

Puesta en Producción Segura

Unidad .

Errores en la configuración de
seguridad y Componentes
vulnerables y desactualizados



"No hay parches para una mala configuración."

—Andrew S. Tanenbaum

Objetivos

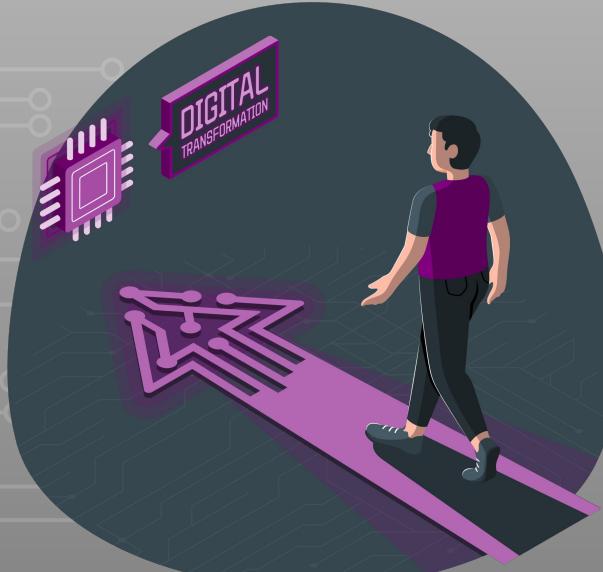
- Comprender cómo las vulnerabilidades relacionadas con errores en la configuración de seguridad, Fallos en la monitorización y logs y Componentes vulnerables y desactualizados pueden comprometer la seguridad de una aplicación web y cómo mitigarlas.
- Conocer otras vulnerabilidades relacionadas con Errores en la configuración de seguridad, Fallos en la monitorización y logs y Componentes vulnerables y desactualizados.
- Ver dónde encontrar información de ellas en las listas CWE y CAPEC.
- Conocer qué es un WAF y su funcionamiento.



Contenidos

- 01** Errores en la configuración de seguridad
- 02** Fallos en la monitorización y el log de seguridad

- 03** Componentes vulnerables y desactualizados
- 04** Firewall a nivel de aplicación (WAF)



01

Errores en la configuración de seguridad



SECURITY MISCONFIGURATION

Hablamos de **SECURITY MISCONFIGURATION** o **Errores en la configuración de seguridad** o **Configuración insegura** cuando encontramos errores en la configuración de seguridad de servidores, bases de datos y aplicaciones web que los hacen inseguros frente a ataques.

Ejemplos de configuraciones inseguras

- Funcionalidades innecesarias habilitadas (puertos, servicios, páginas, privilegios).
- Cuentas por defecto habilitadas.
- Páginas de error por defecto.
- Control de acceso que permite llamadas utilizando métodos HTTP no controlados.
- Otros: CORS, etc.

Corrección y mitigación

Principios clave

- **Seguridad por defecto:** No dejar configuraciones abiertas.
- **Principio de menor privilegio:** No otorgar más permisos de los necesarios.
- **Automatización de configuraciones seguras:** Uso de herramientas como **Ansible** o **Terraform**.

Herramientas recomendadas

- **OWASP ZAP** para pruebas de seguridad web.
- **Nmap** para identificar servicios abiertos.
- **Lynis** para auditoría de servidores Linux.



Corrección y mitigación

Dentro de la configuración de seguridad incorrecta, también se pueden incluir diferentes parámetros relacionados con la política de control de acceso:

- Limitar el número de sesiones por usuario.
- Cerrar la sesión después de un período de inactividad.
- Bloquear la sesión después de sobrepasar un tiempo de uso.
- Limitar el número de registros que puede devolver una consulta.
- Separar las aplicaciones del usuario de las aplicaciones de gestión.



OWASP Top 10

Las vulnerabilidades en la configuración de seguridad ocupan la posición 5 en el boletín OWASP Top-10 del año 2021

- Explotabilidad: los ataques que explotan funcionalidades y cuentas por defecto son muy frecuentes.
- Prevalencia y detectabilidad: algunas configuraciones incorrectas y funcionalidades no necesarias, se pueden detectar de forma automática.
- Impacto técnico: de moderado a alto, dependiendo de la configuración. Si la cuenta comprometida es de administrador, el daño puede ser elevado.



