15-1-2026

# Cyberseguridad

## Ejercicio 1.2

## Las 10 vulnerabilidades OWASP

VERONIQUE GRUÉ

# LAS 10 VULNERABILIDADES PRINCIPALES SEGÚN OWASP

OWASP (Open Worldwide Aplication Security Project, *Proyecto Mundial Abierto de Seguridad de Aplicaciones*) es una comunidad abierta dedicada a permitir a las organizaciones diseñar, desarrollar, adquirir, operar y mantener software para aplicaciones seguras en las que se pueda confiar. Sus programas incluyen proyectos de software de código abierto liderados por la comunidad y conferencias locales y globales, en las que participan cientos de delegaciones en todo el mundo con decenas de miles de miembros.

El Top 10 de OWASP representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web, actualizándose periódicamente para reflejar las amenazas actuales del panorama de ciberseguridad.

## 1. A01:2025 - Control de acceso roto

**Descripción**
El control de acceso aplica políticas que impiden a los usuarios actuar fuera de sus permisos previstos. Las fallas suelen provocar la divulgación no autorizada de información, la modificación o destrucción de todos los datos, o la realización de una función empresarial fuera de los límites del usuario.

**Ejemplo Práctico**
Un usuario registrado sin querer podría modificar la URL de su navegador de www.ejemplo.com/usuario/perfil?id=123 a www.ejemplo.com/usuario/perfil?id=124 y podría acceder a los datos personales de otro usuario, incluyendo dirección, teléfono y otros datos personales.

**Impacto Potencial**

- Acceso no autorizado a datos sensibles de otros usuarios.
- Escalada de privilegios (un usuario con rol básico accede a funciones de administrador).
- Modificación o eliminación de datos críticos.

**Medidas Preventivas**

- Implementar verificación de permisos en el lado del servidor para cada solicitud.
- Denegar acceso por defecto y otorgar permisos específicos según sea necesario.
- Utilizar controles de acceso basados en roles.
- Registrar y loguear todos los intentos de acceso fallidos.
- Implementar verificaciones de propiedad de recursos antes de permitir operaciones.
- Comprobar que el usuario está en la sesión en la parte privada.

Fuente Oficial

# 2. A02:2025 - Configuración incorrecta de seguridad

**Descripción**
Una mala configuración de seguridad ocurre cuando un sistema, una aplicación o un servicio en la nube se configura incorrectamente desde una perspectiva de seguridad, lo que crea vulnerabilidades.

**Ejemplo Práctico**
Un servidor web en producción tiene activado el modo de depuración, lo que muestra mensajes de error detallados con información sobre la estructura interna del sistema, rutas de archivos y versiones de software. Un atacante puede usar esta información para identificar vulnerabilidades conocidas.

**Impacto Potencial**
- Exposición de información sensible del sistema.
- Acceso a paneles de administración con credenciales predeterminadas.
- Servicios innecesarios ejecutándose, que amplían la superficie de ataque.
- Divulgación de la arquitectura interna de la aplicación.

**Medidas Preventivas**
- Desactivar funciones de depuración y mensajes de error detallados en producción.
- Cambiar todas las credenciales predeterminadas.
- Deshabilitar servicios, características y puertos innecesarios.
- Implementar procesos de hardening para todos los componentes
- Mantener actualizados todos los sistemas y aplicar parches de seguridad
- Realizar auditorías de configuración periódicas

Hardening: **fortalecer un sistema informático, red o aplicación** reduciendo su superficie de ataque para hacerlo más resistente a ciberataques, eliminando configuraciones inseguras, servicios innecesarios y vulnerabilidades, mediante la aplicación de parches, restricciones de acceso y configuraciones de seguridad estrictas.

Fuente Oficial

# 3. A03:2025 - Fallos en la cadena de suministro de software

**Descripción**
Los fallos en la cadena de suministro de software son averías u otros problemas durante el proceso de desarrollo, distribución o actualización de software. Suelen deberse a vulnerabilidades o cambios maliciosos en código, herramientas u otras dependencias de terceros de las que depende el sistema.

**Ejemplo Práctico**
Una aplicación web utiliza una biblioteca JavaScript popular descargada desde un CDN público. Un atacante compromete el CDN e inyecta código malicioso en la biblioteca. Todos los sitios web que la utilizan ahora ejecutan código malicioso que roba credenciales de usuarios.

