OwaspTop10-2025-A00-Introducción-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

Introducción

¡Bienvenido a la última entrega del Top Ten de OWASP!

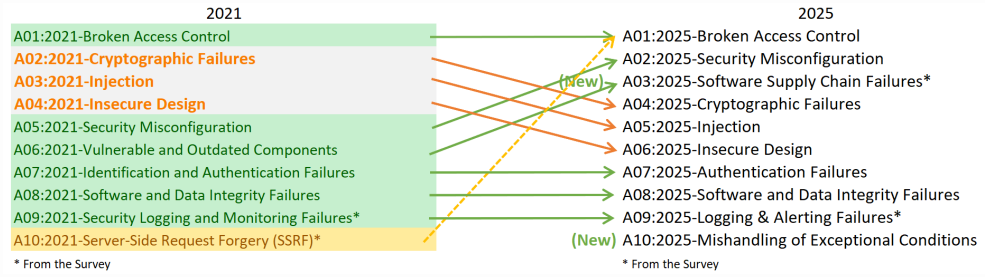
> [!info] Nota del traductor Carlos Allendes. Esta versión Release Candidate 8 (RC8) aún contiene algunos elementos marcados como FIXME, pero ya fue presentada como versión definitiva para efectos de publicación y difusión. Se espera que los FIXMEs pendientes queden resueltos hacia fines de noviembre.

Presentamos el Top 10 de OWASP: 2025

- A01:2025 - Control de acceso roto
- A02:2025 - Mala configuración de seguridad
- A03:2025 - Fallas en la cadena de suministro de software
- A04:2025 - Fallos criptográficos
- A05:2025 - Inyección
- A06:2025 - Diseño inseguro
- A07:2025 - Fallos de autenticación
- A08:2025 - Fallas de integridad de software o datos
- A09:2025 - Fallas de registro y alerta
- A10:2025 - Mal manejo de condiciones excepcionales

Qué ha cambiado en el Top 10 para 2025

Hay dos nuevas categorías y una consolidación en el Top Ten para 2025. Hemos trabajado para mantener nuestro enfoque en la causa raíz de los síntomas tanto como sea posible. Con la complejidad de la ingeniería de software y la seguridad del software, es básicamente imposible crear diez categorías sin cierto nivel de superposición.



Evolución y Trazabilidad

- **A01:2025 - Control de acceso roto** mantiene su posición en el puesto número 1 como el riesgo de seguridad de aplicaciones más grave; Los datos aportados indican que, en promedio, el 3,73% de las aplicaciones probadas tenían una o más de las 40 enumeraciones de debilidades comunes (CWE) de esta categoría. Como lo indica la línea discontinua en la figura anterior, la falsificación de solicitudes del lado del servidor (SSRF) se ha incluido en esta categoría.

- **A02:2025 - Mala configuración de seguridad** subió del puesto número 5 en 2021 al puesto número 2 en 2025. Las configuraciones incorrectas son más frecuentes en los datos de este ciclo. El 3,00% de las aplicaciones probadas tenían uno o más de los 16 CWE de esta categoría. Esto no es sorprendente, ya que la ingeniería de software continúa aumentando la cantidad de comportamiento de una aplicación basado en configuraciones.

- **A03:2025 - Fallas en la cadena de suministro de software** es una expansión de A06:2021-Componentes vulnerables y obsoletos incluir un alcance más amplio de compromisos que ocurren dentro o en todo el ecosistema de dependencias de software, sistemas de construcción e infraestructura de distribución. Esta categoría fue votada abrumadoramente como una de las principales preocupaciones en la encuesta comunitaria. Esta categoría tiene 5 CWE y una presencia limitada en los datos recopilados, pero creemos que esto se debe a los desafíos en las pruebas y esperamos que las pruebas se pongan al día en esta área. Esta categoría tiene la menor cantidad de ocurrencias en los datos, pero también los puntajes promedio más altos de

exploits e impacto de CVE.

- **A04:2025 - Fallos criptográficos** cae dos puestos del puesto número 2 al número 4 en el ranking. Los datos aportados indican que, en promedio, el 3,80% de las aplicaciones tienen uno o más de los 32 CWE de esta categoría. Esta categoría a menudo conduce a la exposición de datos confidenciales o a la vulneración del sistema.

- **A05:2025 - Inyección** cae dos puestos del #3 al #5 en el ranking, manteniendo su posición en relación con Fallas criptográficas y diseño inseguro. La inyección es una de las categorías más probadas, con el mayor número de CVE asociados con los 38 CWE de esta categoría. La inyección incluye una variedad de problemas, desde secuencias de comandos entre sitios (alta frecuencia/bajo impacto) hasta vulnerabilidades de inyección SQL (baja frecuencia/alto impacto).

- **A06:2025 - Diseño inseguro** se desliza dos puestos del puesto 4 al 6 en el ranking a medida que la mala configuración de seguridad y las fallas en la cadena de suministro de software lo superan. Esta categoría se introdujo en 2021 y hemos visto mejoras notables en la industria relacionadas con el modelado de amenazas y un mayor énfasis en el diseño seguro.

- **A07:2025 - Fallos de autenticación** mantiene su posición en el puesto 7 con un ligero cambio de nombre (antes lo era “ Fallos de identificación y autenticación “) para reflejar con mayor precisión los 36 CWE de esta categoría. Esta categoría sigue siendo importante, pero el mayor uso de marcos estandarizados para la autenticación parece estar teniendo efectos beneficiosos en la aparición de fallos de autenticación.

- **A08:2025 - Fallos de integridad de software o datos** continúa en el puesto número 8 de la lista. Esta categoría se centra en la falta de mantenimiento de los límites de confianza y de verificación de la integridad del software, el código y los artefactos de datos a un nivel inferior al de las fallas de la cadena de suministro de software.

- **A09:2025 - Fallas de registro y alerta** conserva su posición en el puesto número 9. Esta categoría tiene un ligero cambio de nombre (anteriormente Fallas en el registro y monitoreo de seguridad ”) enfatizar la importancia de la funcionalidad de alerta necesaria para inducir acciones apropiadas sobre eventos de registro relevantes. Un buen registro sin alertas tiene un valor mínimo para identificar incidentes de seguridad. Esta categoría siempre estará

subrepresentada en los datos y los participantes de la encuesta comunitaria volvieron a votar para ocupar un lugar en la lista.

- **A10:2025 - Mal manejo de condiciones excepcionales** es una nueva categoría para 2025. Esta categoría contiene 24 CWE que se centran en el manejo inadecuado de errores, errores lógicos, apertura fallida y otros escenarios relacionados derivados de condiciones anormales que los sistemas pueden encontrar.

Metodología

Esta entrega del Top Ten sigue estando basada en datos, pero no ciegamente en datos. Clasificamos 12 categorías según los datos aportados y permitimos que dos fueran promovidas o destacadas por las respuestas de la encuesta comunitaria. Lo hacemos por una razón fundamental: examinar los datos aportados es esencialmente mirar al pasado. Los investigadores de seguridad de aplicaciones dedican tiempo a identificar nuevas vulnerabilidades y desarrollar nuevos métodos de prueba. Se necesitan semanas o años para integrar estas pruebas en herramientas y procesos. Para cuando podamos probar de manera confiable una debilidad a gran escala, es posible que hayan pasado años. También existen riesgos importantes que quizás nunca podamos probar de manera confiable y que estén presentes en los datos. Para equilibrar esa visión,Utilizamos una encuesta comunitaria para preguntar a los profesionales de desarrollo y seguridad de aplicaciones en primera línea qué consideran riesgos esenciales que pueden estar subrepresentados en los datos de prueba.

Cómo se estructuran las categorías

Algunas categorías han cambiado con respecto a la entrega anterior del OWASP Top Ten. Aquí hay un resumen de alto nivel de los cambios de categoría.

En esta iteración, solicitamos datos, sin restricciones en los CWE como lo hicimos para la edición de 2021. Solicitamos la cantidad de aplicaciones probadas para un año determinado (a partir de 2021) y la cantidad de aplicaciones con al menos una instancia de un CWE encontradas en las pruebas. Este formato nos permite realizar un seguimiento de la prevalencia de cada CWE dentro de la población de aplicaciones. Ignoramos la frecuencia para nuestros propósitos; si bien puede ser necesaria para otras situaciones, solo oculta la prevalencia real en la población de aplicación. Si una aplicación tiene cuatro instancias de un CWE o 4000 instancias no es parte del cálculo para el Top Ten. Especialmente porque los evaluadores

manuales tienden a enumerar una vulnerabilidad solo una vez sin importar cuántas veces se repita en una aplicación, mientras que los marcos de prueba automatizados enumeran cada instancia de una vulnerabilidad como única.Pasamos de aproximadamente 30 CWE en 2017, a casi 400 CWE en 2021, a 589 CWE en esta edición para analizar en el conjunto de datos. Planeamos realizar análisis de datos adicionales como complemento en el futuro. Este aumento significativo en el número de CWE requiere cambios en la forma en que se estructuran las categorías.