Configuración insegura en CWE y CAPEC

Algunas entradas del **CWE** relacionadas con la configuración de seguridad incorrecta son las siguientes:

- CWE CATEGORY 4: J2EE Environment Issues
- CWE CATEGORY 519: .NET Environment Issues
- CWE-732: Incorrect Permission Assignment for Critical Resource

Entradas del **CAPEC** relacionadas con la configuración de seguridad incorrecta son las siguientes:

- CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs





02

Fallos en la monitorización y el log

Importancia del Logging y Monitorización

¿Por qué es importante?

- Permite detectar ataques en tiempo real.
- Facilita auditorías y análisis forenses.

Errores comunes en Logging:

- No registrar intentos de acceso fallidos.
- No monitorear cambios en configuraciones críticas.
- Almacenar logs sin cifrado, exponiéndose a ataques.



Fallos en la monitorización y el log

- El **registro de eventos en el log** se utiliza para reproducir de la forma más fiel posible, la causa que ha producido algún tipo de error en la aplicación. Para ello, en los ficheros de log se deberían almacenar todos los pasos que han llevado hasta la situación de inconsistencia.
- La **monitorización** de una aplicación se utiliza para determinar, en todo momento, el uso de recursos, tanto físicos como lógicos:
 - Consumo de memoria, CPU, etc.
 - Número de usuarios conectados, sesiones abiertas en total, sesiones abiertas por un mismo usuario, etc.



Fuente de la imagen:
<https://libertia.es/registros-de-logs/>

Fallos en la monitorización y el log

- La **falta de registros de log** puede llevar a que, ante un problema de seguridad, no **exista ningún tipo de información sobre el ataque**, ya que esto va a dificultar la detección y corrección del problema
- Del mismo modo, **los registros de log** deben mostrar **información clara y concisa**. Si el log es excesivo o poco claro, también resulta difícil extraer de él la información útil.
- Almacenar el log sólo de forma local, podría llegar a ser peligroso, si el atacante consigue hacerse con el control de la máquina, por lo que se hace necesario enviar esa información de los logs a otras máquinas.

Buenas prácticas en Logging

¿Qué eventos registrar?

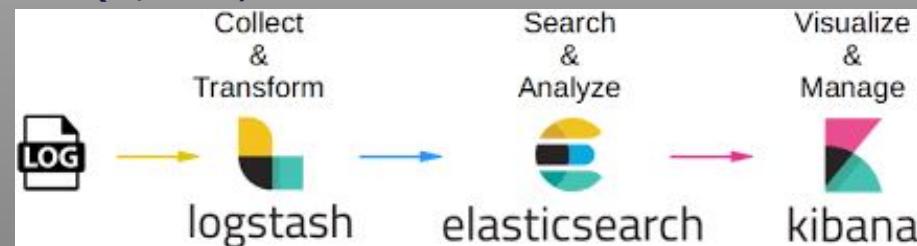
- Eventos de inicio de sesión, especialmente Intentos de acceso fallidos.
- Eventos de control acceso, especialmente sobre recursos sin permisos.
- Acciones administrativas (creación de usuarios, cambios de permisos).
- Transacciones relevantes.
- Peticiones sospechosas (inyecciones SQL, XSS).

Ejemplo de registro de eventos

```
{  
  "timestamp": "2024-03-06T12:00:00Z",  
  "event": "Failed login attempt",  
  "username": "admin",  
  "ip_address": "192.168.1.100"  
}
```

Mejorando la seguridad de los logs:

- Almacenar logs de manera centralizada (ELK, Splunk).
- Evitar incluir datos sensibles en registros.
- Monitorear en tiempo real con alertas automatizadas.



