
Bastionado de un servidor web Apache

Implantación de Aplicaciones Web

Índice general

1 Bastionado de un servidor web Apache	1
1.1 Introducción al bastionado (<i>hardening</i>)	1
1.1.1 Principios fundamentales de seguridad	2
1.1.2 Principales amenazas contra servidores web	3
1.2 Arquitectura y archivos de configuración en Debian/Ubuntu	4
1.2.1 Comandos de administración de Apache	4
1.3 Mantenimiento del sistema y permisos del sistema de ficheros	6
1.3.1 Actualización continua del software	6
1.3.2 Usuario y grupo de ejecución sin privilegios	6
1.3.3 Jerarquía de permisos y propiedad en el sistema de ficheros	6
1.3.3.1 Permisos recomendados para la configuración de Apache (/etc/apache2)	7
1.3.3.2 Permisos recomendados para el directorio web (DocumentRoot)	7
1.3.4 Protección contra la ejecución de scripts en directorios de subidas (<i>uploads</i>)	7
1.4 Ocultación de información del servidor (<i>information disclosure</i>)	9
1.4.1 Configuración de ServerTokens y ServerSignature	9
1.4.2 Ocultar la cabecera X-Powered-By	9
1.4.3 Personalización de páginas de error (ErrorDocument)	10
1.4.4 Eliminación de la página por defecto de Apache	10
1.5 Control de acceso y directivas de directorio seguras	11
1.5.1 Política restrictiva global por defecto	11
1.5.2 Configuración restrictiva del DocumentRoot	11
1.5.3 Bloqueo de archivos ocultos, ficheros de configuración y copias de seguridad	12
1.5.4 Configuración de un VirtualHost por defecto (<i>catch-all / blackhole</i>)	12
1.6 Arquitectura MPM y desactivación de módulos innecesarios	14
1.6.1 Selección del MPM: modernización con mpm_event y PHP-FPM	14
1.6.2 Auditoría de módulos activos	14
1.6.3 Módulos potencialmente peligrosos o innecesarios	15
1.6.4 Configuración segura de mod_status	16
1.7 Restricción de métodos HTTP	17
1.7.1 Desactivar el método HTTP TRACE (<i>cross-site tracing</i>)	17
1.7.2 Limitar métodos permitidos a nivel de directorio	17
1.8 Implementación de cabeceras de seguridad HTTP (<i>HTTP security headers</i>)	18
1.8.1 Descripción detallada de las cabeceras de seguridad	19
1.8.2 Verificación de cabeceras con curl	19
1.9 Configuración robusta de SSL/TLS (HTTPS seguro)	20
1.9.1 Habilitación de módulos criptográficos y rendimiento	20

1.9.2	Selección de protocolos y conjuntos de cifrado (<i>cipher suites</i>)	20
1.9.3	Grapado OCSP (<i>OCSP stapling</i>)	20
1.9.4	Habilitación de HTTP/2	21
1.9.5	Redirección obligatoria de HTTP a HTTPS	21
1.10	Protección contra denegación de servicio (DoS) y agotamiento de recursos	22
1.10.1	Ajuste de tiempos de espera (<i>timeouts</i>) y conexiones persistentes	22
1.10.2	Limitación del tamaño de peticiones y cabeceras	22
1.10.3	Mitigación de Slowloris con <code>mod_reqtimeout</code>	22
1.10.4	Prevención de inundaciones con <code>mod_evasive</code>	23
1.11	Cortafuegos de aplicaciones web (WAF): ModSecurity y OWASP CRS	25
1.11.1	Instalación de ModSecurity	25
1.11.2	Activación del motor de reglas y ciclo de vida en producción	25
1.11.3	Integración del OWASP ModSecurity Core Rule Set (CRS)	26
1.11.4	Inspección de registros forenses de ModSecurity	26
1.12	Detección y bloqueo de intrusiones con Fail2Ban	27
1.12.1	Instalación de Fail2Ban	27
1.12.2	Configuración de jaulas (<i>jails</i>) para Apache	27
1.12.3	Comandos de supervisión de Fail2Ban	28
1.13	Gestión, monitorización y auditoría de registros (<i>logs</i>)	30
1.13.1	Formato enriquecido de registros y soporte para proxies inversos	30
1.13.2	Permisos seguros en el directorio de registros	30
1.13.3	Rotación segura con <code>logrotate</code>	30
1.14	Auditoría de seguridad y herramientas de verificación	32
1.14.1	Verificación con <code>curl</code>	32
1.14.2	Auditoría criptográfica con <code>testssl.sh</code>	32
1.14.3	Escaneo de vulnerabilidades web con <code>Nikto</code>	33
1.14.4	Auditoría completa del sistema con <code>Lynis</code>	33
1.15	Lista de verificación de bastionado	34
1.16	Actividades y ejercicios prácticos de laboratorio	37
1.16.1	Ejercicio 1: Auditoría inicial y ocultación de información	37
1.16.2	Ejercicio 2: Restricción de permisos, bloqueo de ficheros y protección de subidas	37
1.16.3	Ejercicio 3: Configuración del VirtualHost catch-all y redirección HTTPS	37
1.16.4	Ejercicio 4: Implementación y validación de cabeceras de seguridad y cookies	38
1.16.5	Ejercicio 5: Mitigación de Slowloris y protección DoS con <code>mod_reqtimeout</code> y <code>mod_evasive</code>	38
1.16.6	Ejercicio 6: Despliegue de ModSecurity (WAF) y bloqueo de inyecciones SQL	38
1.16.7	Ejercicio 7: Detección y bloqueo activo con Fail2Ban	39
2	Referencias	40
3	Licencia	41

1 Bastionado de un servidor web Apache

1.1. Introducción al bastionado (*hardening*)

El **bastionado** o **endurecimiento de sistemas** (*hardening*) es el conjunto de técnicas, directivas de configuración y buenas prácticas aplicadas a un sistema informático para reducir al mínimo su superficie de exposición y mitigar posibles vulnerabilidades frente a ciberataques.

El servidor web [Apache HTTP Server](#) (denominado comúnmente [apache2](#) en distribuciones GNU/Linux de la familia Debian y Ubuntu) es uno de los servidores web más extendidos en Internet. Tras una instalación básica por defecto, el servidor se entrega configurado para maximizar la compatibilidad y facilitar una rápida puesta en marcha. No obstante, esta configuración inicial incluye módulos innecesarios, cabeceras informativas excesivas y directivas permisivas que resultan inseguras en un entorno de producción real.

El objetivo de este documento es proporcionar a los estudiantes y administradores de sistemas una guía profesional, completa y realista para securizar un servidor web Apache. Las medidas descritas están alineadas con los estándares internacionales de referencia de la industria, incluyendo:

- [CIS \(Center for Internet Security\)](#),
- [NIST \(National Institute of Standards and Technology\)](#),
- [OWASP \(Open Web Application Security Project\)](#).



Capas de defensa en profundidad

Estrategia de seguridad multinivel para servidores web Apache en producción

Seguridad integral

- Cortafuegos perimetral y de red**
 - UFW
 - iptables
 - Grupos de seguridad en la nube
- Sistema operativo base**
 - apt upgrade
 - Usuario www-data (sin root)
 - Permisos 755/644
- Servidor web Apache**
 - Módulos mínimos
 - ServerTokens Prod
 - VirtualHost catch-all
- Cifrado robusto en tránsito**
 - TLS 1.3 / 1.2
 - Cifrados PFS
 - OCSP stapling
 - HTTP/2
- Cabeceras HTTP y cookies**
 - HSTS
 - CSP
 - X-Frame-Options
 - Cookies Secure/HttpOnly/SameSite
- Control de subidas y ficheros**
 - Bloqueo de .env y .git
 - Desactivar PHP en uploads/
 - AllowOverride None
- Cortafuegos WAF en capa 7**
 - ModSecurity v2/v3
 - Reglas OWASP CRS
 - Mitigación de inyecciones SQL y XSS
- Prevención de intrusiones (IPS)**
 - Fail2Ban
 - mod_evasive
 - mod_reqtimeout (anti-SlowLoris)
- Monitorización y auditoría**
 - Registros con tiempo de proceso (%D)
 - mod_remoteip
 - Rotación segura (640)

1.1.1. Principios fundamentales de seguridad

A la hora de diseñar y mantener la configuración de seguridad de un servidor web debemos guiarnos por cuatro principios clave:

1. **Principio de menor privilegio (*Principle of Least Privilege*)**: Cada proceso, usuario y servicio debe ejecutarse exclusivamente con los permisos mínimos e indispensables para desempeñar su función. El servidor web nunca debe atender peticiones HTTP/HTTPS bajo el superusuario `root`.
2. **Minimización de la superficie de ataque (*Attack Surface Reduction*)**: Todo módulo, servicio, método HTTP, puerto o funcionalidad que no sea estrictamente imprescindible debe ser deshabilitado o desinstalado.
3. **Defensa en profundidad (*Defense in Depth*)**: La seguridad no debe depender de una única barrera. Se deben articular múltiples capas superpuestas (cortafuegos de red, control en sistema de ficheros, WAF, TLS robusto, cabeceras HTTP restrictivas y sistemas IPS como Fail2ban).
4. **No confiar en la seguridad por oscuridad (*Security by Obscurity*)**: Aunque ocultar versiones e información técnica dificulta el reconocimiento a escáneres automáticos, la verdadera seguridad reside en con-

figuraciones robustas, cifrados actualizados y gestión continua de parches, no en el mero ocultamiento de nombres o rutas.