Pasamos varios meses agrupando y categorizando los CWE y podríamos haber continuado durante meses más. Tuvimos que parar en algún momento. Existen tipos de CWE tanto de causa raíz como de síntomas, donde los tipos de causa raíz son como “Fallo criptográfico” y “Configuración incorrecta”, en contraste con tipos de síntomas como “Exposición a datos confidenciales” y “Denegación de servicio” Decidimos centrarnos en la causa raíz siempre que sea posible, ya que es más lógico para proporcionar orientación sobre identificación y remediación. Centrarse en la causa raíz por encima del síntoma no es un concepto nuevo; el Top Ten ha sido una mezcla de síntoma y causa raíz. Las CWE también son una mezcla de síntoma y causa raíz; simplemente estamos siendo más deliberados al denunciarlo. Hay un promedio de 25 CWE por categoría en esta entrega, con límites inferiores de 5 CWE para A02:2025-Fallas en la cadena de suministro de software y A09:2025 Fallas de registro y alerta a 40 CWE en A01:2025-Control de acceso roto. Tomamos la decisión de limitar el número de CWE en una categoría a 40. Esta estructura de categorías actualizada ofrece beneficios de capacitación adicionales, ya que las empresas pueden centrarse en CWE que tengan sentido para un lenguaje/marco.

Se nos ha preguntado por qué no pasar a una lista de 10 CWE como Top 10, similar a las 25 debilidades de software más peligrosas de MITRE. Hay dos razones principales por las que utilizamos múltiples CWE en categorías. En primer lugar, no todos los CWE existen en todos los lenguajes o marcos de programación. Esto genera problemas con las herramientas y los programas de capacitación y concientización, ya que es posible que parte del Top Ten no sea aplicable. La segunda razón es que existen múltiples CWE para vulnerabilidades comunes. Por ejemplo, existen múltiples CWE para inyección general, inyección de comandos, secuencias de comandos entre sitios, contraseñas codificadas, falta de validación, desbordamientos de buffer, almacenamiento de texto sin cifrar de información confidencial y muchos otros. Dependiendo de la organización o del evaluador, se pueden utilizar diferentes CWE.Al utilizar una categoría con múltiples CWE, podemos ayudar a aumentar la base y la conciencia de los diferentes tipos de debilidades que pueden ocurrir bajo un nombre de categoría común. En esta edición del Top Ten 2025, hay 248 CWE dentro de las 10

categorías. Hay un total de 968 CWE en el Diccionario descargable de MITRE en el momento de este lanzamiento.

Cómo se utilizan los datos para seleccionar categorías

De manera similar a lo que hicimos para la edición de 2021, aprovechamos los datos de CVE para *Explotabilidad* y *Impacto (técnico)*. Descargamos OWASP Dependency Check y extrajimos las puntuaciones CVSS Exploit e Impact, agrupándolas por CWE relevantes enumerados con los CVE. Se necesitó bastante investigación y esfuerzo, ya que todos los CVE tienen puntuaciones CVSSv2, pero hay fallas en CVSSv2 que CVSSv3 debería abordar. Después de un cierto momento, a todos los CVE también se les asigna una puntuación CVSSv3. Además, se actualizaron los rangos de puntuación y las fórmulas entre CVSSv2 y CVSSv3.

En CVSSv2, tanto Exploit como (Technical) Impact podrían ser de hasta 10,0, pero la fórmula los reduciría al 60% para Exploit y al 40% para Impact. En CVSSv3, el máximo teórico se limitó a 6,0 para Exploit y 4,0 para Impact. Teniendo en cuenta la ponderación, la puntuación de impacto aumentó, casi un punto y medio en promedio en CVSSv3, y la explotabilidad disminuyó casi medio punto en promedio.

Hay aproximadamente 175 000 registros (frente a los 125 000 de 2021) de CVE asignados a CWE en la Base de datos nacional de vulnerabilidades (NVD), extraídos de OWASP Dependency Check. Además, hay 643 CWE únicos asignados a CVE (frente a 241 en 2021). Dentro de los casi 220 000 CVE que se extrajeron, 160 000 tenían puntuaciones CVSS v2, 156 000 tenían puntuaciones CVSS v3 y 6 000 tenían puntuaciones CVSS v4. Muchos CVE tienen múltiples puntuaciones, por lo que suman más de 220k.

Para el Top Ten 2025, calculamos las puntuaciones promedio de explotación e impacto de la siguiente manera. Agrupamos todos los CVE con puntuaciones CVSS por CWE y ponderamos las puntuaciones de explotación e impacto por el porcentaje de la población que tenía CVSSv3, así como la población restante con puntuaciones CVSSv2, para obtener un promedio general. Asignamos estos promedios a los CWE en el conjunto de datos para usarlos como puntuación de Explotación e Impacto (Técnico) para la otra mitad de la ecuación de riesgo.

¿Por qué no utilizar CVSS v4.0?, te preguntarás Esto se debe a que el algoritmo de puntuación se modificó fundamentalmente y ya no proporciona fácilmente la

Explotar o *Impacto* puntuaciones como lo hacen CVSS v2 y CVSSv3. Intentaremos encontrar una forma de utilizar la puntuación CVSS v4.0 para futuras versiones del Top Ten, pero no pudimos determinar una forma oportuna de hacerlo para la edición de 2025.

Por qué utilizamos una encuesta comunitaria

Los resultados de los datos se limitan en gran medida a lo que la industria puede probar de forma automatizada. Hable con un profesional experimentado de AppSec y le informará sobre las cosas que encuentre y las tendencias que vea que aún no están en los datos. Se necesita tiempo para que las personas desarrollen metodologías de prueba para ciertos tipos de vulnerabilidad y luego más tiempo para que esas pruebas se automaticen y se ejecuten contra una gran población de aplicaciones. Todo lo que encontramos es una mirada retrospectiva al pasado y es posible que falten tendencias del año pasado, que no están presentes en los datos.

Por lo tanto, sólo elegimos ocho de las diez categorías de los datos porque están incompletas. Las otras dos categorías provienen de la encuesta de las 10 mejores comunidades. Permite a los profesionales en primera línea votar por lo que consideran los mayores riesgos que podrían no estar en los datos (y que tal vez nunca se expresen en los datos).

Gracias a nuestros colaboradores de datos

Las siguientes organizaciones (junto con varios donantes anónimos) donaron amablemente datos para más de 2,8 millones de aplicaciones para convertir este en el conjunto de datos de seguridad de aplicaciones más grande y completo. Sin ti esto no sería posible.

- Accenture (Prague)
- Anonymous (multiple)
- Bugcrowd
- Contrast Security
- CyptoNet Labs
- Intuitor SoftTech Services
- Orca Security
- Probley
- Semgrep
- Sonar
- usd AG
- Veracode
- Wallarm

|600

Autores principales

- Andrew van der Stock - X: @vanderaj
- Brian Glas - X: @infosecdad
- Neil Smithline - X: @appsecneil
- Tanya Janca - X: @shehackspurple
- Torsten Gigler - Mastodonte: @ torsten_gigler@infosec.exchange

Candidato a liberación

Este candidato de lanzamiento se lanzó originalmente el 6 de noviembre de 2025.

Problemas de registro y solicitudes de extracción

Por favor registre cualquier corrección o problema:

Enlaces del proyecto:

- Página de inicio
- repositorio de GitHub

OwaspTop10-2025-A01-Broken-Access-Control-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A01:2025 Control de acceso roto

Contexto.

Manteniendo su posición en el puesto número 1 del Top Ten, se encontró que el 100% de las aplicaciones probadas tenían algún tipo de control de acceso roto. Los CWE notables incluidos son *CWE-200: Exposición de información confidencial a un actor no autorizado*, *CWE-201: Exposición de información confidencial a través de datos enviados*, *Falsificación de solicitudes del lado del servidor (SSRF) CWE-918*, y *CWE-352: Falsificación de solicitudes entre sitios (CSRF)*. Esta categoría tiene el mayor número de ocurrencias en los datos aportados y el segundo mayor número de CVE relacionados.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
40	20,15%	3,74%	100,00%	42,93%	7.04

Descripción.

El control de acceso aplica políticas tales que los usuarios no pueden actuar fuera de sus permisos previstos. Las fallas generalmente conducen a la divulgación no autorizada de información, modificación o destrucción de todos los datos o al

desempeño de una función comercial fuera de los límites del usuario. Las vulnerabilidades comunes de control de acceso incluyen:

- Violación del principio del mínimo privilegio, comúnmente conocido como denegación por defecto, donde el acceso sólo debe concederse para capacidades, roles o usuarios particulares, pero está disponible para cualquier persona.
- Omitir las comprobaciones de control de acceso modificando la URL (manipulación de parámetros o navegación forzada), el estado interno de la aplicación o la página HTML, o utilizando una herramienta de ataque que modifica las solicitudes de API.
- Permitir ver o editar la cuenta de otra persona proporcionando su identificador único (referencias directas de objetos inseguras)
- Una API accesible a la que le faltan controles de acceso para POST, PUT y DELETE.
- Elevación del privilegio. Actuar como usuario sin iniciar sesión o actuar como administrador cuando se inicia sesión como usuario.
- Manipulación de metadatos, como reproducir o manipular un token de control de acceso JSON Web Token (JWT), una cookie o un campo oculto manipulado para elevar privilegios o abusar de la invalidación de JWT.
- La configuración incorrecta de CORS permite el acceso a la API desde orígenes no autorizados o no confiables.
- Forzar la navegación (adivinar URL) a páginas autenticadas como usuario no autenticado o a páginas privilegiadas como usuario estándar.

