-7M6H6JS:~$ docker exec -it PR2 tail -n 30 /var/log/apache2/modsec_audit.log
--55d21b5b-E--
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>

--55d21b5b-H--
Message: Warning. detected XSS using libinjection. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "56"] [id "941100"] [msg "XSS Attack Detected via libinjection"] [data "Matched Data: XSS data found within ARGS:comentario: <script>alert(1)</script>"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.4"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/152/242"]
Message: Warning. Pattern match "(?:i)<script[^>*>[\\"s\\S]*?" at ARGS:comentario. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "83"] [id "941110"] [msg "XSS Filter - Category 1: Script Tag Vector"] [data "Matched Data: <script> found within ARGS:comentario: <script>alert(1)</script>"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.4"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/152/242"]
Message: Warning. Pattern match "(?:i:(?:<\\w[\"s\\S]*[\"s\\/]|[''])|(?:[\"s\\S]*[\"s\\/]))?(?:on(?:d(?:e(?:vice(?:(:?oriental|m)o)tion|proximity|found|light)|livery(?:success|error)|activate)|n(?:ag(?:e(?:n?:ter|d)|xit)|?:gesture|leav)e|start|drop|over)|op)|i(?:s(?:c(?:hangetimechange ..."))|r(?:ag(?:e(?:n?:ter|d)|xit)|?:gesture|leav)e|start|drop|over)|op)|i(?:s(?:c(?:hangetimechange ..."))|r(?:ag(?:e(?:n?:ter|d)|xit)|?:gesture|leav)e|start|drop|over)|op)" at ARGS:comentario. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "200"] [id "941160"] [msg "NoScript XSS InjectionChecker : HTML Injection"] [data "Matched Data: <script found within ARGS:comentario: <script>alert
```

Figura 7: Revisar los logs para ver el evento de seguridad

2.1.3 Práctica 3: OWASP

Descargamos la imagen pr3:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr3
Using default tag: latest
latest: Pulling from pps10711933/pr3
4f4fb700ef54: Pulling fs layer
703e6d619262: Pulling fs layer
bdab9cb82027: Pulling fs layer
b0c89eafcc7b: Pulling fs layer
51b62e0a6518: Pulling fs layer
3bb890459dfc: Pulling fs layer
4007261b29f8: Pulling fs layer
Digest: sha256:285e3236b1eb9cd103388d5c70980318ee1d5bc3655f0e91920f8d1b77e67145
Status: Downloaded newer image for pps10711933/pr3:latest
docker.io/pps10711933/pr3:latest
```

Figura 8: Descargar la imagen pr3

Ahora ejecutamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 -p 8081:443 --name PR3 pps10711933/pr3
40b6b92d5edb5b9145600967a2e8854702d5c65e60f1f8de29ca2ad54c526cc
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS      PORTS
RTS
40b6b92d5edb   pps10711933/pr3   "apachectl -D FOREGR..."   3 seconds ago   Up 3 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp, 0.0.0.0:8081->443/tcp, [::]:8081->443/tcp   PR3
```

Figura 9: Ejecutar el contenedor

Comprobar que funciona correctamente ModSecurity:

```
izan@DESKTOP-7M6H6JS:~$ curl -k "https://localhost:8081/index.html?testparam=test"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
```

Figura 10: Comprobar que ModSecurity intercepta la petición

2.1.4 Práctica 4: Evitar ataques DoS

Primero descargamos la imagen pr4:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr4
Using default tag: latest
latest: Pulling from pps10711933/pr4
4f4fb700ef54: Pulling fs layer
76c02a830ce8: Pulling fs layer
46a5895c4a01: Pulling fs layer
91c291f3b9ba: Pulling fs layer
Digest: sha256:b8dfe0920dfb6dde3974d0734881dd09f7aabbd3b0ef309a530491e467ec1f2ad
Status: Downloaded newer image for pps10711933/pr4:latest
docker.io/pps10711933/pr4:latest
```