1.1.2. Principales amenazas contra servidores web

Vector de ataque	Descripción	Impacto
Reconocimiento (<i>fingerprinting</i>)	Identificación de versiones exactas de Apache, sistema operativo y lenguajes backend (PHP).	Facilita la búsqueda y ejecución de <i>exploits</i> públicos (CVEs).
Denegación de servicio (DoS / Slowloris)	Agotamiento de procesos e hilos de trabajo mediante el envío lento o masivo de cabeceras/peticiones.	Caída del servicio e indisponibilidad total para usuarios legítimos.
Fuga de información y <i>path traversal</i>	Acceso no autorizado a ficheros del sistema (<code>/etc/passwd</code>), copias de seguridad (<code>.bak</code> , <code>.sql</code>) o ficheros de configuración (<code>.git</code> , <code>.env</code>).	Filtración de credenciales de base de datos, código fuente y datos confidenciales.
Ejecución de <i>web shells</i> en subidas	Subida y posterior ejecución de scripts maliciosos (<code>.php</code> , <code>.phtml</code>) a través de formularios de subida de archivos.	Ejecución remota de código (RCE) y toma de control del servidor.
Ataques de inyección web (SQLi, XSS, LFI)	Explotación de vulnerabilidades en el código de la aplicación web alojada.	Extracción de bases de datos, alteración de la web y robo de sesiones.
Ataques criptográficos a TLS/SSL	Uso de protocolos obsoletos (SSLv3, TLS 1.0, TLS 1.1) o cifrados vulnerables (RC4, 3DES, CBC).	Interceptación y descifrado de credenciales y datos en tránsito (<i>Man-in-the-Middle</i>).
Clickjacking / MIME-sniffing	Incrustación del sitio en marcos invisibles (<i>iframes</i>) o manipulación del tipo de contenido por el navegador.	Secuestro de clics, phishing y ejecución de scripts camuflados.

1.2. Arquitectura y archivos de configuración en Debian/Ubuntu

En las distribuciones GNU/Linux de la familia Debian/Ubuntu, Apache organiza su configuración en el directorio `/etc/apache2/` siguiendo una arquitectura modular y jerárquica gestionada mediante enlaces simbólicos:

1	<code>/etc/apache2/</code>	—	
2	<code>apache2.conf</code>		# Archivo de configuración maestro principal —
3	<code>ports.conf</code>		# Definición de puertos de escucha (Listen 80, 443) —
4	<code>envvars</code>		# Variables de entorno del proceso (usuario, grupo, logs) —
5	<code>magic</code>		# Firmas para detección de tipos MIME —
6	<code>conf-available/</code>		# Fragmentos de configuración global disponibles —
7	<code>security.conf</code>		# Parámetros de seguridad generales (ServerTokens, etc.) —
8	<code>security-headers.conf</code>		# Cabeceras de seguridad HTTP personalizadas —
9	<code>conf-enabled/</code>		# Enlaces simbólicos a conf-available/ activos —
10	<code>mods-available/</code>		# Módulos compilados disponibles en el sistema —
11	<code>mods-enabled/</code>		# Enlaces simbólicos a mods-available/ activos —
12	<code>sites-available/</code>		# Archivos de configuración de VirtualHosts disponibles —
13	<code>000-default.conf</code>		# VirtualHost HTTP por defecto (o catch-all) —
14	<code>default-ssl.conf</code>		# VirtualHost HTTPS por defecto —
15	<code>sites-enabled/</code>		# Enlaces simbólicos a sites-available/ activos

1.2.1. Comandos de administración de Apache

Para gestionar configuraciones, módulos y sitios web de forma limpia y mantenible, se utilizan las herramientas nativas de Debian/Ubuntu:

1	# Gestión de sitios web (VirtualHosts)
2	<code>sudo a2ensite <nombre-sitio>.conf</code>
3	<code>sudo a2dissite <nombre-sitio>.conf</code>
4	
5	# Gestión de módulos
6	<code>sudo a2enmod <nombre-modulo></code>
7	<code>sudo a2dismod <nombre-modulo></code>
8	
9	# Gestión de fragmentos de configuración global
10	<code>sudo a2enconf <nombre-configuracion></code>
11	<code>sudo a2disconf <nombre-configuracion></code>
12	
13	# Comprobación de sintaxis de la configuración (IMPRESINDIBLE antes de reiniciar)
14	<code>sudo apache2ctl configtest</code>
15	
16	# Recarga (sin corte de servicio) o reinicio del servicio
17	<code>sudo systemctl reload apache2</code>
18	<code>sudo systemctl restart apache2</code>

[!IMPORTANT] Ejecuta siempre `sudo apache2ctl configtest` (o `sudo apachectl -t`) antes de recargar o reiniciar el servicio Apache. Si existe algún error sintáctico o una directiva incompatible, este comando te advertirá de inmediato y evitará una caída inesperada del servicio en producción.

1.3. Mantenimiento del sistema y permisos del sistema de ficheros

1.3.1. Actualización continua del software

El primer paso fundamental en cualquier procedimiento de bastionado consiste en mantener el sistema operativo y el software del servidor actualizados con los últimos parches de seguridad:

```
1 sudo apt update
2 sudo apt upgrade -y
```

Para consultar la versión instalada de Apache y los módulos compilados de forma estática:

```
1 apache2 -v
2 apachectl -V
```

1.3.2. Usuario y grupo de ejecución sin privilegios

Cuando el sistema operativo arranca, el proceso maestro de Apache se inicia como superusuario `root` con el único propósito de enlazar los puertos privilegiados TCP 80 y 443. Inmediatamente después, el servidor genera procesos hijos o hilos de trabajo (*worker threads*) que se ejecutan bajo un usuario sin privilegios.

En Ubuntu y Debian, este usuario y grupo se configuran en `/etc/apache2/envvars`:

```
1 export APACHE_RUN_USER=www-data
2 export APACHE_RUN_GROUP=www-data
```

Podemos verificar que los procesos de atención de peticiones se ejecutan bajo `www-data` mediante el comando `ps`:

```
1 ps aux | grep apache2
```

La salida mostrará un único proceso principal perteneciente a `root` y múltiples procesos trabajadores pertenecientes a `www-data`.

[!CAUTION] Nunca cambies `APACHE_RUN_USER` a `root` ni permitas que los procesos hijos atiendan peticiones con privilegios de administración. Si un atacante lograra ejecutar código remoto a través de una vulnerabilidad web, obtendría acceso total e inmediato como superusuario a toda la máquina.

1.3.3. Jerarquía de permisos y propiedad en el sistema de ficheros

Un error clásico de administración consiste en otorgar permisos `777` o asignar la propiedad total del `DocumentRoot` (`/var/www/html`) al usuario `www-data`. Si el servidor web tiene permisos de escritura en todos los archivos, un atacante que explote una vulnerabilidad en una aplicación PHP podría sobrescribir ficheros `.php`, inyectar puertas traseras (*web shells*) o modificar archivos de configuración.

1.3.3.1. Permisos recomendados para la configuración de Apache (/etc/apache2)

Los archivos de configuración de /etc/apache2 deben pertenecer exclusivamente a `root:root` y no ser modificables por el usuario `www-data`:

```
1 sudo chown -R root:root /etc/apache2
2 sudo find /etc/apache2 -type d -exec chmod 755 {} \;
3 sudo find /etc/apache2 -type f -exec chmod 644 {} \;
```

[!WARNING] **Protección estricta de claves privadas TLS/SSL:** Las claves privadas criptográficas (`.key`) nunca deben tener permisos 644. Si almacenas certificados y claves privadas en `/etc/ssl/private` o dentro de `/etc/apache2/ssl/`, asegúrate de que las claves privadas pertenezcan a `root:ssl-cert` o `root:root` con permisos 600 o 640, de modo que ningún usuario no autorizado del sistema pueda leerlas.