Fuente de la imagen:<https://www.geeksforgeeks.org/software-engineering/what-is-elastic-stack-and-elasticsearch/>

Herramientas de monitorización en tiempo real

Herramientas para monitorización:

- **SIEM** (Security Information and Event Management): Splunk, Graylog, ELK.
- **Monitorización de tráfico**: Zeek, Suricata.
- **Detección de intrusos**: Fail2Ban, Snort.

Ejemplo de Configuración en Fail2Ban (Bloquear intentos de acceso fallidos)

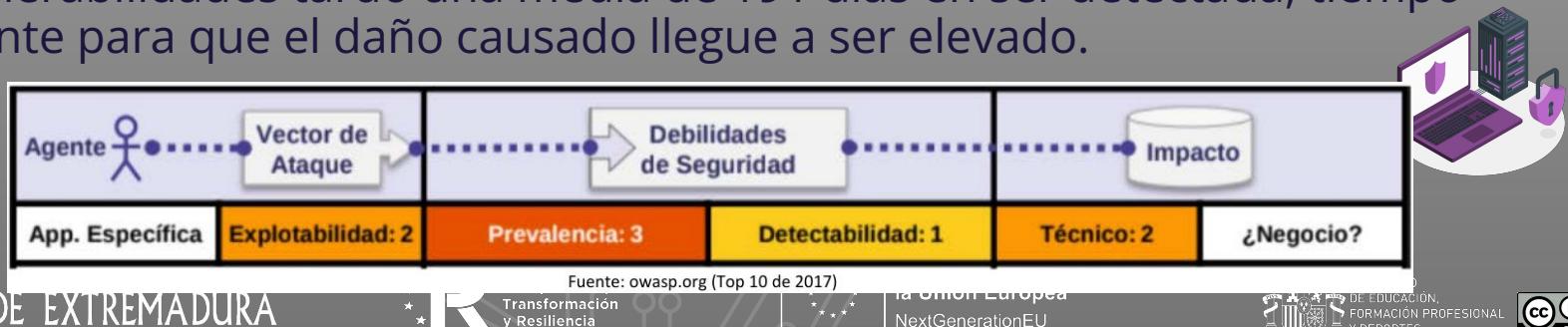
```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
bantime = 3600
maxretry = 5
```



Fallos en la monitorización y log en OWASP TOP 10

Esta vulnerabilidad ocupa la posición 9 en el boletín OWASP Top-10 del año 2021.

- **Explotabilidad:** muchos ataques están basados en la falta de respuesta una vez que se ha producido el acceso no permitido.
- **Prevalencia y detectabilidad:** la prevalencia es muy alta. La escasez de registros de log se puede detectar de forma sencilla analizando el código fuente.
- **Impacto técnico:** según diversos estudios, en el año 2016 la identificación de vulnerabilidades tardó una media de 191 días en ser detectada, tiempo suficiente para que el daño causado llegue a ser elevado.



Fallos en la monitorización y log en CWE

Una entrada del CWE relacionada con el log insuficiente es la siguiente:

- CWE-778: Insufficient Logging



03

Componentes vulnerables y desactualizados



Componentes vulnerables y desactualizados

Los componentes vulnerables y desactualizados son bibliotecas, dependencias, módulos o librerías que un proyecto utiliza y que contienen fallos de seguridad conocidos o versiones antiguas sin parches.

Estos componentes pueden ser aplicaciones o extensiones que utilizamos (IDEs, Plugins de IDE, etc.) o bien librerías de terceros que utilizamos.

No hace falta que el código sea tuyo para que sea un riesgo: una vulnerabilidad en una librería de tercero puede comprometer toda la aplicación que la incluye.



Riesgos de componentes vulnerables

- **Entrada indirecta para atacantes:** Un fallo en una dependencia permite ejecución remota, escalado de privilegios o filtrado de datos.
- **Cadena de suministro (supply-chain):** Un componente comprometido puede propagar el ataque a todos los proyectos que lo usan.
- **Cumplimiento y reputación:** Uso de versiones no parcheadas puede incumplir requisitos legales/contractuales y dañar la confianza.
- **Impacto amplio y silencioso:** Muchas dependencias son transitorias (sub-dependencias) y pasan desapercibidas en auditorías superficiales.