Otro ejemplo real fue el ataque a SolarWinds en 2020, donde código malicioso se insertó en actualizaciones legítimas del software, afectando a miles de organizaciones incluyendo agencias gubernamentales.

**Impacto Potencial**

- Compromiso masivo de múltiples organizaciones a través de una única dependencia.
- Robo de datos a gran escala.
- Instalación de backdoors en sistemas críticos.
- Pérdida de confianza en la cadena de suministro completa.

**Medidas Preventivas**

- Verificar la integridad de componentes mediante hashes y firmas digitales.
- Utilizar solo paquetes de repositorios confiables y oficiales.
- Mantener un inventario de todas las dependencias (SBOM - Software Bill of Materials).
- Monitorizar vulnerabilidades conocidas en dependencias usando herramientas como Dependabot.
- Implementar análisis de seguridad en el pipeline CI/CD.
- Mantener actualizadas las aplicaciones que se necesitas para la aplicación en desarrollo (Ubuntu server, ….), y en explotación.
- Considerar el uso de repositorios privados para dependencias críticas.

Fuente Oficial

# 4. A04:2025 – Fallos criptográficos

**Descripción**

Las prácticas de cifrado inadecuadas exponen datos confidenciales a los ciberdelincuentes. Esto incluye el uso de algoritmos débiles, la falta de cifrado o la implementación incorrecta de mecanismos criptográficos.

**Tipo de cifrado Necesarios:**

**Cifrado en tránsito (Capa 4):** Es fundamental cifrar los datos mientras viajan por la red. Antiguamente era costoso, pero hoy las CPU modernas (con soporte AES) y servicios gratuitos como Let’s Encrypt hacen que gestionar certificados sea rápido y automático.

**Cifrado en reposo y aplicación (Capa 7):** No basta con proteger el "viaje" de los datos; también hay que cifrarlos cuando están almacenados y, en casos muy sensibles, dentro de la propia aplicación.

**Datos críticos y normativa:** Información como contraseñas, tarjetas de crédito o datos médicos deben tener protección extra para cumplir con leyes como el RGPD (privacidad en Europa) o el PCI DSS (pagos con tarjeta).

**Ejemplo Práctico**

Una tienda online almacena contraseñas de usuarios en texto plano en su base de datos. Un atacante obtiene acceso a la base de datos y puede ver todas las contraseñas directamente. Como muchos usuarios reutilizan contraseñas, el atacante puede ahora acceder a sus cuentas en otros servicios (email, banca online, etc.).

**Impacto Potencial**

- Robo de credenciales y contraseñas.
- Exposición de datos financieros y personales.
- Incumplimiento de reglamentos como RGPD, resultando en multas millonarias.
- Pérdida de confianza de clientes y daño reputacional.
- Robo de identidad de usuarios.

El RGPD regula el tratamiento de los datos personales que realizan personas, empresas, organizaciones, Administraciones públicas de las personas que residan en la Unión Europea.

**Medidas Preventivas**

- Utilizar **algoritmos criptográficos fuertes y actualizados.**
- Implementar HTTPS/TLS en todas las comunicaciones.
- Almacenar contraseñas con algoritmos de hash seguros como bcrypt, scrypt o Argon2.
- Cifrar datos sensibles en reposo en bases de datos.
- No inventar criptografía propia; usar bibliotecas probadas y actualizadas.
- Implementar gestión segura de claves criptográficas.
- Usar certificados SSL/TLS válidos y mantenerlos actualizados.
- Ejemplo de medida en la aplicación LoginLogoff hash de usuario+contraseña.

Los algoritmos criptográficos clave en informática incluyen **AES** para cifrado simétrico (confidencialidad de datos), **RSA** y **ECC** para cifrado asimétrico (intercambio seguro de claves y firmas digitales, como en TLS/SSL), y funciones de hash como **SHA-256** para verificar la integridad de los datos.

Fuente Oficial

# 5. A05:2025 – Ataques de Inyección

**Descripción**

Una vulnerabilidad de inyección es un fallo de la aplicación que permite que una entrada de usuario no confiable se envíe a un intérprete (por ejemplo, un navegador, una base de datos, la línea de comandos) y hace que el intérprete ejecute partes de esa entrada como comandos.