1.3.3.2. Permisos recomendados para el directorio web (DocumentRoot)

1. Los directorios deben tener permisos 755 (`rw-r-xr-x`) o 750 en entornos con múltiples usuarios locales.
2. Los ficheros regulares deben tener permisos 644 (`rw-r--r--`) o 640.
3. El propietario de los archivos debe ser `root` o un usuario de despliegue administrativo (`deploy`), perteneciendo al grupo `www-data`.
4. Únicamente los directorios destinados explícitamente a la subida de archivos (por ejemplo, `uploads/` o `media/`) deben tener permisos de escritura para `www-data`.

```
1 # Establecer propietario y grupo
2 sudo chown -R root:www-data /var/www/html
3
4 # Ajustar permisos estándar para directorios y archivos
5 sudo find /var/www/html -type d -exec chmod 755 {} \;
6 sudo find /var/www/html -type f -exec chmod 644 {} \;
7
8 # Directorios de subida específicos que requieren escritura:
9 sudo chown -R www-data:www-data /var/www/html/uploads
10 sudo chmod -R 755 /var/www/html/uploads
```

1.3.4. Protección contra la ejecución de scripts en directorios de subidas (uploads)

Una de las técnicas más frecuentes de intrusión consiste en aprovechar un formulario de subida de ficheros para subir un script PHP malicioso (por ejemplo `shell.php` o `c99.php`) a la carpeta `uploads/` y solicitarlo vía HTTP para obtener una terminal interactiva.

Aunque la carpeta tenga permisos de escritura para `www-data`, debemos prohibir taxativamente a Apache que ejecute cualquier tipo de script en ese directorio.

Añadimos la siguiente directiva dentro de la configuración del VirtualHost o en `/etc/apache2/conf-available/security.conf`:

```
1 # Bloquear la ejecución de scripts en directorios de subida de archivos
2 <Directory "/var/www/html/uploads">
3     # Desactivar ejecución de scripts CGI e includes
4     Options -ExecCGI -Includes
5
6     # Denegar acceso directo a cualquier extensión ejecutable
7     <FilesMatch "(?i)\.(php|phtml|phar|php3|php4|php5|php7|php8|cgi|pl|py|sh|
8         jsp|asp|exe)$">
9         Require all denied
10    </FilesMatch>
11
12    # Forzar a Apache a tratar cualquier fichero restante como estático
13    ForceType application/octet-stream
14    <IfModule mod_php.c>
15        php_admin_flag engine off
16    </IfModule>
17 </Directory>
```

Con esta regla, incluso si un atacante consigue subir un fichero `.php`, cualquier intento de solicitarlo devolverá un error 403 `Forbidden` y el código nunca será ejecutado por el servidor.

1.4. Ocultación de información del servidor (*information disclosure*)

Por defecto, Apache incluye en la cabecera HTTP `Server` y en los mensajes de error el nombre del servidor, la versión exacta de Apache, el sistema operativo de base y las versiones de los módulos instalados (por ejemplo: `Apache/2.4.58 (Ubuntu) OpenSSL/3.0.13 PHP/8.3.6`). Esta información facilita la fase de reconocimiento (*fingerprinting*) a posibles atacantes.

Configuración	Cabecera HTTP <code>Server</code> emitida	Nivel de exposición
Por defecto (Insegura)	<code>Server: Apache/2.4.58 (Ubuntu) mod_ssl/2.4.58 OpenSSL/3.0.13 PHP/8.3.6</code>	Inseguro / Crítico (expone versiones de Apache, SO, OpenSSL y PHP)
Bastionada (Segura)	<code>Server: Apache</code>	Seguro (mínimo permitido por el estándar HTTP)

1.4.1. Configuración de `ServerTokens` y `ServerSignature`

Para limitar al mínimo la información expuesta, editamos el archivo `/etc/apache2/conf-available/security.conf`:

```

1 # Opciones disponibles para ServerTokens:
2 # Full    -> Apache/2.4.58 (Ubuntu) PHP/8.3.6 mod_ssl/2.4.58
3 # OS      -> Apache/2.4.58 (Ubuntu)
4 # Minor   -> Apache/2.4
5 # Min     -> Apache/2.4.58
6 # Prod    -> Apache (Mínima información permitida por el estándar HTTP)
7
8 ServerTokens Prod
9 ServerSignature Off

```

- **`ServerTokens Prod`**: Configura la cabecera HTTP `Server` para que devuelva únicamente la palabra genérica `Apache`, ocultando la versión exacta, la distribución Linux y los módulos.
- **`ServerSignature Off`**: Desactiva el pie de página que Apache genera en las páginas de error (404, 403, 500) donde tradicionalmente se mostraba la versión, el nombre del host y el puerto de escucha.

1.4.2. Ocultar la cabecera `X-Powered-By`

Los intérpretes de backend como PHP añaden por defecto la cabecera `X-Powered-By: PHP/8.x.x`. Debemos desactivarla en el archivo de configuración `php.ini` correspondiente:

```

1 ; En /etc/php/8.x/apache2/php.ini o /etc/php/8.x/fpm/php.ini
2 expose_php = Off

```

Adicionalmente, podemos forzar a Apache a eliminar cualquier cabecera `X-Powered-By` residual generada por aplicaciones o proxies inversos mediante `mod_headers`:

```
1 # Requiere: sudo a2enmod headers
2 Header always unset X-Powered-By
3 Header unset X-Powered-By
```

1.4.3. Personalización de páginas de error (ErrorDocument)

Las páginas de error estándar de Apache pueden revelar detalles sobre la estructura interna del servidor y rutas del sistema operativo. Es una buena práctica redirigir los errores comunes a documentos HTML genéricos y amigables:

```
1 # En /etc/apache2/conf-available/security.conf o dentro de cada VirtualHost
2 ErrorDocument 400 /errores/400.html
3 ErrorDocument 401 /errores/401.html
4 ErrorDocument 403 /errores/403.html
5 ErrorDocument 404 /errores/404.html
6 ErrorDocument 500 /errores/500.html
```

1.4.4. Eliminación de la página por defecto de Apache

La página predeterminada de instalación (`/var/www/html/index.html`) indica que el servidor acaba de ser instalado y expone la estructura de archivos de Ubuntu/Debian. Debemos sustituirla por el contenido real de la aplicación o eliminarla:

```
1 sudo rm /var/www/html/index.html
```

1.5. Control de acceso y directivas de directorio seguras

El control de acceso al sistema de archivos es una de las defensas más críticas de Apache. Si no se restringe adecuadamente, un atacante podría solicitar archivos del sistema base fuera del directorio web a través de vulnerabilidades de cruce de directorios (*Directory Traversal*).

1.5.1. Política restrictiva global por defecto

En el archivo principal `/etc/apache2/apache2.conf`, debemos asegurarnos de que la directiva para la raíz del sistema de ficheros (`/`) bloquee todo el tráfico por defecto:

```
1 <Directory />
2     Options None
3     AllowOverride None
4     Require all denied
5 </Directory>
```

Esta directiva implementa el principio de denegación por defecto (*deny all by default*): ningún usuario web puede leer absolutamente nada del disco salvo que se autorice explícitamente en un bloque `<Directory>` posterior.

1.5.2. Configuración restrictiva del DocumentRoot

Para el directorio donde residen las páginas web públicas (`/var/www/html`), configuramos permisos selectivos:

```
1 <Directory /var/www/html>
2     Options -Indexes -FollowSymLinks +SymLinksIfOwnerMatch -Includes -ExecCGI
3     AllowOverride None
4     Require all granted
5 </Directory>
```

Analicemos en detalle cada opción de seguridad:

1. **-Indexes:** Desactiva el listado automático de directorios. Si un usuario accede a una carpeta que no contiene un fichero `index.html` o `index.php`, Apache no mostrará el listado de archivos contenidos, devolviendo un error 403 `Forbidden`. Esto previene la enumeración de archivos de la aplicación.
2. **-FollowSymLinks +SymLinksIfOwnerMatch:** Desactiva el seguimiento incondicional de enlaces simbólicos. Con `SymLinksIfOwnerMatch`, Apache solo seguirá un enlace simbólico si el propietario del enlace coincide con el propietario del archivo destino, evitando que un usuario web cree un enlace simbólico que apunte a `/etc/passwd` o `/etc/shadow`.
3. **-Includes:** Desactiva los *Server Side Includes* (SSI), evitando la inyección de directivas del lado del servidor en archivos HTML.
4. **-ExecCGI:** Deshabilita la ejecución de scripts CGI arbitrarios en este directorio.
5. **AllowOverride None:** Deshabilita el uso de archivos `.htaccess`. Los archivos `.htaccess` reducen el rendimiento del servidor (Apache debe inspeccionar cada directorio en cada petición) y permiten

a usuarios locales anular directivas de seguridad. Toda la configuración debe residir en los archivos principales del servidor.