Corrección y mitigación

- Hoy en día, es muy frecuente el uso de librerías de terceros para implementar funcionalidades que son comunes y repetibles en prácticamente todas las aplicaciones.
- Siempre es preferible **utilizar librerías confiables ya creadas**, en vez de utilizar nuestros propios métodos, pero siempre es preferible confiar en proyectos "maduros".
- Utilizar **herramientas de comprobación de dependencias** de las librerías utilizadas: plugins de IDEs o herramientas como **OWASP Dependency Check**.

Librerías de terceros confiables

- Dentro del entorno **Java**, un referente de este tipo de librerías es el **proyecto Apache Commons**, que proporciona un gran número de librerías de código abierto bajo Licencia Apache para funcionalidades específicas.
- En **Python** nos podemos encontrar con el ecosistema **PyPI**.
- En **PHP** tenemos **Symfony Componentes**, de **HttpFoundation**, que es similar a Apache Commons y también disponemos de repositorios de **Composer**, al igual que frameworks de los que hemos hablado ya: **Django, Flask, Symfony, Laravel**)

Librerías Apache Commons para java

Ejemplos de librerías Apache Commons:

- **commons-lang**: funcionalidades extendidas del paquete java.lang.
- **commons-collections**: funcionalidades relacionadas con listas, conjuntos, etc.
- **commons-logging**: gestión de registros de log.
- **commons-email**: envío de correos electrónicos.
- **commons-pool**: pool de conexiones.
- **commons-dbcp**: pool de conexiones a base de datos.
- **commons-net**: funciones y utilidades de red.



Ecosistema PyPI de Python

El ecosistema PyPI, contiene paquetes muy usados que cubren funciones para:

requests : HTTP cliente sencillo.

validators : Validación de correos, URLs, etc.

python-dateutil / arrow : Manejo avanzado de fechas.

cryptography : Criptografía moderna y segura.

pathlib (en la stdlib moderna) : Manejo de rutas de archivos.

pydantic : Validación y serialización de datos.

Python también trae una stdlib muy extensa (json, csv, sqlite3, hashlib, logging...), por lo que muchas veces no hace falta instalar librerías externas.

Packagist del repositorio de Composer en PHP

Ejemplos de librerías comunes de Packagist para PHP:

guzzlehttp/guzzle : Cliente HTTP (equivalente a requests en Python).

ramsey/uuid : Generación de UUIDs.

respect/validation : Validación de datos.

monolog/monolog : Logging avanzado.

phpseclib/phpseclib : Criptografía pura en PHP.

symfony/* : **Symfony Components** (colección modular: HttpFoundation, Console, Validator, etc.), muy similar en espíritu a Apache Commons.



Funcionalidad	Java (Apache Commons)	Python (PyPI / stdlib)	PHP (Packagist / Composer)
HTTP cliente	HttpClient (Apache HttpComponents)	requests	guzzlehttp/guzzle
Validación	commons-validator	validators, pydantic	respect/validation
Colecciones / Utils	commons-collections, commons-lang3	itertools, functools (stdlib)	Symfony Components (symfony/var-dumper, symfony/string)
Fechas y tiempo	commons-lang3 (DateUtils)	datetime, dateutil, arrow	nesbot/carbon
Criptografía	commons-crypto	cryptography, hashlib	phpseclib/phpseclib, openssl
Logging	commons-logging	logging (stdlib)	monolog/monolog
Ficheros / IO	commons-io	pathlib, shutil, os	symfony/filesystem
UUID / Identificadores	commons-id	uuid (stdlib)	ramsey/uuid

OWASP Dependency-Check

OWASP Dependency-Check es una herramienta de OWASP que **analiza dependencias** de proyectos en busca de vulnerabilidades conocidas.

Funcionamiento:

- Escanea paquetes en Java, Python, JavaScript, .NET, PHP.
- Compara versiones con la base de datos de vulnerabilidades CVE (Common Vulnerabilities and Exposures).