Cómo prevenir.

El control de acceso solo es efectivo cuando se implementa en código confiable del lado del servidor o API sin servidor, donde el atacante no puede modificar la verificación del control de acceso ni los metadatos.

- Salvo recursos públicos, denegar por defecto.
- Implemente mecanismos de control de acceso una vez y reutilícelos en toda la aplicación, incluida la minimización del uso de recursos compartidos entre orígenes (CORS).
- Los controles de acceso a los modelos deberían imponer la propiedad de los registros en lugar de permitir a los usuarios crear, leer, actualizar o eliminar cualquier registro.
- Los modelos de dominio deben hacer cumplir requisitos únicos de límites comerciales de aplicaciones.
- Deshabilite la lista de directorios del servidor web y asegúrese de que los

metadatos de los archivos (por ejemplo,.git) y los archivos de respaldo no estén presentes dentro de las raíces web.
- Registre fallas en el control de acceso y alerte a los administradores cuando corresponda (por ejemplo, fallas repetidas).
- Implemente límites de velocidad en el acceso a la API y al controlador para minimizar el daño causado por las herramientas de ataque automatizadas.
- Los identificadores de sesión con estado deben invalidarse en el servidor después de cerrar sesión. Los tokens JWT sin estado deben tener una vida útil corta para minimizar la ventana de oportunidad para un atacante. Para los JWT de mayor duración, se recomienda encarecidamente seguir los estándares OAuth para revocar el acceso.
- Utilice kits de herramientas o patrones bien establecidos que proporcionen controles de acceso declarativos y simples.

Los desarrolladores y el personal de control de calidad deben incluir control de acceso funcional en sus pruebas unitarias y de integración.

Ejemplos de escenarios de ataque.

Escenario #1: La aplicación utiliza datos no verificados en una llamada SQL que accede a la información de la cuenta:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Un atacante puede simplemente modificar el parámetro ‘acct’ del navegador para enviar cualquier número de cuenta deseado. Si no se verifica correctamente, el atacante puede acceder a la cuenta de cualquier usuario.

https://*example.com/app/accountInfo?acct=notmyacct*

Escenario #2: Un atacante simplemente obliga a los navegadores a apuntar a las URL. Se requieren derechos de administrador para acceder a la página de administración.

https://*example.com/app/getappInfo*
https://*example.com/app/admin_getappInfo*

Si un usuario no autenticado puede acceder a cualquiera de las páginas, es una falla. Si alguien que no es administrador puede acceder a la página de administración, eso es una falla.

Escenario #3: Una aplicación coloca todo su control de acceso en su front-end. Mientras el atacante no pueda llegar https://example.com/app/admin_getappInfo Debido a que el código JavaScript se ejecuta en el navegador, simplemente pueden ejecutar:

$ curl https://*example.com/app/admin_getappInfo*

desde la línea de comando.

Referencias.

- OWASP Proactive Controls: C1: Implement Access Control
- OWASP Application Security Verification Standard: V8 Authorization
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Authorization
- PortSwigger: Exploiting CORS misconfiguration
- OAuth: Revoking Access

List of Mapped CWEs

- CWE-22 Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)
- CWE-23 Relative Path Traversal
- CWE-36 Absolute Path Traversal
- CWE-59 Improper Link Resolution Before File Access (‘Link Following’)
- CWE-61 UNIX Symbolic Link (Symlink) Following
- CWE-65 Windows Hard Link
- CWE-200 Exposure of Sensitive Information to an Unauthorized Actor
- CWE-201 Exposure of Sensitive Information Through Sent Data
- CWE-219 Storage of File with Sensitive Data Under Web Root
- CWE-276 Incorrect Default Permissions
- CWE-281 Improper Preservation of Permissions

- CWE-282 Improper Ownership Management
- CWE-283 Unverified Ownership
- CWE-284 Improper Access Control
- CWE-285 Improper Authorization
- CWE-352 Cross-Site Request Forgery (CSRF)
- CWE-359 Exposure of Private Personal Information to an Unauthorized Actor
- CWE-377 Insecure Temporary File
- CWE-379 Creation of Temporary File in Directory with Insecure Permissions
- CWE-402 Transmission of Private Resources into a New Sphere (‘Resource Leak’)
- CWE-424 Improper Protection of Alternate Path
- CWE-425 Direct Request (‘Forced Browsing’)
- CWE-441 Unintended Proxy or Intermediary (‘Confused Deputy’)
- CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere
- CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory
- CWE-540 Inclusion of Sensitive Information in Source Code
- CWE-548 Exposure of Information Through Directory Listing
- CWE-552 Files or Directories Accessible to External Parties
- CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key
- CWE-601 URL Redirection to Untrusted Site (‘Open Redirect’)
- CWE-615 Inclusion of Sensitive Information in Source Code Comments
- CWE-639 Authorization Bypass Through User-Controlled Key
- CWE-668 Exposure of Resource to Wrong Sphere
- CWE-732 Incorrect Permission Assignment for Critical Resource

- CWE-749 Exposed Dangerous Method or Function
- CWE-862 Missing Authorization
- CWE-863 Incorrect Authorization
- CWE-918 Server-Side Request Forgery (SSRF)
- CWE-922 Insecure Storage of Sensitive Information
- CWE-1275 Sensitive Cookie with Improper SameSite Attribute

OwaspTop10-2025-A02-Mala-configuración-de-seguridad-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A02:2025 Mala configuración de seguridad

Contexto.

Pasando del puesto número 5 en la edición anterior, se encontró que el 100% de las aplicaciones probadas tenían algún tipo de configuración incorrecta, con una tasa de incidencia promedio del 3,00% y más de 719 000 ocurrencias de una enumeración de debilidad común (CWE) en esta categoría de riesgo. Con más cambios hacia software altamente configurable, no sorprende ver que esta categoría avance. Los CWE notables incluidos son *Configuración de CWE-16* y *CWE-611 Restricción incorrecta de la referencia de entidad externa XML (XXE)* .

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
16	27,70%	3,00%	100,00%	52,35%	7,96

Descripción.

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

La aplicación podría ser vulnerable si:

- Falta un refuerzo de seguridad adecuado en cualquier parte de la pila de aplicaciones o permisos configurados incorrectamente en los servicios en la nube.
- Se habilitan o instalan funciones innecesarias (por ejemplo, puertos, servicios, páginas, cuentas, marcos de prueba o privilegios innecesarios).
- Las cuentas predeterminadas y sus contraseñas aún están habilitadas y sin cambios.
- Falta de configuración central para interceptar mensajes de error excesivos. El manejo de errores revela rastros de pila u otros mensajes de error demasiado informativos a los usuarios.
- Para los sistemas actualizados, las últimas funciones de seguridad están deshabilitadas o no están configuradas de forma segura.
- Priorización excesiva de la compatibilidad con versiones anteriores que conduce a una configuración insegura.
- Las configuraciones de seguridad en los servidores de aplicaciones, marcos de aplicaciones (por ejemplo, Struts, Spring, ASP.NET), bibliotecas, bases de datos, etc., no están configuradas en valores seguros.
- El servidor no envía encabezados o directivas de seguridad, o no están configurados para valores seguros.

Sin un proceso concertado y repetible de endurecimiento de la configuración de seguridad de las aplicaciones, los sistemas corren un mayor riesgo.

Cómo prevenir.

Se deben implementar procesos de instalación seguros, que incluyan:

- Un proceso de endurecimiento repetible que permite la implementación rápida y sencilla de otro entorno que esté bloqueado adecuadamente. Los entornos de desarrollo, control de calidad y producción deben configurarse de manera idéntica, utilizando diferentes credenciales en cada entorno. Este proceso debe automatizarse para minimizar el esfuerzo necesario para configurar un nuevo entorno seguro.
- Una plataforma mínima sin características, componentes, documentación ni muestras innecesarias. Elimine o no instale funciones y marcos no utilizados.
- Una tarea para revisar y actualizar las configuraciones apropiadas para todas las notas de seguridad, actualizaciones y parches como parte del proceso de administración de parches (ver A03:2025- Fallas en la cadena de suministro de software). Revise los permisos de almacenamiento en la nube (por

ejemplo, los permisos del depósito S3).
- Una arquitectura de aplicación segmentada proporciona una separación efectiva y segura entre componentes o inquilinos, con segmentación, contenedorización o grupos de seguridad en la nube (ACL).
- Envío de directivas de seguridad a clientes, por ejemplo, encabezados de seguridad.
- Un proceso automatizado para verificar la efectividad de las configuraciones y ajustes en todos los entornos.
- Agregue de forma proactiva una configuración central para interceptar mensajes de error excesivos como copia de seguridad.
- Si estas variaciones no están automatizadas, deberán verificarse manualmente anualmente como mínimo.