**Ejemplo Práctico**

**Inyección SQL:** Un formulario de login tiene un campo de usuario. En lugar de escribir un nombre válido, el atacante escribe: admin' OR '1'='1' --
La consulta SQL resultante sería:
SELECT * FROM usuarios WHERE nombre='admin' OR '1'='1' --' AND password='...'
Como '1'='1' siempre es verdadero, el atacante accede sin conocer la contraseña real.
**Inyección de Comandos:** Un campo que permite hacer ping a una IP recibe: 8.8.8.8; rm -rf /, ejecutando no solo el ping sino también un comando destructivo del sistema.

**Impacto Potencial**

- Extracción completa de bases de datos.
- Modificación o eliminación de datos.
- Ejecución de comandos en el servidor.
- Escalada de privilegios en el sistema.
- Toma de control total del servidor.

**Medidas Preventivas**

- Utilizar consultas parametrizadas o prepared statements.(utilizar ? o :parámetro)
- Validar y sanitizar todas las entradas de usuario.
- Implementar listas blancas de caracteres permitidos.
- Usar ORMs (Object-Relational Mapping) que manejen la sanitización.(Eloquent(PHP), Hibernate(Java)…)
- Aplicar el principio de mínimo privilegio en cuentas de base de datos.
- Implementar Web Application Firewalls (WAF.Instalar o configurar una capa de seguridad que analiza todo el tráfico (las peticiones HTTP/HTTPS)).
- Neutralizar caracteres especiales según el contexto de uso, para que no se confunda con el código.
  (ejemplo: **Sin neutralizar:** INSERT INTO usuarios VALUES ('O'Connor'); → **¡ERROR!** (Esa comilla extra rompe la sintaxis).
  **Neutralizado:** INSERT INTO usuarios VALUES ('O\'Connor'); → El \ le dice a MySQL: "la comilla que sigue es solo texto, no cierres la instrucción todavía".

Fuente Oficial

# 6. A06:2025 - Diseño inseguro

El diseño inseguro es una categoría amplia que representa diferentes debilidades, expresadas como un diseño de control ineficaz o ausente. **Existe una diferencia fundamental entre diseño inseguro e implementación insegura**: un diseño inseguro no se puede solucionar con una implementación perfecta, ya que nunca se crearon los controles de seguridad necesarios para defenderse de ataques específicos.

**Ejemplo Práctico**

Una aplicación de banca online permite realizar transferencias ilimitadas sin implementar ningún mecanismo de verificación adicional como OTP (contraseña de un solo uso) o autenticación de dos factores. El sistema fue diseñado sin considerar el riesgo de que una cuenta comprometida pudiera vaciar fondos sin controles adicionales.

Otro ejemplo: un sistema de reservas de cine permite a un usuario reservar infinitos asientos sin validación, sin límite de tiempo para completar la compra, causando bloqueo de inventario para otros usuarios legítimos.

**Impacto Potencial**

- Abuso de funcionalidades del negocio.
- Fraude financiero sin posibilidad de prevención técnica.
- Ataques de denegación de servicio por diseño.
- Imposibilidad de implementar seguridad sin rediseño completo.
- Pérdidas económicas significativas.

**Medidas Preventivas**

- Realizar modelado de amenazas durante la fase de diseño.
- Implementar historias de usuario de abuso y casos límite
  **(Ejemplo:**
  Una empresa diseña un sistema de Cupones de Descuento.
  *Historia de Usuario*: "El cliente: quiero aplicar un cupón del 10% para pagar menos".
  *Historia de Abuso*: "El atacante: quiero aplicar el mismo cupón 50 veces en la misma compra para que el producto me salga gratis".
  *Caso Límite:* "¿Qué pasa si el cupón hace que el total de la cuenta sea 0? ¿El banco permite transacciones de 0 euros?".)
- Establecer controles de negocio apropiados (límites, verificaciones, timeouts)
- Diseñar con el principio de "seguridad por defecto".
- Involucrar expertos en seguridad desde las fases iniciales del proyecto.
- Documentar requisitos de seguridad junto con requisitos funcionales.
- Realizar revisiones de arquitectura de seguridad.
- No enseñar mensaje de “el usuario no existe” para no dar pistas

Fuente Oficial

# 7. A07:2025 - Errores de autenticación

**Descripción**
Cuando un atacante logra engañar a un sistema para que reconozca a un usuario inválido o incorrecto como legítimo, se presenta esta vulnerabilidad. Incluye fallos en los mecanismos de verificación de identidad, gestión de sesiones y recuperación de credenciales

**Ejemplo Práctico**
Una aplicación permite intentos ilimitados de inicio de sesión sin bloqueos ni CAPTCHAs. Un atacante puede realizar un ataque de fuerza bruta probando miles de contraseñas comunes hasta encontrar la correcta.