Ejemplo de uso en un proyecto Java (Maven):

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.14.1</version>
</dependency>
```

- Dependency-Check alerta si la versión es vulnerable (Ejemplo: Log4Shell en Log4j).

Componentes vulnerables en OWASP TOP 10

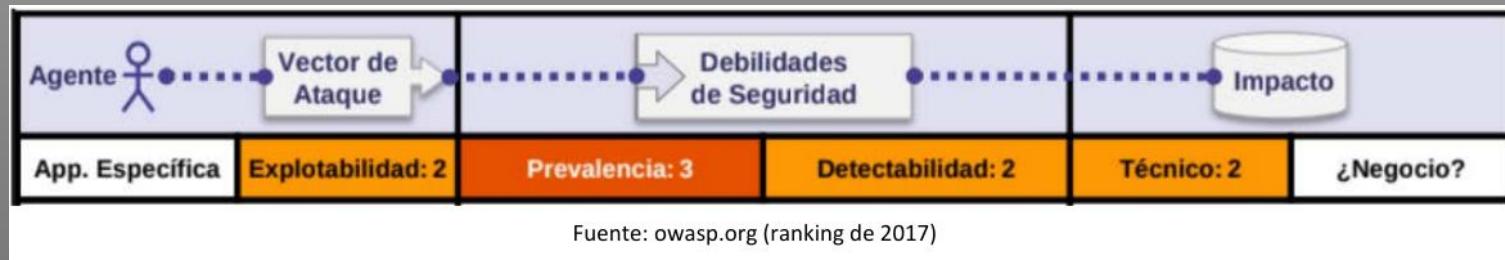
Esta vulnerabilidad, tiene categoría propia dentro de OWASP TOP 10, ocupando el lugar 6º en 2021: **Componentes Vulnerables y Desactualizados**, eso nos muestra su importancia.

- **Explotabilidad**: si las vulnerabilidades son conocidas, es muy posible que ya existan también ataques publicados
- **Prevalencia** y detectabilidad: en muchas ocasiones, los desarrolladores no son conscientes de todos los componentes que están utilizando. La detección de librerías vulnerables se puede hacer de forma automática analizando la estructura del proyecto, por ejemplo, con el plugin de maven **dependency-checker-maven**.



Componentes vulnerables en OWASP TOP 10

- **Impacto técnico:** si no se mantienen las librerías actualizadas, el impacto puede llegar a ser elevado. En el caso de las librerías de terceros, es probable que existan ataques públicos y conocidos lo que puede incrementar el número potencial de atacantes.



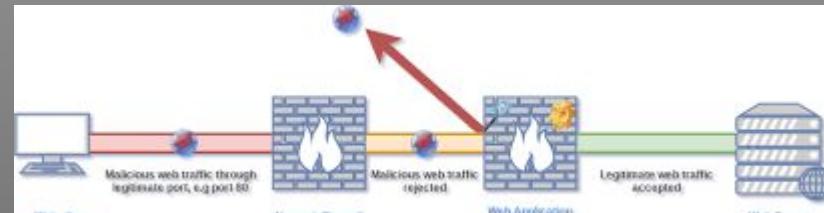


04

Firewall a nivel de aplicación (WAF)

Web Application Firewall (WAF)

- **Web Application Firewall (WAF):** Filtro de seguridad que protege aplicaciones web frente a ataques comunes (ej. SQL Injection, XSS, LFI, RFI...).
- Se sitúa entre el cliente y el servidor web, inspeccionando el tráfico HTTP/HTTPS.
- Como las aplicaciones se encuentran ubicadas en el servidor web, se suele instalar en él (Apache, NGINX).
- Puede trabajar en modo detección (monitoring) o modo bloqueo (prevention).



Fuente de la imagen:
https://commons.wikimedia.org/wiki/File:WAF_Archi.png

Reglas en un WAF

Los WAF funcionan mediante reglas de detección y bloqueo que analizan las peticiones HTTP/HTTPS.

Tipos de reglas

Reglas básicas:

- Detectan patrones simples en parámetros, cabeceras o URLs.
- Ejemplo: bloquear <script> para evitar XSS.

Reglas avanzadas (contextuales)

- Analizan contexto y comportamiento (frecuencia de peticiones, combinación de parámetros, etc.).
- Ejemplo: detectar SQLi aunque el atacante use técnicas de ofuscación.

Reglas personalizadas:

- Adaptadas a la aplicación: rutas críticas, formatos esperados, patrones de negocio.
- Ejemplo: solo aceptar números en el parámetro id.



Ventajas e inconvenientes de los WAF

Ventajas

- Bloquea ataques comunes sin necesidad de modificar la aplicación.
- Protege contra vulnerabilidades conocidas y 0-day con reglas genéricas.
- Ayuda a cumplir normativas (ej. PCI DSS exige WAF o pruebas de seguridad).
- Flexibilidad se puede usar en servidores locales, en cloud o como servicio gestionado.

Inconvenientes:

- Puede generar falsos positivos si las reglas no están bien afinadas.
- No sustituye al desarrollo seguro: solo mitiga, no soluciona el problema en el código.
- Puede introducir latencia en el tráfico web.
- Requiere mantenimiento continuo (actualización de reglas, monitorización).

Tipos de WAF

Según dónde esté instalado podemos encontrar:

- En el **servidor local**: ModSecurity (para Apache, Nginx, IIS).
- **Reverse Proxy dedicado**: NAXSI (para Nginx).
- **Soluciones comerciales**: F5, Barracuda, Fortinet.
- **En la nube (WAF-as-a-Service)**: Cloudflare WAF, AWS WAF, Azure Front Door, Imperva.
- **Integrado en appliances de seguridad**: Firewalls de nueva generación (NGFW) incluyen módulos WAF.

Instalación ModSecurity en Apache/Nginx

1. Instalar el módulo:

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