Figura 11: Descargar la imagen pr4

Ahora ejecutamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 --name PR4 pps10711933/pr4
5fb15a1288c81adb47c394d3cebdfeae2f4f25a012d8e4eaa5c127962eea5529
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE       COMMAND          CREATED        STATUS      PORTS
RTS           NAMES
5fb15a1288c8   pps10711933/pr4   "apachectl -D FOREGR..."   5 seconds ago   Up 5 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   PR4
```

Figura 12: Ejecución del contenedor

Una vez el contenedor está en marcha, comprobamos que el módulo mod_evasive bloquea las peticiones masivas:

```
izan@DESKTOP-7M6H6JS:~$ ab -n 3000 -c 50 http://localhost:8080/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
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)
Completed 300 requests
Completed 600 requests
Completed 900 requests
Completed 1200 requests
Completed 1500 requests
Completed 1800 requests
Completed 2100 requests
Completed 2400 requests
Completed 2700 requests
Completed 3000 requests
Finished 3000 requests

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

Document Path:         /
Document Length:      240 bytes

Concurrency Level:    50
Time taken for tests: 0.784 seconds
Complete requests:   3000
Failed requests:      2993
           (Connect: 0, Receive: 0, Length: 2993, Exceptions: 0)
Non-2xx responses:   2993
Total transferred:   1089917 bytes
HTML transferred:    597287 bytes
Requests per second: 3826.03 [#/sec] (mean)
Time per request:    13.068 [ms] (mean)
Time per request:    0.261 [ms] (mean, across all concurrent requests)
Transfer rate:       1357.44 [Kbytes/sec] received
```

Figura 13: Informe de Apache Bench

Como se puede observar, solo han llegado 7 peticiones, las demás han sido bloqueadas.

2.2 Certificados

2.2.1 Instalar un certificado digital en el servidor Apache y realizar la imagen Docker.

Primero nos bajamos la imagen:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr6
Using default tag: latest
latest: Pulling from pps10711933/pr6
9ab06824fd4c: Pulling fs layer
bd67d0e12449: Pulling fs layer
Digest: sha256:d6fd9fa462b05fa2896d9ff768ab747b927bd00a37525357c7c1ec32bf733b8e
Status: Downloaded newer image for pps10711933/pr6:latest
docker.io/pps10711933/pr6:latest
```

Figura 14: Descargar la imagen pr6

Una vez tenemos la imagen, ejecutamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 -p 8081:443 --name PR6 pps10711933/pr6
d76462899b5c38446a484d181b7f221447cc8eb61fdb456074e2a1b736d53075
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE           COMMAND          CREATED         STATUS          NAMES
d76462899b5c   pps10711933/pr6   "apachectl -D FOREGR..."   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   PR6
```

Figura 15: Ejecutar el contenedor con la imagen pr6

Una vez el contenedor está en marcha, ya podemos hacer las diferentes comprobaciones para revisar que se hace la redirección de HTTP a HTTPS y que la conexión es HTTPS y que están los headers de seguridad:

```
izan@DESKTOP-7M6H6JS:~$ curl -Ik http://localhost:8080
HTTP/1.1 301 Moved Permanently
Date: Tue, 27 Jan 2026 21:59:40 GMT
Server: Apache
Location: https://localhost:8081/
Content-Type: text/html; charset=iso-8859-1
```

Figura 16: Verificación de la redirección de HTTP a HTTPS

```
izan@DESKTOP-7M6H6JS:~$ curl -Ik https://localhost:8081
HTTP/1.1 200 OK
Date: Tue, 27 Jan 2026 22:00:17 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Thu, 22 Jan 2026 22:04:45 GMT
ETag: "f0-649013a77f940"
Accept-Ranges: bytes
Content-Length: 240
Vary: Accept-Encoding
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com
Content-Type: text/html
```

Figura 17: Verificar que la conexión es HTTPS y que están los headers de seguridad

2.3 Apache Hardening Best Practices

2.3.1 Securizar el servidor Apache y realizar la imagen Docker.

Vamos a realizar un pull de la imagen final llamada pr7:

```
izan@DESKTOP-7M6H6JS:~$ docker pull pps10711933/pr7
Using default tag: latest
latest: Pulling from pps10711933/pr7
4373b89757e5: Pull complete
8a037d9ce859: Pull complete
10b70dbdba34: Pull complete
5e25c62f6e39: Pull complete
60af9a506517: Download complete
Digest: sha256:1d9db536bd6e05e6cd61d153ce1104c64470c38522d486e3b286fa6d12684a70
Status: Downloaded newer image for pps10711933/pr7:latest
docker.io/pps10711933/pr7:latest
```

Figura 18: Descargar la imagen pr7

Una vez disponemos de la imagen, arrancamos el contenedor:

```
izan@DESKTOP-7M6H6JS:~$ docker run -d --rm -p 8080:80 -p 8081:443 --name PR7 pps10711933/pr7
2adf03662919a4a124fb445a7447f34782d471b7ffc1f3e7f15a01a7051e6d57
izan@DESKTOP-7M6H6JS:~$ docker ps
CONTAINER ID   IMAGE           COMMAND       CREATED      STATUS      PORTS
NAMES
2adf03662919   pps10711933/pr7   "apachectl -D FOREGR..."   5 seconds ago   Up 4 seconds   0.0.0.0
:8080->80/tcp, [::]:8080->80/tcp, 0.0.0.0:8081->443/tcp, [::]:8081->443/tcp   PR7
```

Figura 19: Arrancar el contenedor

Una vez hecho esto, realizamos las comprobaciones para ver que todo es correcto:

```
izan@DESKTOP-7M6H6JS:~$ curl -Ik http://localhost:8080
HTTP/1.1 301 Moved Permanently
Date: Tue, 27 Jan 2026 22:05:50 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Location: https://localhost:8081/
Content-Type: text/html; charset=iso-8859-1
```

Figura 20: Redirección de HTTP a HTTPS

```
izan@DESKTOP-7M6H6JS:~$ curl -Ik https://localhost:8081
HTTP/1.1 200 OK
Date: Tue, 27 Jan 2026 22:06:32 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000; includeSubDomains
Last-Modified: Thu, 22 Jan 2026 22:04:45 GMT
Accept-Ranges: bytes
Content-Length: 240
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 userscripts.example.com
Content-Type: text/html
```

Figura 21: Verificar la conexión HTTPS y los headers de seguridad

```
izan@DESKTOP-7M6H6JS:~$ curl -X PUT -k https://localhost:8081
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
izan@DESKTOP-7M6H6JS:~$ curl -X DELETE -k https://localhost:8081
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
```

Figura 22: Probar que los métodos HTTP inseguros están bloqueados

4. Bibliografía

<https://psegarrac.github.io/Ciberseguridad-PePS/tema3/seguridad/web/2021/03/01/Hardening-Servidor.html>
<https://psegarrac.github.io/Ciberseguridad-PePS/tema1/practicas/2020/11/08/P1-SSL.html>
<https://geekflare.com/cybersecurity/apache-web-server-hardening-security/>
<https://geekflare.com/es/nginx-webserver-security-hardening-guide/>

5. Conclusión

Con la realización de esta práctica he aprendido a securizar un servidor Apache aplicando distintas configuraciones básicas de seguridad. He podido comprobar cómo funcionan medidas como el uso de certificados HTTPS, la protección frente a ataques comunes, el bloqueo de peticiones maliciosas y la limitación de ataques de denegación de servicio.

También he aprendido a utilizar contenedores Docker para desplegar y probar estas configuraciones de forma sencilla, verificando que los cambios funcionan correctamente mediante diferentes pruebas. En general, esta actividad me ha ayudado a entender mejor la importancia de la seguridad en servidores web y cómo aplicar medidas básicas para proteger una aplicación web.