Otro ejemplo: una aplicación genera tokens de sesión predecibles o secuenciales (session1, session2, session3...). Un atacante puede adivinar tokens válidos de otros usuarios y secuestrar sus sesiones.

**Impacto Potencial**

- Suplantación de identidad de usuarios legítimos.
- Acceso no autorizado a cuentas.
- Secuestro de sesiones activas.
- Compromiso masivo mediante ataques de fuerza bruta o credential stuffing.(Ataque con credenciales de Dark Web)
- Bypass completo de controles de acceso.(Entrar por una puerta trasera desprotegida)

**Medidas Preventivas**

- Implementar autenticación multifactor (MFA/2FA).
- Establecer límites de intentos de login fallidos con bloqueos temporales.
- Generar tokens de sesión criptográficamente seguros y aleatorios.
- Implementar timeouts de sesión apropiados.

- No exponer identificadores de sesión en URLs.
- Invalidar sesiones en el servidor tras logout.
- Implementar políticas de contraseñas fuertes.
- Proteger contra ataques de fuerza bruta con rate limiting y CAPTCHAs.
- Notificar a usuarios sobre inicios de sesión inusuales.
- En loginLogoff *: empty($_SESSION['usuarioVGDAWAppLoginLogoff']),* si no se ha iniciado sesión no se puede acceder a esta página.

Fuente Oficial

# 8. A08:2025 - Fallos de integridad de software o datos

**Descripción**

Los fallos de integridad de software y datos se relacionan con código e infraestructura que no protege contra código o datos no válidos o no confiables que se consideran confiables y válidos. Un ejemplo de esto es cuando una aplicación depende de complementos, bibliotecas o módulos de fuentes, repositorios y redes de entrega de contenido (CDN) no confiables.

**Ejemplo Práctico**

Una aplicación realiza actualizaciones automáticas descargando código desde un servidor sin verificar su firma digital. Un atacante compromete el servidor de actualizaciones e inyecta malware. Todos los clientes descargan e instalan automáticamente el código malicioso creyendo que es una actualización legítima.
Caso real: el ataque a CCleaner en 2017, donde una versión legítima del software fue modificada con malware y distribuida a millones de usuarios a través de canales oficiales. (articulo)

**Impacto Potencial**

- Instalación de malware en sistemas legítimos.
- Compromiso de integridad de datos críticos.
- Modificación no detectada de código en producción.
- Distribución masiva de software comprometido.
- Pérdida de confianza en procesos de actualización.

**Medidas Preventivas**

- Implementar firmas digitales para todo código y actualizaciones.
- Verificar checksums e integridad de archivos descargados.
- Utilizar pipelines CI/CD seguros con controles de integridad.(Sello y firma Digitales)
- Implementar Subresource Integrity (SRI) para recursos externos.(Añadir etiqueta de integridad si necesitas llamar un archivo externo en la web)
- Segregar y proteger entornos de desarrollo, staging y producción.
- Realizar revisiones de código y análisis de seguridad automáticos.
- Mantener registros de auditoría de cambios en producción.
- Utilizar herramientas de detección de cambios no autorizados.

Fuente Oficial

# 9. A09:2025 - Errores de registro y alertas de seguridad

**Descripción**
Sin registro ni monitorización, no se pueden detectar ataques ni infracciones, y sin alertas es muy difícil responder con rapidez y eficacia ante un incidente de seguridad. La insuficiencia de registro, monitorización continua, detección y alertas para iniciar respuestas activas puede ocurrir en cualquier momento.