Ejemplos de escenarios de ataque.

Escenario #1: El servidor de aplicaciones viene con aplicaciones de muestra que no se eliminan del servidor de producción. Estas aplicaciones de muestra tienen fallas de seguridad conocidas que los atacantes utilizan para comprometer el servidor. Supongamos que una de estas aplicaciones es la consola de administración y las cuentas predeterminadas no se modificaron. En ese caso, el atacante inicia sesión con la contraseña predeterminada y toma el control.

Escenario #2: La lista de directorios no está deshabilitada en el servidor. Un atacante descubre que puede simplemente enumerar directorios. El atacante encuentra y descarga las clases Java compiladas, que descompilan y realizan ingeniería inversa para ver el código. Luego, el atacante encuentra una falla grave en el control de acceso en la aplicación.

Escenario #3: La configuración del servidor de aplicaciones permite devolver a los usuarios mensajes de error detallados, como rastros de pila. Esto expone potencialmente información confidencial o fallas subyacentes, como versiones de componentes que se sabe que son vulnerables.

Escenario #4: Un proveedor de servicios en la nube (CSP) tiene de forma predeterminada permisos para compartir abiertos a Internet. Esto permite acceder a datos confidenciales almacenados en el almacenamiento en la nube.

Referencias.

- OWASP Testing Guide: Configuration Management

- OWASP Testing Guide: Testing for Error Codes
- Application Security Verification Standard 5.0.0
- NIST Guide to General Server Hardening
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration
- ScienceDirect: Security Misconfiguration

List of Mapped CWEs

- CWE-5 J2EE Misconfiguration: Data Transmission Without Encryption
- CWE-11 ASP.NET Misconfiguration: Creating Debug Binary
- CWE-13 ASP.NET Misconfiguration: Password in Configuration File
- CWE-15 External Control of System or Configuration Setting
- CWE-16 Configuration
- CWE-260 Password in Configuration File
- CWE-315 Cleartext Storage of Sensitive Information in a Cookie
- CWE-489 Active Debug Code
- CWE-526 Exposure of Sensitive Information Through Environmental Variables
- CWE-547 Use of Hard-coded, Security-relevant Constants
- CWE-611 Improper Restriction of XML External Entity Reference
- CWE-614 Sensitive Cookie in HTTPS Session Without ‘Secure’ Attribute
- CWE-776 Improper Restriction of Recursive Entity References in DTDs (‘XML Entity Expansion’)
- CWE-942 Permissive Cross-domain Policy with Untrusted Domains
- CWE-1004 Sensitive Cookie Without ‘HttpOnly’ Flag
- CWE-1174 ASP.NET Misconfiguration: Improper Model Validation

OwaspTop10-2025-A03-Fallos-en-la-cadena-de-suministro-de-software-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A03:2025 Fallos en la cadena de suministro de software

Contexto.

Este tema ocupó el primer lugar en la encuesta de las comunidades, con exactamente el 50% de los encuestados ubicándolo en el puesto número 1. Desde que apareció inicialmente en el Top 10 de 2013 como “A9 – Uso de componentes con vulnerabilidades conocidas”, el riesgo ha crecido en alcance para incluir todas las fallas de la cadena de suministro, no solo aquellas que involucran vulnerabilidades conocidas. A pesar de este mayor alcance, las fallas en la cadena de suministro siguen siendo un desafío para identificar, ya que solo 11 vulnerabilidades y exposiciones comunes (CVE) tienen los CWE relacionados. Sin embargo, cuando se prueba y se informa en los datos aportados, esta categoría tiene la tasa de incidencia promedio más alta, con un 5,19%. Los CWE relevantes son *CWE-477: Uso de funciones obsoletas, CWE-1104: Uso de componentes de terceros sin mantenimiento*, CWE-1329: *Dependencia de un componente que no se puede actualizar*, y *CWE-1395: Dependencia de componentes de terceros vulnerables*.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
5	8,81%	5,19%	65,42%	28,93%	8.17

Descripción.

Las fallas en la cadena de suministro de software son averías u otros compromisos en el proceso de creación, distribución o actualización de software. A menudo son causados por vulnerabilidades o cambios maliciosos en el código, las herramientas u otras dependencias de terceros en las que se basa el sistema.

Es probable que usted sea vulnerable si:

- Si no realiza un seguimiento cuidadoso de las versiones de todos los componentes que utiliza (tanto del lado del cliente como del lado del servidor). Esto incluye componentes que utiliza directamente, así como dependencias anidadas (transitivas).
- Si el software es vulnerable, no es compatible o está desactualizado. Esto incluye el sistema operativo, el servidor web/de aplicaciones, el sistema de gestión de bases de datos (DBMS), las aplicaciones, las API y todos los componentes, los entornos de ejecución y las bibliotecas.
- Si no busca vulnerabilidades con regularidad y no se suscribe a boletines de seguridad relacionados con los componentes que utiliza.
- Si no tiene un proceso de gestión de cambios o seguimiento de cambios dentro de su cadena de suministro, incluido el seguimiento de IDE, extensiones y actualizaciones de IDE, cambios en el repositorio de código de su organización, entornos aislados, repositorios de imágenes y bibliotecas, la forma en que se crean y almacenan los artefactos, etc. Cada parte de su cadena de suministro debe documentarse, especialmente los cambios.
- Si no ha endurecido cada parte de su cadena de suministro, con un enfoque especial en el control de acceso y la aplicación del mínimo privilegio.
- Si sus sistemas de cadena de suministro no tienen ninguna separación de funciones. Ninguna persona debería poder escribir código y promoverlo hasta su producción sin la supervisión de otro ser humano.
- Si a los desarrolladores, DevOps o profesionales de infraestructura se les permite descargar y utilizar componentes de fuentes no confiables para su uso en producción.

- Si no corrige o actualiza la plataforma, los marcos y las dependencias subyacentes de manera oportuna y basada en riesgos. Esto sucede comúnmente en entornos donde el parche es una tarea mensual o trimestral bajo control de cambios, lo que deja a las organizaciones abiertas a días o meses de exposición innecesaria antes de corregir vulnerabilidades.
- Si los desarrolladores de software no prueban la compatibilidad de bibliotecas actualizadas, mejoradas o parcheadas.
- Si no asegura las configuraciones de cada parte de su sistema (ver A02:2025-Configuración incorrecta de seguridad).
- Si tiene una pipeline CI/CD compleja que utiliza muchos componentes pero tiene una seguridad más débil que el resto de su aplicación.

Cómo prevenir.

Debería existir un proceso de gestión de parches para:

- Conozca su lista de materiales de software (SBOM) de todo su software y administre el diccionario SBOM de forma centralizada.
- Realice un seguimiento no sólo de sus propias dependencias, sino también de sus dependencias (transitivas), etc.
- Elimine dependencias no utilizadas, funciones, componentes, archivos y documentación innecesarios. Reducción de la superficie de ataque.
- Inventario continuo de las versiones de los componentes del lado del cliente y del lado del servidor (por ejemplo, marcos, bibliotecas) y sus dependencias utilizando herramientas como versiones, OWASP Dependency Check, reveile.js, etc.
- Supervise continuamente fuentes como Vulnerabilidad y exposiciones comunes (CVE) y Base de datos nacional de vulnerabilidades (NVD) para detectar vulnerabilidades en los componentes que utiliza. Utilice análisis de composición de software, cadena de suministro de software o herramientas SBOM centradas en la seguridad para automatizar el proceso. Suscríbase a alertas por correo electrónico sobre vulnerabilidades de seguridad relacionadas con los componentes que utiliza.
- Obtenga componentes únicamente de fuentes oficiales (confiables) a través de enlaces seguros. Prefiera paquetes firmados para reducir la posibilidad de incluir un componente malicioso modificado (consulte A08:2025-Fallos de integridad de software y datos).
- Elegir deliberadamente qué versión de una dependencia utiliza y actualizarla solo cuando sea necesario.
- Supervise bibliotecas y componentes que no reciben mantenimiento o que no crean parches de seguridad para versiones anteriores. Si no es posible aplicar

parches, considere implementar un parche virtual para monitorear, detectar o proteger contra el problema descubierto.
- Actualice su CI/CD, IDE y cualquier otra herramienta para desarrolladores con regularidad
- Trate los componentes de su pipeline de CI/CD como parte de este proceso; fortalézcalos, monitoreelos y documente los cambios en consecuencia

Debería existir un proceso de gestión de cambios o un sistema de seguimiento para realizar un seguimiento de los cambios en:

- Su configuración de CI/CD (todas las herramientas de compilación y pipeline)
- Tu repositorio de código
- Sandbox areas
- IDE de desarrolladores
- Sus herramientas SBOM y artefactos creados
- Sus sistemas de registro y registros
- Integraciones de terceros, como SaaS
- Repositorio de artefactos
- Registro de contenedores

Mejore los siguientes sistemas, que incluyen habilitar MFA y bloquear IAM: * Su repositorio de código (que incluye no registrar secretos, proteger ramas y copias de seguridad) * Estaciones de trabajo para desarrolladores (parches regulares, MFA, monitoreo y más) * Su servidor de compilación y CI/CD (separación de funciones, control de acceso, compilaciones firmadas, secretos con alcance ambiental, registros a prueba de manipulaciones, más) * Sus artefactos (garantice la integridad mediante providencia, firma y sellado de tiempo, promueva los artefactos en lugar de reconstruirlos para cada entorno, asegúrese de que las compilaciones sean inmutables) * La infraestructura como código se administra como todo el código, incluido el uso de PR y control de versiones

Cada organización debe garantizar un plan continuo para monitorear, clasificar y aplicar actualizaciones o cambios de configuración durante la vida útil de la aplicación o cartera.