[!WARNING] Si tu aplicación (por ejemplo, WordPress o Drupal) necesita reglas de reescritura de URLs mediante `.htaccess`, limita `AllowOverride` únicamente a lo necesario, como `AllowOverride FileInfo Limit Options=SymLinksIfOwnerMatch` en lugar de `AllowOverride All`.

1.5.3. Bloqueo de archivos ocultos, ficheros de configuración y copias de seguridad

Es habitual que durante el despliegue de aplicaciones queden expuestos archivos sensibles de control de versiones (`.git`), variables de entorno (`.env`), volcados de bases de datos (`.sql`) o copias de seguridad de editores (`.bak`, `.swp`).

Podemos bloquear globalmente el acceso a todos estos patrones añadiendo la siguiente configuración en `/etc/apache2/conf-available/security.conf`:

```
1 # Bloquear acceso a archivos y directorios ocultos (.git, .env, .htaccess)
2 <FilesMatch "^\.>"
3     Require all denied
4 </FilesMatch>
5
6 # Excepción explícita para el reto HTTP-01 de Let's Encrypt / Certbot (.well-known)
7 <Directory "/var/www/html/.well-known">
8     Require all granted
9 </Directory>
10
11 # Bloquear extensiones de archivos sensibles, volcados y copias de seguridad
12 <FilesMatch "(?i)\.(bak|config|sql|fla|psd|ini|log|sh|env|swp|dist|inc|yaml|yml|md|tpl)$">
13     Require all denied
14 </FilesMatch>
```

1.5.4. Configuración de un VirtualHost por defecto (*catch-all* / *blackhole*)

Cuando un atacante escanea rangos de direcciones IP o envía peticiones con una cabecera `Host` manipulada (*Host Header Injection*), Apache recurre por defecto al primer VirtualHost cargado en orden alfabético (generalmente `000-default.conf`).

Para neutralizar escaneos masivos y evitar fugas de información, se recomienda configurar un **VirtualHost por defecto atrapa-todo** (*catch-all*) que capture cualquier petición dirigida directamente a la dirección IP o con un nombre de dominio desconocido, respondiendo de inmediato con un error 403 `Forbidden` o 400 `Bad Request`:

```
1 # /etc/apache2/sites-available/000-catchall.conf
2 <VirtualHost *:80>
3     ServerName default
4     ServerAlias *
5
```

```
6      # Denegar cualquier petición no dirigida a un dominio autorizado
7      <Location />
8          Require all denied
9      </Location>
10
11      ErrorLog ${APACHE_LOG_DIR}/catchall_error.log
12      CustomLog ${APACHE_LOG_DIR}/catchall_access.log combined
13  </VirtualHost>
```

Posteriormente, habilitamos este sitio como el primero de la lista con `sudo a2ensite 000-catchall.conf`. De este modo, los VirtualHosts legítimos solo responderán cuando la cabecera `Host` coincida exactamente con su `ServerName` o `ServerAlias` configurado.

1.6. Arquitectura MPM y desactivación de módulos innecesarios

Cada módulo cargado en Apache consume memoria RAM y añade código que puede contener vulnerabilidades de seguridad potenciales. Siguiendo el principio de minimización, debemos auditar y desactivar todos los módulos que no sean indispensables.

1.6.1. Selección del MPM: modernización con `mpm_event` y PHP-FPM

En versiones modernas de Apache (2.4+) sobre Debian/Ubuntu, la arquitectura recomendada consiste en utilizar el módulo de multiprocesamiento `mpm_event` junto con **PHP-FPM** a través de `mod_proxy_fcgi`, en lugar del módulo obsoleto `mpm_prefork` con `libapache2-mod-php`.

Característica	<code>mpm_prefork</code> + <code>libapache2-mod-php</code> (Obsoleto)	<code>mpm_event</code> + PHP-FPM / <code>proxy_fcgi</code> (Recomendado)
Modelo de concurrencia	Proceso pesado dedicado por conexión	Hilos asíncronos ligeros sin bloqueo
Consumo de memoria	Muy alto (duplica el intérprete PHP en cada proceso)	Mínimo y optimizado
Resistencia DoS/Slowloris	Muy baja (agotamiento rápido de procesos)	Muy alta (descriptores de socket no bloqueantes)
Aislamiento de seguridad	PHP se ejecuta en el mismo espacio que Apache	PHP aislado en un proceso/socket Unix independiente
Soporte HTTP/2	No compatible	Soporte nativo completo

Para migrar a `mpm_event` y PHP-FPM:

```
1 # Deshabilitar módulo PHP clásico y MPM prefork
2 sudo a2dismod php8.x mpm_prefork
3
4 # Habilitar MPM event, proxy FastCGI y configuración PHP-FPM
5 sudo a2enmod mpm_event proxy proxy_fcgi setenvif
6 sudo a2enconf php8.x-fpm
7 sudo systemctl restart apache2
```

1.6.2. Auditoría de módulos activos

Para listar todos los módulos compilados y cargados actualmente en memoria:

```
1 apache2ctl -M
```

1.6.3. Módulos potencialmente peligrosos o innecesarios

A continuación se detallan los módulos que deben revisarse o desactivarse en la mayoría de entornos de producción:

Módulo	Función	Riesgo de seguridad	Acción recomendada
<code>mod_autoindex</code>	Genera listados HTML de directorios.	Facilita la enumeración de archivos y carpetas del servidor.	Desactivar (<code>a2dismod autoindex</code>).
<code>mod_info</code>	Muestra la configuración completa del servidor en <code>/server-info</code> .	Fuga crítica de información y rutas internas.	Desactivar en producción (<code>a2dismod info</code>).
<code>mod_status</code>	Muestra estadísticas y peticiones activas en <code>/server-status</code> .	Fuga de IPs de clientes, URLs consultadas y carga del sistema.	Restringir por IP o desactivar.
<code>mod_userdir</code>	Mapea URLs tipo <code>/~usuario/</code> al directorio <code>/home/usuario/public_html</code> .	Facilita la enumeración de usuarios locales del sistema Linux.	Desactivar (<code>a2dismod userdir</code>).
<code>mod_cgi/mod_cgid</code>	Ejecuta scripts CGI tradicionales en <code>/cgi-bin/</code> .	Riesgo de ejecución de código y consumo excesivo de recursos.	Desactivar si se usa PHP-FPM (<code>a2dismod cgi cgid</code>).
<code>mod_include</code>	Ejecuta directivas Server Side Includes (SSI).	Riesgo de inyección de código del lado del servidor.	Desactivar si no se usa (<code>a2dismod include</code>).
<code>mod_dav/mod_dav_fs</code>	Protocolo WebDAV para gestión remota de archivos.	Riesgo de subida y modificación no autorizada de ficheros.	Desactivar si no se requiere (<code>a2dismod dav dav_fs</code>).

Para desactivar los módulos innecesarios de forma segura:

```
1 sudo a2dismod autoindex status info userdir cgi cgid include dav dav_fs
2 sudo systemctl restart apache2
```

[!CAUTION] Si utilizas el módulo `mod_proxy`, verifica que la directiva `ProxyRequests Off` esté establecida a nivel global. Dejar `ProxyRequests On` convierte tu servidor en un **Proxy Abierto (Open Proxy)**, permitiendo a ciberdelincuentes utilizar tu infraestructura como pasarela anónima para lanzar ataques a terceros.

1.6.4. Configuración segura de mod_status

Si requieres mantener activo `mod_status` para herramientas de monitorización interna (como Prometheus, Zabbix o Nagios), debes restringir estrictamente el acceso para que solo responda a peticiones desde `localhost` o desde la IP específica del servidor de monitorización:

```
1 # En /etc/apache2/mods-available/status.conf
2 <IfModule mod_status.c>
3     <Location /server-status>
4         SetHandler server-status
5         Require local
6         Require ip 192.168.1.50
7     </Location>
8
9     ExtendedStatus On
10 </IfModule>
```

1.7. Restricción de métodos HTTP

El protocolo HTTP define diversos métodos de petición ([GET](#), [POST](#), [HEAD](#), [PUT](#), [DELETE](#), [OPTIONS](#), [TRACE](#), [PATCH](#), [CONNECT](#)). La inmensa mayoría de sitios web estándar únicamente necesitan procesar peticiones [GET](#), [POST](#) y [HEAD](#).

1.7.1. Desactivar el método HTTP TRACE (*cross-site tracing*)

El método [TRACE](#) se diseñó para depuración de red: el servidor web devuelve al cliente el contenido exacto de la petición recibida. Sin embargo, este comportamiento permite a un atacante que ejecute código JavaScript (vía XSS) eludir la protección de cookies marcadas como [HttpOnly](#) mediante técnicas de *Cross-Site Tracing* (XST).