**Ejemplo Práctico**
Un atacante realiza 10,000 intentos de login fallidos contra diferentes cuentas durante 3 días. Sin registros adecuados ni alertas configuradas, nadie detecta el ataque de fuerza bruta en curso. Eventualmente el atacante compromete varias cuentas y extrae datos sensibles. Cuando finalmente se descubre la brecha, ha pasado un mes y no hay registros que permitan determinar qué datos fueron comprometidos.

**Impacto Potencial**
- Ataques no detectados que permanecen activos durante meses.
- Imposibilidad de realizar análisis forense tras incidentes.
- Incapacidad para cumplir requisitos regulatorios de auditoría.
- Tiempo de respuesta extremadamente largo ante incidentes.
- Desconocimiento del alcance real de brechas de seguridad.
- Repetición de ataques por falta de aprendizaje.

**Medidas Preventivas**
- Registrar todos los eventos de seguridad relevantes (logins, fallos de autenticación, cambios de privilegios).
- Implementar monitorización continua y análisis de logs.
- Configurar alertas automáticas para eventos sospechosos.
- Proteger logs contra alteración y garantizar su integridad.
- Centralizar logs en sistemas SIEM (Security Information and Event Management).
- Establecer tiempos de retención adecuados para logs.
- Realizar revisiones periódicas de logs y patrones anómalos.
- Implementar correlación de eventos para detectar patrones de ataque.
- Crear procedimientos de respuesta ante alertas de seguridad.

Fuente Oficial

# 10. A10:2025 - Mal manejo de condiciones excepcionales

**Descripción**
La gestión inadecuada de condiciones excepcionales en el software ocurre cuando los programas no logran prevenir, detectar ni responder a situaciones inusuales e impredecibles, lo que provoca fallos, comportamientos inesperados y, en ocasiones, vulnerabilidades. Esto puede implicar una o más de las siguientes tres fallas: la aplicación no previene la ocurrencia de una situación inusual, no la identifica en el momento en que ocurre o responde de forma deficiente o nula a la situación posteriormente.

**Ejemplo Práctico**

Una aplicación de procesamiento de pagos no valida correctamente los valores de entrada. Un atacante envía un monto negativo en una transacción (-100€). El sistema, sin validación adecuada, procesa la transacción "al revés", acreditando dinero en lugar de debitarlo, permitiendo al atacante generar fondos de la nada.

Otro ejemplo: una API no maneja correctamente errores cuando recibe datos malformados. En lugar de rechazar la petición, el error causa que se exponga información sensible del stack trace, revelando rutas del sistema y versiones de software vulnerable.

**Impacto Potencial**

- Caídas de aplicación (Denial of Service).
- Exposición de información sensible a través de mensajes de error.
- Comportamiento inesperado que puede ser explotado.
- Corrupción de datos.
- Escalada de privilegios aprovechando estados inconsistentes.

**Medidas Preventivas**

- Implementar validación exhaustiva de entradas (tipos, rangos, formatos).
- Utilizar try-catch o manejo de excepciones apropiado en todo el código.
- Definir comportamientos por defecto seguros ante errores.
- No exponer detalles técnicos en mensajes de error al usuario.
- Registrar errores detalladamente para diagnóstico interno.
- Implementar límites y validaciones en valores numéricos (overflow/underflow).
- Realizar testing exhaustivo incluyendo casos límite y entradas maliciosas.
- Implementar timeouts para evitar operaciones indefinidas.

Fuente Oficial

# 11. Conclusión

**Aspectos Clave a Recordar:**

1. **La seguridad debe ser integral:** No basta con protegerse contra una vulnerabilidad; todas deben ser consideradas en conjunto.
2. **Seguridad por diseño:** Muchas vulnerabilidades (especialmente el diseño inseguro) no pueden corregirse después; deben considerarse desde el inicio del proyecto.
3. **Mantenerse informados:** El panorama de amenazas evoluciona constantemente. OWASP actualiza su lista periódicamente para reflejar nuevas amenazas.
4. **Defensa en profundidad:** Implementar múltiples capas de seguridad asegura que si una falla, otras protejan el sistema.
5. **Responsabilidad compartida:** La seguridad no es solo responsabilidad del equipo de seguridad; desarrolladores, administradores y usuarios juegan un papel crucial.

**Referencias**

- OWASP Top 10 - 2025: https://owasp.org/Top10/
- OWASP Foundation: https://owasp.org/
- OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org/