Ejemplos de escenarios de ataque.

Escenario #1: Un proveedor confiable se ve comprometido con malware, lo que hace que sus sistemas informáticos se vean comprometidos cuando actualiza. El ejemplo más famoso de esto es probablemente:

- El compromiso de SolarWinds de 2019 que llevó a que ~18.000

organizaciones se vieran comprometidas. https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack

Escenario #2: Un proveedor confiable se ve comprometido de tal manera que se comporta maliciosamente solo bajo una condición específica.

- El robo de Bybit de 1.500 millones de dólares en 2025 causado por un ataque a la cadena de suministro en un software de billetera que solo se ejecutaba cuando se usaba la billetera de destino. https://thehackernews.com/2025/02/bybit-hack-traced-to-safewallet-supply.html

Escenario #3: El ataque a la cadena de suministro de GlassWorm en 2025 contra el mercado de VS Code hizo que actores maliciosos implementaran código invisible y autorreplicante en una extensión legítima en VS Marketplace, así como varias extensiones en OpenVSX Marketplace, que se actualizaban automáticamente en las máquinas de los desarrolladores. El gusano inmediatamente recopiló secretos locales de las máquinas de los desarrolladores, intentó establecer comando y control y, si era posible, vació las billeteras criptográficas de los desarrolladores. Este ataque a la cadena de suministro fue extremadamente avanzado, de rápida propagación y dañino, y al apuntar a las máquinas de los desarrolladores demostró que los propios desarrolladores son ahora objetivos principales de los ataques a la cadena de suministro.

Escenario #4: Los componentes normalmente se ejecutan con los mismos privilegios que la propia aplicación, por lo que las fallas en cualquier componente pueden tener un impacto grave. Estas fallas pueden ser accidentales (por ejemplo, un error de codificación) o intencionales (por ejemplo, una puerta trasera en un componente). Algunos ejemplos de vulnerabilidades de componentes explotables descubiertas son:

- Se ha culpado a CVE-2017-5638, una vulnerabilidad de ejecución remota de código de Struts 2 que permite la ejecución de código arbitrario en el servidor, de infracciones importantes.
- Si bien la Internet de las cosas (IoT) suele ser difícil o imposible de parchear, la importancia de parchearlas puede ser grande (por ejemplo, dispositivos biomédicos).

Existen herramientas automatizadas para ayudar a los atacantes a encontrar sistemas sin parches o mal configurados. Por ejemplo, el IoT de Shodan El motor de búsqueda puede ayudarle a encontrar dispositivos que aún sufren la vulnerabilidad Heartbleed parcheada en abril de 2014.

Referencias

- OWASP Application Security Verification Standard: V15 Secure Coding and Architecture
- OWASP Cheat Sheet Series: Dependency Graph SBOM
- OWASP Cheat Sheet Series: Vulnerable Dependency Management
- OWASP Dependency-Track
- OWASP CycloneDX
- OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling
- OWASP Dependency Check (for Java and .NET libraries)
- OWASP Testing Guide - Map Application Architecture (OTG-INFO-010)
- OWASP Virtual Patching Best Practices
- The Unfortunate Reality of Insecure Libraries
- MITRE Common Vulnerabilities and Exposures (CVE) search
- National Vulnerability Database (NVD)
- Retire.js for detecting known vulnerable JavaScript libraries
- GitHub Advisory Database
- Ruby Libraries Security Advisory Database and Tools
- SAFECode Software Integrity Controls (PDF)
- Glassworm supply chain attack
- PhantomRaven supply chain attack campaign

List of Mapped CWEs

- CWE-447 Use of Obsolete Function
- CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities
- CWE-1104 Use of Unmaintained Third Party Components
- CWE-1329 Reliance on Component That is Not Updateable
- CWE-1395 Dependency on Vulnerable Third-Party Component

OwaspTop10-2025-A04-Fallos-criptográficos-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A04:2025 Fallos criptográficos

Contexto.

Bajando dos posiciones hasta el puesto número 4, esta debilidad se centra en fallas relacionadas con la falta de criptografía, criptografía insuficientemente fuerte, filtración de claves criptográficas y errores relacionados. Tres de las enumeraciones de debilidades comunes (CWE) más comunes en este riesgo implicaron el uso de un generador de números pseudoaleatorios débil: *CWE-327 Uso de un algoritmo criptográfico roto o riesgoso, CWE-331: entropía insuficiente*, *CWE-1241: Uso de un algoritmo predecible en un generador de números aleatorios*, y *CWE-338 Uso de un generador de números pseudoaleatorios (PRNG) criptográficamente débil*.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
32	13,77%	3,80%	100,00%	47,74%	7.23

Descripción.

En términos generales, todos los datos en tránsito deben estar cifrados en el capa de transporte (Capa OSI 4). Los obstáculos anteriores, como el rendimiento de la

CPU y la gestión de claves/certificados privados, ahora los manejan las CPU que tienen instrucciones diseñadas para acelerar el cifrado (por ejemplo: Soporte AES) y la gestión de claves privadas y certificados se simplifica mediante servicios como LetsEncrypt.org con importantes proveedores de nube que brindan servicios de gestión de certificados aún más integrados para sus plataformas específicas.

Más allá de proteger la capa de transporte, es importante determinar qué datos necesitan cifrado en reposo, así como qué datos necesitan cifrado adicional en tránsito (en el capa de aplicación, capa OSI 7). Por ejemplo, las contraseñas, los números de tarjetas de crédito, los registros médicos, la información personal y los secretos comerciales requieren protección adicional, especialmente si esos datos están sujetos a las leyes de privacidad, por ejemplo, el Reglamento General de Protección de Datos (GDPR) de la UE o regulaciones como el Estándar de Seguridad de Datos PCI (PCI DSS). Para todos estos datos:

- ¿Se utilizan algoritmos o protocolos criptográficos antiguos o débiles de forma predeterminada o en código más antiguo?
- ¿Se utilizan claves criptográficas predeterminadas, se generan claves criptográficas débiles, se reutilizan claves o falta una gestión y rotación de claves adecuadas?
- ¿Las claves criptográficas se registran en los repositorios de código fuente?
- ¿No se aplica el cifrado, por ejemplo, faltan directivas o encabezados de seguridad de encabezados HTTP (navegador)?
- ¿El certificado de servidor recibido y la cadena de confianza están validados correctamente?
- ¿Los vectores de inicialización ignorados, reutilizados o no generados son lo suficientemente seguros para el modo de operación criptográfico? ¿Se utiliza un modo de funcionamiento inseguro como el ECB (Electronic Codebook)? ¿Se utiliza cifrado cuando el cifrado autenticado es más apropiado?
- ¿Se utilizan contraseñas como claves criptográficas en ausencia de una función de derivación de claves basada en contraseñas?
- ¿Se utiliza la aleatoriedad con fines criptográficos, pero que no fueron diseñados para cumplir con los requisitos criptográficos? Incluso si se elige la función correcta, ¿el desarrollador debe sembrarla y/o de lo contrario, el desarrollador ha sobrescrito la funcionalidad original de “strong seeding” con una semilla que carece de suficiente entropía/imprevisibilidad?
- ¿Se utilizan funciones hash obsoletas como MD5 o SHA1, o se utilizan funciones hash no criptográficas cuando se necesitan funciones hash criptográficas?
- ¿Se utilizan métodos de relleno criptográfico obsoletos como PKCS número 1 v1.5?

- ¿Son explotables los mensajes de error criptográficos o la información del canal lateral, por ejemplo en forma de ataques de **padding oracle attack** ?
- ¿Se puede degradar u omitir el algoritmo criptográfico?

Ver referencias ASVS: Criptografía (V11), Comunicación Segura (V12) y Protección de Datos (V14).

Cómo prevenir.