Para desactivar el método [TRACE](#) en todo el servidor, editamos `/etc/apache2/conf-available/security.conf`:

```
1 TraceEnable Off
```

1.7.2. Limitar métodos permitidos a nivel de directorio

Podemos restringir de forma estricta los métodos admitidos en el [DocumentRoot](#) mediante la directiva `<LimitExcept>`:

```
1 <Directory /var/www/html>
2     # Si la aplicación web utiliza APIs o peticiones CORS, incluye OPTIONS
3     <LimitExcept GET POST HEAD OPTIONS>
4         Require all denied
5     </LimitExcept>
6 </Directory>
```

Si la aplicación ofrece una API REST que requiere métodos como [PUT](#), [DELETE](#) o [PATCH](#), dichos métodos deben autorizarse exclusivamente dentro del bloque `<Location /api>` correspondiente, manteniéndolos bloqueados en el resto del sitio:

```
1 <Location /api>
2     <LimitExcept GET POST PUT DELETE PATCH OPTIONS HEAD>
3         Require all granted
4     </LimitExcept>
5 </Location>
```

1.8. Implementación de cabeceras de seguridad HTTP (*HTTP security headers*)

Las cabeceras de seguridad HTTP son instrucciones que el servidor web envía en la respuesta HTTP para obligar al navegador del cliente a activar mecanismos de protección integrados. Su correcta implementación mitiga ataques tan extendidos como el *Clickjacking*, *Cross-Site Scripting* (XSS), inyección de datos y *MIME-sniffing*.

Para gestionar cabeceras en Apache, es imprescindible habilitar el módulo `mod_headers`:

```
1 sudo a2enmod headers
```

A continuación, crearemos un archivo de configuración dedicado `/etc/apache2/conf-available/security-headers.conf`:

```
1 # /etc/apache2/conf-available/security-headers.conf
2 <IfModule mod_headers.c>
3     # 1. HTTP Strict Transport Security (HSTS) - Solo sobre conexiones HTTPS
4     <If "%{HTTPS} == 'on'">
5         Header always set Strict-Transport-Security "max-age=31536000;
6             includeSubDomains; preload"
7     </If>
8
9     # 2. Protección contra Clickjacking
10    Header always set X-Frame-Options "SAMEORIGIN"
11
12    # 3. Protección contra MIME-Sniffing
13    Header always set X-Content-Type-Options "nosniff"
14
15    # 4. Política de Referrer
16    Header always set Referrer-Policy "strict-origin-when-cross-origin"
17
18    # 5. Restricción de APIs y Hardware del Navegador (Permissions-Policy)
19    Header always set Permissions-Policy "camera=(), microphone=(), geolocation
20        =(), payment=(), usb=()"
21
22    # 6. Aislamiento de Origen Cruzado (COOP y CORP)
23    Header always set Cross-Origin-Opener-Policy "same-origin"
24    Header always set Cross-Origin-Resource-Policy "same-origin"
25
26    # 7. Content Security Policy (CSP)
27    Header always set Content-Security-Policy "default-src 'self'; script-src '
28        self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src
29        'self'; connect-src 'self'; frame-ancestors 'self'; form-action 'self';
30        base-uri 'self';"
31
32    # 8. Desactivar el filtro XSS obsoleto y propenso a vulnerabilidades en
33    navegadores antiguos
34    Header always set X-XSS-Protection "0"
35
36    # 9. Asegurar Cookies de Sesión (Inyectar Secure, HttpOnly y SameSite)
37    Header edit Set-Cookie ^(.*)$ "$1; Secure; HttpOnly; SameSite=Lax"
38 </IfModule>
```

Habilitamos la configuración y recargamos Apache:

```
1 sudo a2enconf security-headers
2 sudo apache2ctl configtest
3 sudo systemctl reload apache2
```

1.8.1. Descripción detallada de las cabeceras de seguridad

Cabecera HTTP	Propósito y mitigación de seguridad
Strict-Transport-Security (HSTS)	Obliga al navegador a usar siempre HTTPS durante el tiempo indicado (31536000s = 1 año), impidiendo ataques de degradación SSL Stripping.
X-Frame-Options	Evita que el sitio sea incrustado en <code><iframe></code> o <code><frame></code> externos, neutralizando por completo ataques de Clickjacking.
X-Content-Type-Options	Fija el valor <code>nosniff</code> , impidiendo que el navegador intente adivinar el tipo MIME de un fichero y ejecute scripts camuflados como imágenes.
Referrer-Policy	Controla qué información de la URL de origen se envía en la cabecera <code>Referer</code> al navegar hacia enlaces externos, protegiendo tokens y rutas sensibles.
Permissions-Policy	Desactiva el acceso a APIs de hardware sensibles del cliente como la cámara, el micrófono, el sensor de geolocalización o pasarelas de pago.
Content-Security-Policy (CSP)	Especifica exactamente de qué orígenes autorizados el navegador puede cargar scripts, estilos, imágenes y fuentes. Máxima protección frente a XSS e inyecciones.
SameSite / Secure / HttpOnly en Set-Cookie	Protege las cookies de sesión contra robo mediante XSS (<code>HttpOnly</code>), interceptación en tránsito (<code>Secure</code>) y ataques CSRF (<code>SameSite=Lax</code>).

1.8.2. Verificación de cabeceras con curl

Podemos comprobar rápidamente desde el terminal que nuestro servidor emite todas las cabeceras correctamente:

```
1 curl -I https://midominio.com
```

1.9. Configuración robusta de SSL/TLS (HTTPS seguro)

El cifrado en tránsito mediante el protocolo **HTTPS** es obligatorio en la web actual. No obstante, habilitar HTTPS no es suficiente por sí solo: es necesario garantizar que la configuración criptográfica utilice únicamente protocolos y algoritmos seguros.

1.9.1. Habilitación de módulos criptográficos y rendimiento

```
1 # Módulos SSL, almacenamiento en caché de sesiones y soporte para HTTP/2
2 sudo a2enmod ssl
3 sudo a2enmod socache_shmcb
4 sudo a2enmod http2
```

1.9.2. Selección de protocolos y conjuntos de cifrado (*cipher suites*)

Debemos deshabilitar de forma categórica protocolos vulnerables como SSLv2, SSLv3, TLS 1.0 y TLS 1.1, permitiendo exclusivamente **TLS 1.2** y **TLS 1.3**.

Editamos el archivo global de configuración SSL `/etc/apache2/mods-available/ssl.conf` o la configuración de nuestro VirtualHost HTTPS:

```
1 # Protocolos permitidos (Solo TLS 1.2 y TLS 1.3)
2 SSLProtocol -all +TLSv1.2 +TLSv1.3
3
4 # Cifrados seguros para TLS 1.2 (con secreto perfecto hacia adelante - PFS)
5 SSLCipherSuite ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
   ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
   POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
   AES256-GCM-SHA384
6
7 # Preferencia de cifrados (en OpenSSL moderno con TLS 1.3 se recomienda off)
8 SSLHonorCipherOrder off
9
10 # Desactivar compresión SSL para neutralizar ataques CRIME / BREACH
11 SSLCompression off
12
13 # Desactivar Session Tickets si no son estrictamente necesarios
14 SSLSessionTickets off
```

1.9.3. Grapado OCSP (*OCSP stapling*)

El **grapado OCSP** (*OCSP stapling*) mejora significativamente el rendimiento y la privacidad del usuario. En lugar de que cada navegador contacte con la Autoridad de Certificación (CA) para comprobar si el certificado ha sido revocado, el propio servidor web Apache consulta periódicamente el estado de revocación a la CA y adjunta la respuesta firmada directamente durante el saludo TLS (*TLS Handshake*).

Añadimos la siguiente configuración a nivel global (en `/etc/apache2/conf-available/ssl-stapling.conf`):

```
1 <IfModule mod_ssl.c>
2     SSLUseStapling on
3     SSLStaplingCache "shmcb:${APACHE_RUN_DIR}/ssl_stapling(32768)"
4     SSLStaplingResponseTime 5
5     SSLStaplingResponderTimeout 5
6     SSLStaplingReturnUntrusted off
7 </IfModule>
```

1.9.4. Habilitación de HTTP/2

El protocolo **HTTP/2** ofrece mejoras drásticas de rendimiento (multiplexación de peticiones sobre una única conexión TCP, compresión de cabeceras HPACK) y exige de forma obligatoria el uso de TLS moderno:

```
1 # En el archivo de VirtualHost o a nivel global
2 Protocols h2 http/1.1
```

1.9.5. Redirección obligatoria de HTTP a HTTPS

Todo el tráfico que llegue al puerto TCP 80 debe ser redirigido inmediatamente al puerto seguro 443 con un código de estado 301 **Moved Permanently**:

```
1 # /etc/apache2/sites-available/000-default.conf
2 <VirtualHost *:80>
3     ServerName midominio.com
4     ServerAlias www.midominio.com
5
6     # Redirección permanente a HTTPS preservando la ruta solicitada
7     Redirect permanent / https://midominio.com/
8 </VirtualHost>
```

1.10. Protección contra denegación de servicio (DoS) y agotamiento de recursos

Un servidor web Apache mal parametrizado puede ser fácilmente inutilizado mediante ataques de Denegación de Servicio basados en el agotamiento de descriptores de sockets, procesos y memoria (*Slowloris*, *Slow HTTP POST* o inundaciones de peticiones).