```
sudo a2enmod security2
```

En Nginx se compila o se usa como módulo dinámico.

2. Activar las reglas OWASP CRS (Core Rule Set):

- Descargar desde: <https://coreruleset.org>
- Configurar en /etc/modsecurity/modsecurity.conf.

3. Modo detección primero:

- Revisar logs en /var/log/apache2/modsec_audit.log.

4. Pasar a modo bloqueo:

- Activar SecRuleEngine On.



Fuente de la imagen:
<https://owasp.org/www-project-modsecurity>

OWASP Core Rule Set (CRS)

- Proyecto abierto de OWASP con conjunto de reglas estándar para ModSecurity y otros WAF.
- **Protege contra:**
 - Inyecciones (SQLi, XSS, LDAPi, RCE, LFI, RFI).
 - Exfiltración de datos sensibles.
 - Anomalías en protocolo HTTP.
 - Bots, crawlers y scanners.
- Se actualiza constantemente con nuevas amenazas.
- Muy usado como punto de partida para desplegar un WAF.
- Ejemplo de regla: Bloquear intentos de inyección SQL en un parámetro *id*:

```
SecRule ARGS:id "@rx select |union |insert |delete |drop" \  
"id:1001,phase:2,deny,status:403,msg:'Possible SQL Injection detectada'"
```

Otros conjuntos de reglas / vendors

- **Trustwave SpiderLabs Rules:** (para ModSecurity, más reglas comerciales).
- **Atomicorp Rules:** (comerciales, más ajustadas y con soporte).
- **Vendors en la nube:** (Cloudflare, AWS WAF, Azure) tienen propias reglas administradas que el usuario puede activar o complementar con reglas personalizadas.



Bibliografía y Webgrafía

- **Presentaciones de Puesta en Producción Segura.** *Rafael López García.*
- **Seguridad de aplicaciones.** José Losada Pérez.
- **Presentaciones de Puesta en Producción Segura.** *Rafael Fuentes Ferrer.*
- <https://owasp.org/Top10/es/>
- <https://nvd.nist.gov/vuln>



Gracias!

¿Alguna pregunta?



informatica.iesvalledeljerteplasencia.es



coordinacion.cenfp@iesvp.es



C/ Pedro y Francisco González, s/n
10600, Plasencia (Cáceres)



927 01 77 74