Haga lo siguiente, como mínimo, y consulte las referencias:

- Clasificar y etiquetar datos procesados, almacenados o transmitidos por una aplicación. Identifique qué datos son confidenciales según las leyes de privacidad, los requisitos reglamentarios o las necesidades comerciales.
- Almacene sus claves más confidenciales en módulos de seguridad de hardware (HSM) basado en hardware o en la nube.
- Utilice implementaciones confiables de algoritmos criptográficos siempre que sea posible.
- No almacene datos confidenciales innecesariamente. Deséchelo lo antes posible o utilice tokenización compatible con PCI DSS o incluso truncamiento. Los datos que no se conservan no pueden ser robados.
- Asegúrese de cifrar todos los datos confidenciales en reposo.
- Asegúrese de que existan algoritmos, protocolos y claves estándar actualizados y sólidos; utilice una gestión de claves adecuada.
- Cifre todos los datos en tránsito con protocolos seguros como TLS con cifrados de secreto directo (FS), priorización de cifrado por parte del servidor y parámetros seguros. Imponer el cifrado mediante directivas como HTTP Strict Transport Security (HSTS).
- Deshabilite el almacenamiento en caché para respuestas que contengan datos confidenciales. Esto incluye el almacenamiento en caché en su servidor web de CDN y cualquier almacenamiento en caché de aplicaciones (por ejemplo: Redis).
- Aplicar los controles de seguridad requeridos según la clasificación de datos.
- No utilice protocolos no cifrados como FTP y SMTP.
- Almacene contraseñas utilizando fuertes funciones hash adaptativas y saladas con un factor de trabajo (factor de retardo), como Argon2, scrypt, bcrypt (sistemas heredados) o PBKDF2-HMAC-SHA-256. Para sistemas heredados que utilizan bcrypt, obtenga más consejos en OWASP-Cheatsheet(Hoja de trucos de): almacenamiento de contraseñas
- Los vectores de inicialización deben elegirse de forma apropiada para el modo de operación. Esto podría significar utilizar un CSPRNG (generador de

números pseudoaleatorios criptográficamente seguro). Para los modos que requieren un nonce, el vector de inicialización (IV) no necesita un CSPRNG. En todos los casos, el IV nunca debe usarse dos veces para una clave fija.
- Utilice siempre cifrado autenticado en lugar de sólo cifrado.
- Las claves deben generarse criptográficamente de forma aleatoria y almacenarse en la memoria como matrices de bytes. Si se utiliza una contraseña, ésta debe convertirse en una clave mediante una función de derivación de clave base de contraseña adecuada.
- Asegúrese de que se utilice aleatoriedad criptográfica cuando sea apropiado y que no haya sido sembrada de manera predecible o con baja entropía. La mayoría de las API modernas no requieren que el desarrollador inicie el CSPRNG para que sea seguro.
- Evite funciones criptográficas obsoletas, métodos de construcción de bloques y esquemas de relleno (block building methods and padding schemes), como MD5, SHA1, modo de encadenamiento de bloques de cifrado (CBC), PKCS número 1 v1.5.
- Asegúrese de que las configuraciones cumplan con los requisitos de seguridad haciéndolas revisar por especialistas en seguridad, herramientas diseñadas para este propósito o ambos.
- Considere prepararse para la criptografía poscuántica (PQC), consulte las referencias (ENISA, NIST)

Ejemplos de escenarios de ataque.

Escenario #1: Un sitio no utiliza ni aplica TLS para todas las páginas o admite un cifrado débil. Un atacante monitorea el tráfico de la red (por ejemplo, en una red inalámbrica insegura), degrada las conexiones de HTTPS a HTTP, intercepta solicitudes y roba la cookie de sesión del usuario. Luego, el atacante reproduce esta cookie y secuestra la sesión (autenticada) del usuario, accediendo o modificando los datos privados del usuario. En lugar de lo anterior, podrían alterar todos los datos transportados, por ejemplo, el destinatario de una transferencia de dinero.

Escenario #2: La base de datos de contraseñas utiliza hashes simples o sin sal para almacenar las contraseñas de todos. Una falla en la carga de archivos permite a un atacante recuperar la base de datos de contraseñas. Todos los hashes sin sal se pueden exponer con una tabla arcoíris de hashes precalculados. Las GPU pueden descifrar los hashes generados por funciones hash simples o rápidas, incluso si estuvieran salados.

Referencias.

- OWASP Proactive Controls: C2: Use Cryptography to Protect Data
- OWASP Application Security Verification Standard (ASVS): V11, 12, 14
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography
- ENISA: A Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography
- NIST Releases First 3 Finalized Post-Quantum Encryption Standards

List of Mapped CWEs

- CWE-261 Weak Encoding for Password
- CWE-296 Improper Following of a Certificate’s Chain of Trust
- CWE-319 Cleartext Transmission of Sensitive Information
- CWE-320 Key Management Errors (Prohibited)
- CWE-321 Use of Hard-coded Cryptographic Key
- CWE-322 Key Exchange without Entity Authentication
- CWE-323 Reusing a Nonce, Key Pair in Encryption
- CWE-324 Use of a Key Past its Expiration Date
- CWE-325 Missing Required Cryptographic Step
- CWE-326 Inadequate Encryption Strength
- CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- CWE-328 Reversible One-Way Hash
- CWE-329 Not Using a Random IV with CBC Mode
- CWE-330 Use of Insufficiently Random Values

- CWE-331 Insufficient Entropy
- CWE-332 Insufficient Entropy in PRNG
- CWE-334 Small Space of Random Values
- CWE-335 Incorrect Usage of Seeds in Pseudo-Random Number Generator(PRNG)
- CWE-336 Same Seed in Pseudo-Random Number Generator (PRNG)
- CWE-337 Predictable Seed in Pseudo-Random Number Generator (PRNG)
- CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator(PRNG)
- CWE-340 Generation of Predictable Numbers or Identifiers
- CWE-342 Predictable Exact Value from Previous Values
- CWE-347 Improper Verification of Cryptographic Signature
- CWE-523 Unprotected Transport of Credentials
- CWE-757 Selection of Less-Secure Algorithm During Negotiation(‘Algorithm Downgrade’)
- CWE-759 Use of a One-Way Hash without a Salt
- CWE-760 Use of a One-Way Hash with a Predictable Salt
- CWE-780 Use of RSA Algorithm without OAEP
- CWE-916 Use of Password Hash With Insufficient Computational Effort
- CWE-1240 Use of a Cryptographic Primitive with a Risky Implementation
- CWE-1241 Use of Predictable Algorithm in Random Number Generator

OwaspTop10-2025-A05-Inyección-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A05:2025 Inyección

Contexto.

La inyección cae dos puestos del puesto número 3 al número 5 en el ranking, manteniendo su posición en relación con A04:2025-Fallos criptográficos y A06:2025-Diseño inseguro. La inyección es una de las categorías más probadas y el 100% de las aplicaciones se prueban para algún tipo de inyección. Tuvo el mayor número de CVE para cualquier categoría, con 37 CWE en esta categoría. La inyección incluye secuencias de comandos entre sitios (alta frecuencia/bajo impacto) con más de 30 000 CVE e inyección SQL (baja frecuencia/alto impacto) con más de 14 000 CVE. La enorme cantidad de CVE informados para CWE-79 Neutralización inadecuada de la entrada durante la generación de páginas web (‘Cross-site Scripting’) reduce el impacto ponderado promedio de esta categoría.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
37	13,77%	3,08%	100,00%	42,93%	7.15

Descripción.

Una vulnerabilidad de inyección es una falla del sistema que permite a un atacante insertar código o comandos maliciosos (como SQL o código de shell) en los campos de entrada de un programa, engañando al sistema para que ejecute el

código o los comandos como si fuera parte del sistema. Esto puede tener consecuencias verdaderamente nefastas.

Una aplicación es vulnerable a ataques cuando:

- La aplicación no valida, filtra ni desinfecta los datos proporcionados por el usuario.
- Las consultas dinámicas o las llamadas no parametrizadas sin escape consciente del contexto se utilizan directamente en el intérprete.
- Los datos no desinfectados se utilizan dentro de los parámetros de búsqueda de mapeo relacional de objetos (ORM) para extraer registros confidenciales adicionales.
- Los datos potencialmente hostiles se utilizan o concatenan directamente. El SQL o comando contiene la estructura y los datos maliciosos en consultas dinámicas, comandos o procedimientos almacenados.

Algunas de las inyecciones más comunes son SQL, NoSQL, comando del sistema operativo, mapeo relacional de objetos (ORM), LDAP e inyección de lenguaje de expresión (EL) o biblioteca de navegación de gráficos de objetos (OGNL). El concepto es idéntico entre todos los intérpretes. La detección se logra mejor mediante una combinación de revisión del código fuente junto con pruebas automatizadas (incluido el fuzzing) de todos los parámetros, encabezados, URL, cookies, JSON, SOAP y entradas de datos XML. La adición de herramientas de prueba de seguridad de aplicaciones estáticas (SAST), dinámicas (DAST) e interactivas (IAST) al flujo de CI/CD también puede ser útil para identificar fallas de inyección antes de la implementación de producción.