1.10.1. Ajuste de tiempos de espera (*timeouts*) y conexiones persistentes

En `/etc/apache2/apache2.conf`, reducimos los tiempos de espera excesivos por defecto:

```
1 # Tiempo máximo (en segundos) que Apache esperará para recibir una petición
2 Timeout 30
3
4 # Habilitar conexiones HTTP persistentes (Keep-Alive)
5 KeepAlive On
6
7 # Número máximo de peticiones por conexión persistente
8 MaxKeepAliveRequests 100
9
10 # Tiempo de espera (en segundos) para la siguiente petición en la misma conexión
11 KeepAliveTimeout 5
```

1.10.2. Limitación del tamaño de peticiones y cabeceras

Para prevenir desbordamientos de búfer y subidas masivas de datos que consuman el ancho de banda del servidor, establecemos límites estrictos:

```
1 # Limita el tamaño del cuerpo de la petición HTTP a 10 MB (10485760 bytes)
2 # Ajustar según las necesidades reales de subida de la aplicación
3 LimitRequestBody 10485760
4
5 # Limita el tamaño de la línea de petición HTTP (URI) a 4094 bytes
6 LimitRequestLine 4094
7
8 # Limita el número de campos en las cabeceras HTTP a 50
9 LimitRequestFields 50
10
11 # Limita el tamaño máximo de cada campo de cabecera a 8190 bytes
12 LimitRequestFieldSize 8190
```

1.10.3. Mitigación de Slowloris con `mod_reqtimeout`

El ataque **Slowloris** mantiene abiertas miles de conexiones HTTP enviando cabeceras a una velocidad extremadamente lenta (1 byte cada pocos segundos), agotando todos los hilos de Apache. El módulo nativo

`mod_reqtimeout` neutraliza esta técnica exigiendo un tiempo máximo y una tasa mínima de transferencia de datos:

```
1 sudo a2enmod reqtimeout
```

Configuramos en `/etc/apache2/mods-available/reqtimeout.conf`:

```
1 <IfModule mod_reqtimeout.c>
2     # Espera un máximo de 20s a 40s para recibir cabeceras; si la tasa es menor
    de 500 bytes/s, cancela la conexión
3     RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500
4 </IfModule>
```

1.10.4. Prevención de inundaciones con `mod_evasive`

`mod_evasive` proporciona protección frente a ataques de fuerza bruta, escaneo masivo de URLs y ataques DoS HTTP en capa de aplicación.

```
1 sudo apt install libapache2-mod-evasive -y
```

Configuramos `/etc/apache2/mods-available/evasive.conf`:

```
1 <IfModule mod_evasive20.c>
2     # Tamaño de la tabla hash interna
3     DOSHashTableSize    3097
4
5     # Número máximo de peticiones a una misma página en el intervalo (por
    defecto 1s)
6     DOSPageCount        10
7     DOSPageInterval     1
8
9     # Número máximo de peticiones totales a un mismo sitio en el intervalo (por
    defecto 1s)
10    DOSSiteCount         100
11    DOSSiteInterval     1
12
13    # Tiempo de bloqueo en segundos para una IP infractora (recibirá error 403
    Forbidden)
14    DOSBlockingPeriod    60
15
16    # Directorio para registrar bloqueos temporales
17    DOSLogDir             "/var/log/mod_evasive"
18
19    # Lista blanca de IPs que no deben ser bloqueadas (por ejemplo localhost o
    monitorización)
20    DOSWhitelist          127.0.0.1
21 </IfModule>
```

[!TIP] **Ajuste realista de umbrales en `mod_evasive`:** En sitios web modernos con abundantes elementos estáticos (múltiples archivos CSS, JS, fuentes e imágenes), un único acceso de un usuario legítimo puede disparar más de 40 peticiones simultáneas. Establecer `DOSSiteCount` por debajo de 50 provocaría falsos positivos (bloqueo a usuarios legítimos). Valores de `DOSPageCount` 10 y `DOSSiteCount` 100

ofrecen un equilibrio robusto y realista.

Creamos el directorio de logs con los permisos adecuados:

```
1 sudo mkdir -p /var/log/mod_evasive
2 sudo chown -R www-data:www-data /var/log/mod_evasive
3 sudo a2enmod evasive
4 sudo systemctl restart apache2
```

1.11. Cortafuegos de aplicaciones web (WAF): ModSecurity y OWASP CRS

Un **WAF** (*Web Application Firewall*) opera en la capa 7 del modelo OSI (capa de aplicación), inspeccionando en tiempo real tanto el tráfico HTTP entrante como las respuestas salientes del servidor.

Mientras que un cortafuegos tradicional (como UFW o iptables) solo filtra puertos e IPs, **ModSecurity** analiza el contenido de las peticiones para detectar y bloquear patrones de inyección SQL (SQLi), Cross-Site Scripting (XSS), inclusión de archivos locales y remotos (LFI/RFI), y ataques contra vulnerabilidades de día cero (*0-days*).



1.11.1. Instalación de ModSecurity

En Ubuntu y Debian instalamos el módulo oficial:

```
1 sudo apt install libapache2-mod-security2 -y
```

1.11.2. Activación del motor de reglas y ciclo de vida en producción

Por defecto, ModSecurity viene configurado en modo solo detección (*DetectionOnly*). En entornos profesionales se recomienda seguir este procedimiento:

1. **Fase de detección (*staging*):** Mantener `SecRuleEngine DetectionOnly` durante unos días para registrar las peticiones legítimas y detectar posibles falsos positivos.

2. **Exclusión de falsos positivos:** Si alguna regla del CRS bloquea una funcionalidad legítima (como la edición de entradas en WordPress o subida de ficheros JSON), se desactiva puntualmente esa regla mediante `SecRuleRemoveById <ID_REGLA>`.
3. **Fase de bloqueo (*production*):** Activar el motor en modo de bloqueo `SecRuleEngine On`.

```
1 # Copiar el archivo de plantilla recomendado
2 sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/
   modsecurity.conf
3
4 # Habilitar el motor de bloqueo (SecRuleEngine On)
5 sudo sed -i 's/SecRuleEngine DetectionOnly/SecRuleEngine On/' /etc/modsecurity/
   modsecurity.conf
```

1.11.3. Integración del OWASP ModSecurity Core Rule Set (CRS)

El **OWASP CRS** es el conjunto de reglas de seguridad más reconocido y mantenido del mundo para proteger servidores web.

Instalamos el paquete del CRS en el sistema:

```
1 sudo apt install modsecurity-crs -y
```

Verificamos en `/etc/apache2/mods-available/security2.conf` que las reglas del CRS se carguen adecuadamente:

```
1 <IfModule security2_module>
2     SecDataDir /var/cache/modsecurity
3     IncludeOptional /etc/modsecurity/*.conf
4     IncludeOptional /usr/share/modsecurity-crs/*.conf
5     IncludeOptional /usr/share/modsecurity-crs/rules/*.conf
6 </IfModule>
```

Reiniciamos el servicio:

```
1 sudo apache2ctl configtest
2 sudo systemctl restart apache2
```

1.11.4. Inspección de registros forenses de ModSecurity

Cualquier ataque bloqueado quedará detalladamente registrado en el archivo de auditoría:

```
1 sudo tail -f /var/log/apache2/modsec_audit.log
```

1.12. Detección y bloqueo de intrusiones con Fail2Ban

Fail2Ban es un sistema de prevención de intrusiones (IPS) que monitoriza continuamente los archivos de registro (*logs*) de Apache y del sistema operativo. Cuando detecta múltiples intentos de acceso fallidos, escaneos de vulnerabilidades o patrones anómalos procedentes de una misma dirección IP, añade automáticamente una regla en el cortafuegos de red (*nftables* o *iptables*) para bloquear el tráfico de esa IP durante un tiempo determinado.