Una clase relacionada de vulnerabilidades de inyección se ha vuelto común en los LLM. Estas se analizan por separado en el Top 10 de OWASP LLM, específicamente LLM01:2025 Inyección inmediata.

Cómo prevenir.

La mejor manera de evitar la inyección requiere mantener los datos separados de los comandos y consultas:

- La opción preferida es utilizar una API segura, que evite por completo el uso del intérprete, proporcione una interfaz parametrizada o migre a Object Relational Mapping Tools (ORM). **Nota:** Incluso cuando están parametrizados, los procedimientos almacenados aún pueden introducir inyección SQL si PL/SQL o T-SQL concatenan consultas y datos o ejecutan datos hostiles con EXECUTE IMMEDIATE o exec().

Cuando no es posible separar los datos de los comandos, puede reducir las amenazas utilizando las siguientes técnicas.

- Utilice una validación de entrada positiva del lado del servidor. Esta no es una defensa completa ya que muchas aplicaciones requieren caracteres especiales, como áreas de texto o API para aplicaciones móviles.
- Para cualquier consulta dinámica residual, escape caracteres especiales utilizando la sintaxis de escape específica para ese intérprete. **Nota:** No se puede escapar de estructuras SQL como nombres de tablas, nombres de columnas, etc. y, por lo tanto, los nombres de estructuras proporcionados por el usuario son peligrosos. Este es un problema común en el software de redacción de informes.

Advertencia Estas técnicas implican analizar y escapar cadenas complejas, lo que las hace propensas a errores y no robustas frente a cambios menores en el sistema subyacente.

Ejemplos de escenarios de ataque.

Escenario #1: Una aplicación utiliza datos no confiables en la construcción de la siguiente llamada SQL vulnerable:

```
String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Escenario #2: De manera similar, la confianza ciega de una aplicación en los marcos puede generar consultas que aún son vulnerables (por ejemplo, Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getPara
```

En ambos casos, el atacante modifica el valor del parámetro ‘id’ en su navegador para enviar: ’ UNION SLEEP(10);–. Por ejemplo:

http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--

Esto cambia el significado de ambas consultas para devolver todos los registros de la tabla de cuentas. Ataques más peligrosos podrían modificar o eliminar datos o incluso invocar procedimientos almacenados.

Referencias.

- OWASP Proactive Controls: Secure Database Access
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, and ORM Injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014
- PortSwigger: Server-side template injection
- Awesome Fuzzing: a list of fuzzing resources

List of Mapped CWEs

- CWE-20 Improper Input Validation
- CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component (‘Injection’)
- CWE-76 Improper Neutralization of Equivalent Special Elements
- CWE-77 Improper Neutralization of Special Elements used in a Command (‘Command Injection’)
- CWE-78 Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)
- CWE-79 Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)
- CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- CWE-83 Improper Neutralization of Script in Attributes in a Web Page
- CWE-86 Improper Neutralization of Invalid Characters in Identifiers in Web Pages
- CWE-88 Improper Neutralization of Argument Delimiters in a Command (‘Argument Injection’)
- CWE-89 Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)

- CWE-90 Improper Neutralization of Special Elements used in an LDAP Query (‘LDAP Injection’)
- CWE-91 XML Injection (aka Blind XPath Injection)
- CWE-93 Improper Neutralization of CRLF Sequences (‘CRLF Injection’)
- CWE-94 Improper Control of Generation of Code (‘Code Injection’)
- CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code (‘Eval Injection’)
- CWE-96 Improper Neutralization of Directives in Statically Saved Code (‘Static Code Injection’)
- CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
- CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program (‘PHP Remote File Inclusion’)
- CWE-99 Improper Control of Resource Identifiers (‘Resource Injection’)
- CWE-103 Struts: Incomplete validate() Method Definition
- CWE-104 Struts: Form Bean Does Not Extend Validation Class
- CWE-112 Missing XML Validation
- CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers (‘HTTP Response Splitting’)
- CWE-114 Process Control
- CWE-115 Misinterpretation of Output
- CWE-116 Improper Encoding or Escaping of Output
- CWE-129 Improper Validation of Array Index
- CWE-159 Improper Handling of Invalid Use of Special Elements
- CWE-470 Use of Externally-Controlled Input to Select Classes or Code (‘Unsafe Reflection’)
- CWE-493 Critical Public Variable Without Final Modifier

- CWE-500 Public Static Field Not Marked Final
- CWE-564 SQL Injection: Hibernate
- CWE-610 Externally Controlled Reference to a Resource in Another Sphere
- CWE-643 Improper Neutralization of Data within XPath Expressions (‘XPath Injection’)
- CWE-644 Improper Neutralization of HTTP Headers for Scripting Syntax
- CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement (‘Expression Language Injection’)

OwaspTop10-2025-A06-Diseño-inseguro-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A06:2025 Diseño inseguro

Contexto.

Insecure Design se desliza dos puestos del puesto 4 al 6 en el ranking como **A02:2025-Configuración incorrecta de seguridad** y **A03:2025-Fallas en la cadena de suministro de software** saltarlo. Esta categoría se introdujo en 2021 y hemos visto mejoras notables en la industria relacionadas con el modelado de amenazas y un mayor énfasis en el diseño seguro. Esta categoría se centra en los riesgos relacionados con el diseño y las fallas arquitectónicas, con un llamado a un mayor uso del modelado de amenazas, patrones de diseño seguros y arquitecturas de referencia. Esto incluye fallas en la lógica empresarial de una aplicación, por ejemplo, la falta de definición de cambios de estado no deseados o inesperados dentro de una aplicación. Como comunidad, necesitamos ir más allá del “desplazamiento a la izquierda” en el espacio de codificación, hacia actividades previas a la codificación, como la redacción de requisitos y el diseño de aplicaciones, que son fundamentales para los principios de Secure by Design (por ejemplo, ver **Establecer un programa AppSec moderno: fase de planificación y diseño**). Las enumeraciones de debilidades comunes notables (CWE) incluyen: *CWE-256: Almacenamiento desprotegido de credenciales, CWE-269 Gestión inadecuada de privilegios, CWE-434 Carga ilimitada de archivos con tipo peligroso, CWE-501: Violación de límites de confianza y CWE-522: Credenciales insuficientemente protegidas.*

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
39	22,18%	1,86%	88,76%	35,18%	6,96

Descripción.

El diseño inseguro es una categoría amplia que representa diferentes debilidades, expresadas como “diseño de control faltante o ineficaz” El diseño inseguro no es la fuente de todas las demás diez categorías de riesgo principales. Tenga en cuenta que existe una diferencia entre diseño inseguro e implementación insegura. Diferenciamos entre fallas de diseño y defectos de implementación por una razón: tienen diferentes causas fundamentales, ocurren en diferentes momentos del proceso de desarrollo y tienen diferentes remediaciones. Un diseño seguro aún puede tener defectos de implementación que generen vulnerabilidades que pueden ser explotadas. 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.Uno de los factores que contribuye al diseño inseguro es la falta de perfiles de riesgo empresarial inherentes al software o sistema que se está desarrollando y, por tanto, la imposibilidad de determinar qué nivel de diseño de seguridad se requiere.

Tres partes clave para tener un diseño seguro son:

- Recopilación de requisitos y gestión de recursos
- Creando un diseño seguro
- Tener un ciclo de vida de desarrollo seguro

Requisitos y gestión de recursos

Recopilar y negociar los requisitos comerciales para una aplicación con la empresa, incluidos los requisitos de protección relacionados con la confidencialidad, integridad, disponibilidad y autenticidad de todos los activos de datos y la lógica comercial esperada. Tenga en cuenta qué tan expuesta estará su aplicación y si necesita segregación de inquilinos (más allá de los necesarios para el control de acceso). Recopilar los requisitos técnicos, incluidos los requisitos de seguridad funcionales y no funcionales. Planificar y negociar el presupuesto que cubra todo el diseño, construcción, pruebas y operación, incluidas las actividades

de seguridad.

Diseño seguro

El diseño seguro es una cultura y metodología que evalúa constantemente las amenazas y garantiza que el código esté diseñado y probado de manera sólida para prevenir métodos de ataque conocidos. El modelado de amenazas debe integrarse en sesiones de refinamiento (o actividades similares); busque cambios en los flujos de datos y en el control de acceso u otros controles de seguridad. En el desarrollo de la historia del usuario, determine el flujo correcto y los estados de falla, asegúrese de que sean bien comprendidos y acordados por las partes responsables e afectadas. Analizar los supuestos y condiciones de los flujos esperados y de falla para garantizar que sigan siendo precisos y deseables. Determinar cómo validar los supuestos y hacer cumplir las condiciones necesarias para un comportamiento adecuado. Asegúrese de que los resultados estén documentados en la historia del usuario. Aprenda de los errores y ofrezca incentivos positivos para promover mejoras.El diseño seguro no es un complemento ni una herramienta que puedas agregar al software.