1.12.1. Instalación de Fail2Ban

```
1 sudo apt install fail2ban -y
```

1.12.2. Configuración de jaulas (*jails*) para Apache

En lugar de modificar el archivo original `/etc/fail2ban/jail.conf`, creamos una copia de configuración personalizada en `/etc/fail2ban/jail.local`:

```

1 # /etc/fail2ban/jail.local
2
3 [DEFAULT]
4 # Tiempo de baneo por defecto: 1 hora (3600 segundos)
5 bantime = 3600
6 # Ventana de tiempo para contar reintentos: 10 minutos (600 segundos)
7 findtime = 600
8 # Número máximo de intentos antes del baneo
9 maxretry = 5
  
```

```
10 # IPs en lista blanca que nunca serán bloqueadas
11 ignoreip = 127.0.0.1/8 ::1 192.168.1.0/24
12
13 # 1. Protección contra fuerza bruta en autenticación básica HTTP
14 [apache-auth]
15 enabled = true
16 port = http,https
17 logpath = %(apache_error_log)s
18
19 # 2. Bloqueo de escáneres maliciosos y bots dañinos
20 [apache-badbots]
21 enabled = true
22 port = http,https
23 logpath = %(apache_access_log)s
24 bantime = 86400
25 maxretry = 1
26
27 # 3. Bloqueo de escaneos a scripts inexistentes (.php, .asp, .cgi)
28 [apache-noscript]
29 enabled = true
30 port = http,https
31 logpath = %(apache_error_log)s
32 maxretry = 3
33
34 # 4. Bloqueo de intentos de desbordamiento de búfer en URIs
35 [apache-overflows]
36 enabled = true
37 port = http,https
38 logpath = %(apache_error_log)s
39 maxretry = 2
40
41 # 5. Integración con ModSecurity (baneo automático de atacantes detectados por
    el WAF)
42 [apache-modsecurity]
43 enabled = true
44 port = http,https
45 logpath = /var/log/apache2/modsec_audit.log
46 maxretry = 2
```

Iniciamos y habilitamos el servicio:

```
1 sudo systemctl enable fail2ban
2 sudo systemctl restart fail2ban
```

1.12.3. Comandos de supervisión de Fail2Ban

Para comprobar el estado de las jaulas y consultar las IPs actualmente bloqueadas:

```
1 # Ver estado general y jaulas activas
2 sudo fail2ban-client status
3
4 # Ver detalle de una jaula específica (ej. apache-auth o apache-noscript)
5 sudo fail2ban-client status apache-auth
6
```

```
7 # Desbloquear manualmente una IP baneada por error
8 sudo fail2ban-client set apache-auth unbanip 192.168.1.100
```

1.13. Gestión, monitorización y auditoría de registros (logs)

Los registros de actividad (*logs*) constituyen la principal fuente de información forense ante un incidente de seguridad y son indispensables para la monitorización continua.

1.13.1. Formato enriquecido de registros y soporte para proxies inversos

En entornos donde Apache se encuentra tras un balanceador de carga, CDN o proxy inverso (como Cloudflare, HAProxy o Nginx), la IP registrada por defecto sería la del propio proxy. Es fundamental registrar la IP real del cliente activando el módulo `mod_remoteip`:

```
1 sudo a2enmod remoteip
```

Configuramos en `/etc/apache2/conf-available/remoteip.conf`:

```
1 <IfModule mod_remoteip.c>
2     RemoteIPHeader X-Forwarded-For
3     # Declarar rangos de confianza de tus proxies/balanceadores internos
4     RemoteIPInternalProxy 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 127.0.0.1
5 </IfModule>
```

En `/etc/apache2/apache2.conf`, definimos un formato de log enriquecido que incluya el tiempo de respuesta del servidor en microsegundos (%D):

```
1 # Formato combinado avanzado incluyendo tiempo de proceso (%D) y cabeceras
  completas
2 LogFormat "%a %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" %D"
  combined_secure
3 CustomLog ${APACHE_LOG_DIR}/access.log combined_secure
4
5 # Nivel de detalle de errores: en producción se recomienda "warn" o "notice"
6 LogLevel warn
7 ErrorLog ${APACHE_LOG_DIR}/error.log
```

1.13.2. Permisos seguros en el directorio de registros

Los registros nunca deben ser legibles por usuarios no privilegiados del sistema:

```
1 sudo chown -R root:adm /var/log/apache2
2 sudo chmod 750 /var/log/apache2
3 sudo chmod 640 /var/log/apache2/*.log
```

1.13.3. Rotación segura con logrotate

Verificamos la configuración de rotación en `/etc/logrotate.d/apache2` para evitar el agotamiento del espacio en disco y garantizar la retención adecuada de evidencias para auditorías:

```
1 /var/log/apache2/*.log {
2     daily
3     missingok
4     rotate 30
5     compress
6     delaycompress
7     notifempty
8     create 640 root adm
9     sharedscripts
10    postrotate
11        /usr/sbin/apache2ctl graceful > /dev/null
12    endscript
13 }
```

1.14. Auditoría de seguridad y herramientas de verificación

Una vez aplicadas las directivas de bastionado, es imprescindible auditar y validar la eficacia de las medidas adoptadas utilizando herramientas profesionales.

Herramienta	Tipo	Objetivo principal
curl / httpie	Línea de comandos (CLI)	Inspección de cabeceras HTTP y códigos de respuesta.
testssl.sh	Script bash local (CLI)	Análisis exhaustivo de protocolos y suites TLS/SSL.
Nikto	Escáner de seguridad	Detección de ficheros sensibles y vulnerabilidades web.
Nmap NSE	Escáner de red	Auditoría de puertos abiertos y métodos HTTP admitidos.
Lynis	Auditoría de host	Evaluación integral del bastionado del SO y Apache.
SecurityHeaders.com	Servicio web online	Calificación y diagnóstico de cabeceras HTTP.
SSL Labs (Qualys)	Servicio web online	Calificación A+ de la configuración y certificados SSL.

1.14.1. Verificación con curl

```
1 # Inspeccionar cabeceras HTTP y validar ausencia de versiones
2 curl -I https://midominio.com
3
4 # Probar si el método TRACE está desactivado (debe devolver 403 o 405)
5 curl -X TRACE https://midominio.com
6
7 # Intentar acceder a un archivo sensible bloqueado (debe devolver 403 Forbidden)
8 curl -I https://midominio.com/.env
9 curl -I https://midominio.com/.git/config
10 curl -I https://midominio.com/backup.sql
```

1.14.2. Auditoría criptográfica con testssl.sh

testssl.sh es una herramienta de software libre que evalúa cualquier servicio TLS para detectar cifrados obsoletos y vulnerabilidades conocidas (Heartbleed, POODLE, ROBOT, DROWN, CRIME):

```
1 # Descarga y ejecución de testssl
2 git clone --depth 1 https://github.com/drwetter/testssl.sh.git
```

```
3 ./testssl.sh/testssl.sh https://midominio.com
```

1.14.3. Escaneo de vulnerabilidades web con Nikto

```
1 sudo apt install nikto -y
2 nikto -h https://midominio.com -ssl
```

1.14.4. Auditoría completa del sistema con Lynis

```
1 sudo apt install lynis -y
2 sudo lynis audit system
```

1.15. Lista de verificación de bastionado

A continuación se resume la lista de verificación completa que todo administrador de sistemas debe validar antes del paso a producción:

Nº	Medida de bastionado	Directiva / Comando clave	Archivo afectado
1	Actualización periódica del SO	<code>apt update && apt upgrade -y</code>	Sistema operativo
2	Proceso bajo usuario sin privilegios	<code>APACHE_RUN_USER= www-data</code>	<code>/etc/apache2/ envvars</code>
3	Permisos restrictivos en DocumentRoot	Directorios 755, Ficheros 644, Propietario <code>root:www-data</code>	<code>/var/www/html</code>
4	Bloqueo de ejecución en subidas (<i>uploads</i>)	<code><Directory "/ uploads"> ForceType application/ octet-stream ...</code>	<code>conf-available/ security.conf</code>
5	Ocultar versión del servidor	<code>ServerTokens Prod y ServerSignature Off</code>	<code>conf-available/ security.conf</code>
6	Ocultar cabeceras backend PHP	<code>expose_php = Off /Header always unset X-Powered- By</code>	<code>php.ini/security -headers.conf</code>
7	Páginas de error personalizadas	<code>ErrorDocument 404 /errores /404.html</code>	<code>conf-available/ security.conf</code>
8	Denegación global por defecto	<code><Directory /> Require all denied </ Directory></code>	<code>apache2.conf</code>
9	Desactivar listado de directorios	<code>Options -Indexes -FollowSymLinks + SymLinksIfOwnerMatch</code>	<code>apache2.conf o VirtualHost</code>

Nº	Medida de bastionado	Directiva / Comando clave	Archivo afectado
10	Desactivar <code>.htaccess</code> innecesarios	<code>AllowOverride None</code>	<code>apache2.conf</code>
11	Bloqueo de ficheros sensibles y ocultos	<code><FilesMatch "^\\. "> Require all denied </FilesMatch></code>	<code>conf-available/security.conf</code>
12	VirtualHost catch-all / por defecto	<code>000-catchall.conf</code> con <code>Require all denied</code>	<code>sites-available/000-catchall.conf</code>
13	Arquitectura moderna con <code>mpm_event</code>	<code>a2enmod mpm_event proxy_fcgi y php-fpm</code>	<code>mods-enabled/</code>
14	Módulos mínimos e indispensables	<code>a2dismod autoindex status info userdir cgi include dav</code>	<code>mods-enabled/</code>
15	Desactivar método HTTP TRACE	<code>TraceEnable Off</code>	<code>conf-available/security.conf</code>
16	Limitar métodos HTTP estándar	<code><LimitExcept GET POST HEAD OPTIONS> Require all denied</code>	<code>apache2.conf</code> o <code>VirtualHost</code>
17	Cabeceras de seguridad HTTP	HSTS, CSP, X-Frame-Options, X-Content-Type-Options, etc.	<code>conf-available/security-headers.conf</code>
18	Cookies de sesión seguras	<code>Header edit Set-Cookie ... "Secure; HttpOnly; SameSite=Lax"</code>	<code>conf-available/security-headers.conf</code>
19	Cifrado TLS 1.3 y TLS 1.2 moderno	<code>SSLProtocol -all +TLSv1.2 +TLSv1.3</code>	<code>mods-available/ssl.conf</code>

Nº	Medida de bastionado	Directiva / Comando clave	Archivo afectado
20	Grapado OCSP y HTTP/2	<code>SSLUseStapling on</code> y <code>Protocols h2 http/1.1</code>	<code>ssl-stapling.conf/VirtualHost</code>
21	Mitigación DoS y Slowloris	<code>mod_reqtimeout</code> y reducción de <code>Timeout 30</code>	<code>apache2.conf/reqtimeout.conf</code>
22	Prevención de abusos y flood	<code>mod_evasive</code> con umbrales realistas	<code>mods-available/evasive.conf</code>
23	Cortafuegos WAF en capa 7	ModSecurity con OWASP Core Rule Set (<code>SecRuleEngine On</code>)	<code>/etc/modsecurity/</code>
24	Prevención de intrusiones con IPS	Fail2Ban con jaulas para Apache y WAF	<code>/etc/fail2ban/jail.local</code>
25	Registros enriquecidos y rotación	Formato <code>combined_secure</code> , permisos 640, <code>logrotate</code>	<code>/var/log/apache2/</code>

1.16. Actividades y ejercicios prácticos de laboratorio

A continuación se plantean siete actividades prácticas guiadas para afianzar los conocimientos en una máquina virtual de prácticas con Ubuntu Server.

1.16.1. Ejercicio 1: Auditoría inicial y ocultación de información

1. Instala el servidor Apache en tu máquina virtual Ubuntu Server (`sudo apt install apache2`).
2. Realiza una petición con `curl -I http://localhost` y anota la cabecera `Server` y las versiones visibles.
3. Modifica `/etc/apache2/conf-available/security.conf` para establecer `ServerTokens ProdyServerSignature Off`.
4. Habilita la configuración si no lo estuviera (`sudo a2enconf security`), comprueba la sintaxis con `apache2ctl configtest` y recarga el servicio.
5. Vuelve a ejecutar `curl -I http://localhost` y comprueba que la versión exacta y la distribución han desaparecido.

1.16.2. Ejercicio 2: Restricción de permisos, bloqueo de ficheros y protección de subidas

1. Crea en `/var/www/html/` un archivo oculto llamado `.env` que contenga una clave simulada (`DB_PASSWORD=secreto123`) y un volcado de base de datos `backup.sql`.
2. Crea un directorio `/var/www/html/uploads/` y dentro de él crea un script de prueba `info.php` que contenga `<?php phpinfo(); ?>`.
3. Comprueba con el navegador o `curl` que inicialmente el servidor permite descargar `.env` y ejecutar `info.php`.
4. Añade a `/etc/apache2/conf-available/security.conf` las reglas `<FilesMatch>` para bloquear archivos que comiencen por punto y extensiones sensibles (`.sql`, `.bak`, `.env`).
5. Añade la directiva de protección en `<Directory "/var/www/html/uploads">` para denegar la ejecución de scripts PHP y forzar `ForceType application/octet-stream`.
6. Recarga Apache y verifica con `curl -I http://localhost/.env` y `curl http://localhost/uploads/info.php` que en ambos casos se devuelve un código de estado 403 `Forbidden`.

1.16.3. Ejercicio 3: Configuración del VirtualHost catch-all y redirección HTTPS

1. Crea el archivo `/etc/apache2/sites-available/000-catchall.conf` con una configuración que deniegue todo el tráfico a peticiones dirigidas por IP directa.
2. Habilita el sitio con `sudo a2ensite 000-catchall.conf`.
3. Configura tu VirtualHost principal (por ejemplo `misitio.conf`) con `ServerName misitio.local`.

4. Comprueba que al acceder mediante `curl -H "Host: misitio.local" http://localhost` se accede a la web, mientras que al acceder mediante `curl http://localhost` se recibe un error 403 `Forbidden`.

1.16.4. Ejercicio 4: Implementación y validación de cabeceras de seguridad y cookies

1. Habilita el módulo de cabeceras (`sudo a2enmod headers`).
2. Crea el archivo `/etc/apache2/conf-available/security-headers.conf` e incluye las cabeceras recomendadas (`X-Frame-Options`, `X-Content-Type-Options`, `Referrer-Policy`, `Permissions-Policy`, `Content-Security-Policy`, `cookie security`).
3. Habilita la configuración (`sudo a2enconf security-headers`), valida con `apache2ctl configtest` y reinicia Apache.
4. Realiza una petición con `curl -I http://localhost` y verifica que todas las cabeceras aparecen correctamente en la respuesta HTTP.

1.16.5. Ejercicio 5: Mitigación de Slowloris y protección DoS con `mod_reqtimeout` y `mod_evasive`

1. Habilita el módulo `reqtimeout` (`sudo a2enmod reqtimeout`).
2. Configura en `apache2.conf` un `Timeout` de 30 segundos y `KeepAliveTimeout` de 5 segundos.
3. Ajusta en `reqtimeout.conf` los tiempos de lectura de cabeceras y cuerpo con tasa mínima de 500 bytes/s.
4. Instala y configura `libapache2-mod-evasive` con umbrales adecuados (`DOSPageCount 10`, `DOSSiteCount 100`).
5. Valida la sintaxis y reinicia el servicio.

1.16.6. Ejercicio 6: Despliegue de ModSecurity (WAF) y bloqueo de inyecciones SQL

1. Instala los paquetes `libapache2-mod-security2` y `modsecurity-crs`.
2. Activa `SecRuleEngine On` en `/etc/modsecurity/modsecurity.conf`.
3. Verifica la inclusión de las reglas OWASP en `/etc/apache2/mods-available/security2.conf` y reinicia Apache.
4. Simula un ataque de inyección SQL enviando la siguiente petición:

```
1 curl "http://localhost/?id=1%20UNION%20SELECT%20null,username,password%20FROM%20users"
```

5. Verifica que Apache responde con un error 403 `Forbidden` y revisa la entrada generada en `/var/log/apache2/modsec_audit.log`.

1.16.7. Ejercicio 7: Detección y bloqueo activo con Fail2Ban

1. Instala el paquete `fail2ban` (`sudo apt install fail2ban -y`).
 2. Crea `/etc/fail2ban/jail.local` habilitando las jaulas `apache-auth`, `apache-badbots`, `apache-noscript` y `apache-overflows`.
 3. Inicia el servicio Fail2ban y verifica su estado con `sudo fail2ban-client status`.
 4. Simula múltiples peticiones a scripts inexistentes (`curl http://localhost/admin.php`, `curl http://localhost/setup.cgi`, etc.) desde otra máquina o usando herramientas de prueba.
 5. Comprueba con `sudo fail2ban-client status apache-noscript` que la IP atacante ha sido detectada y bloqueada en el cortafuegos.
-

2 Referencias

- [Documentación oficial de Apache HTTP Server \(Consejos de Seguridad\)](#)
- [CIS Apache HTTP Server Benchmark](#)
- [NIST Special Publication 800-44: Guidelines on Securing Public Web Servers](#)
- [OWASP Web Server Security Cheat Sheet](#)
- [OWASP ModSecurity Core Rule Set \(CRS\)](#)
- [Mozilla SSL Configuration Generator](#)
- [Guía de Cabeceras de Seguridad HTTP de MDN Web Docs](#)
- [Herramienta de Auditoría testssl.sh](#)
- [Herramienta de Análisis Security Headers de Scott Helme](#)
- [Qualys SSL Labs Server Test](#)

3 Licencia

Esta página forma parte del curso Implantación de Aplicaciones Web de José Juan Sánchez Hernández y su contenido se distribuye bajo una licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.