Ciclo de vida del desarrollo seguro

El software seguro requiere un ciclo de vida de desarrollo seguro, un patrón de diseño seguro, una metodología de carreteras pavimentadas, una biblioteca de componentes segura, herramientas apropiadas, modelos de amenazas y autopsias de incidentes que se utilizan para mejorar el proceso. Comuníquese con sus especialistas en seguridad al comienzo de un proyecto de software, durante todo el proyecto y para realizar un mantenimiento continuo del software. Considere aprovechar el Modelo de madurez de garantía de software OWASP (SAMM) para ayudarle a estructurar sus esfuerzos de desarrollo de software seguro.

Cómo prevenir.

- Establecer y utilizar un ciclo de vida de desarrollo seguro con profesionales de AppSec para ayudar a evaluar y diseñar controles relacionados con la seguridad y la privacidad
- Establecer y utilizar una biblioteca de patrones de diseño seguros o componentes de carreteras pavimentadas
- Utilice modelos de amenazas para partes críticas de la aplicación, como autenticación, control de acceso, lógica empresarial y flujos de claves

- Integre lenguaje y controles de seguridad en las historias de usuario
- Integre comprobaciones de plausibilidad en cada nivel de su aplicación (desde el frontend hasta el backend)
- Escriba pruebas unitarias y de integración para validar que todos los flujos críticos sean resistentes al modelo de amenaza. Compilar casos de uso *y* casos de mal uso para cada nivel de su aplicación.
- Segregar capas de niveles en las capas del sistema y de la red, según las necesidades de exposición y protección
- Segregar a los inquilinos de manera sólida por diseño en todos los niveles

Ejemplos de escenarios de ataque.

Escenario #1: Un flujo de trabajo de recuperación de credenciales puede incluir "preguntas y respuestas", lo cual está prohibido por NIST 800-63b, OWASP ASVS y OWASP Top 10. No se puede confiar en las preguntas y respuestas como evidencia de identidad, ya que más de una persona puede conocer las respuestas. Esta funcionalidad debería eliminarse y reemplazarse por un diseño más seguro.

Escenario #2: Una cadena de cines permite descuentos en reservas grupales y cuenta con un máximo de quince asistentes antes de requerir un depósito. Los atacantes podrían modelar este flujo y probar si pueden encontrar un vector de ataque en la lógica empresarial de la aplicación, por ejemplo, reservando seiscientas plazas y todos los cines a la vez en unas pocas solicitudes, lo que provocaría una pérdida masiva de ingresos.

Escenario #3: El sitio web de comercio electrónico de una cadena minorista no tiene protección contra bots administrados por revendedores que compran tarjetas de video de alta gama para revenderlas en sitios web de subastas. Esto crea una publicidad terrible para los fabricantes de tarjetas de vídeo y los propietarios de cadenas minoristas, y soporta la mala sangre de los entusiastas que no pueden obtener estas tarjetas a ningún precio. Un diseño anti-bot cuidadoso y reglas lógicas de dominio, como compras realizadas dentro de unos segundos de disponibilidad, podrían identificar compras no auténticas y rechazar dichas transacciones.

Referencias.

- OWASP Cheat Sheet: Secure Design Principles
- OWASP SAMM: Design | Secure Architecture

- OWASP SAMM: Design | Threat Assessment
- NIST – Guidelines on Minimum Standards for Developer Verification of Software
- The Threat Modeling Manifesto
- Awesome Threat Modeling

List of Mapped CWEs

- CWE-73 External Control of File Name or Path
- CWE-183 Permissive List of Allowed Inputs
- CWE-256 Unprotected Storage of Credentials
- CWE-266 Incorrect Privilege Assignment
- CWE-269 Improper Privilege Management
- CWE-286 Incorrect User Management
- CWE-311 Missing Encryption of Sensitive Data
- CWE-312 Cleartext Storage of Sensitive Information
- CWE-313 Cleartext Storage in a File or on Disk
- CWE-316 Cleartext Storage of Sensitive Information in Memory
- CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’)
- CWE-382 J2EE Bad Practices: Use of System.exit()
- CWE-419 Unprotected Primary Channel
- CWE-434 Unrestricted Upload of File with Dangerous Type
- CWE-436 Interpretation Conflict
- CWE-444 Inconsistent Interpretation of HTTP Requests (‘HTTP Request Smuggling’)
- CWE-451 User Interface (UI) Misrepresentation of Critical Information
- CWE-454 External Initialization of Trusted Variables or Data Stores

- CWE-472 External Control of Assumed-Immutable Web Parameter
- CWE-501 Trust Boundary Violation
- CWE-522 Insufficiently Protected Credentials
- CWE-525 Use of Web Browser Cache Containing Sensitive Information
- CWE-539 Use of Persistent Cookies Containing Sensitive Information
- CWE-598 Use of GET Request Method With Sensitive Query Strings
- CWE-602 Client-Side Enforcement of Server-Side Security
- CWE-628 Function Call with Incorrectly Specified Arguments
- CWE-642 External Control of Critical State Data
- CWE-646 Reliance on File Name or Extension of Externally-Supplied File
- CWE-653 Insufficient Compartmentalization
- CWE-656 Reliance on Security Through Obscurity
- CWE-657 Violation of Secure Design Principles
- CWE-676 Use of Potentially Dangerous Function
- CWE-693 Protection Mechanism Failure
- CWE-799 Improper Control of Interaction Frequency
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- CWE-841 Improper Enforcement of Behavioral Workflow
- CWE-1021 Improper Restriction of Rendered UI Layers or Frames
- CWE-1022 Use of Web Link to Untrusted Target with window.opener Access
- CWE-1125 Excessive Attack Surface

OwaspTop10-2025-A07-Fallos-de-autenticación-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A07:2025 Fallos de autenticación

Contexto.

Authentication Failures mantiene su posición en el puesto número 7 con un ligero cambio de nombre para reflejar con mayor precisión los 36 CWE de esta categoría. A pesar de los beneficios de los marcos estandarizados, esta categoría ha mantenido su puesto número 7 desde 2021. Los CWE notables incluidos son *CWE-259 Uso de contraseña codificada*, *CWE-297: Validación incorrecta del certificado con desajuste del host*, *CWE-287: Autenticación incorrecta*, *CWE-384: Fijación de sesión*, y *CWE-798 Uso de credenciales codificadas*.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
36	15,80%	2,92%	100,00%	37,14%	7,69

Descripción.

Cuando un atacante puede engañar a un sistema para que reconozca como legítimo a un usuario no válido o incorrecto, esta vulnerabilidad está presente. Puede haber debilidades de autenticación si la aplicación:

- Permite ataques automatizados como el relleno de credenciales (credential stuffing), donde el atacante tiene una “breached list” de nombres de usuario y contraseñas válidos. Más recientemente, este tipo de ataque se ha ampliado para incluir ataques híbridos de contraseñas y relleno de credenciales (hybrid credential stuffing attack), también conocidos como ataques de “password spray attacks”, donde el atacante utiliza variaciones o incrementos de credenciales filtradas (spilled credentials) para obtener acceso, por ejemplo, probando Password1!, Password2!, Password3! y así sucesivamente.

- Permite la fuerza bruta u otros ataques automatizados y programados que no se bloquean rápidamente.

- Permite contraseñas predeterminadas, débiles o bien conocidas, como “Contraseña1” o nombre de usuario “admin” con una contraseña “admin”.

- Permite a los usuarios crear nuevas cuentas con credenciales violadas ya conocidas.

- Permite el uso de procesos de recuperación de credenciales débiles o ineficaces y de contraseñas olvidadas, como “respuestas basadas en el conocimiento”, que no pueden hacerse seguros.

- Utiliza almacenes de datos de contraseñas de texto plano, o débilmente cifradas y/o hash débil (ver A02:2021-Fallos criptográficos).

- Tiene autenticación multifactor (MFA) faltante o ineficaz.

- Permite el uso de alternativas débiles o ineficaces si no está disponible la autenticación multifactor (MFA).

- Expone un identificador de sesión en la URL, un campo oculto u otra ubicación insegura a la que pueda acceder el cliente.

- Reutiliza el mismo identificador de sesión después de iniciar sesión correctamente.

- No invalida correctamente las sesiones de usuario ni los tokens de autenticación (principalmente tokens de inicio de sesión único (SSO)) durante el cierre de sesión o un período de inactividad.

Cómo prevenir.

- Siempre que sea posible, implemente y haga cumplir el uso de autenticación multifactor (MFA) para evitar ataques automatizados de relleno de credenciales, fuerza bruta y reutilización de credenciales robadas.

- Siempre que sea posible, fomente y habilite el uso de administradores de contraseñas para ayudar a los usuarios a tomar mejores decisiones.

- No envíe ni implemente con credenciales predeterminadas, especialmente para usuarios administradores.

- Implemente comprobaciones de contraseñas débiles, comparando las contraseñas nuevas o modificadas con la lista de las 10000 peores contraseñas.

- Durante la creación de una nueva cuenta y los cambios de contraseña, valide con listas de credenciales violadas conocidas (por ejemplo: usando haveibeenpwned.com).

- Alinee las políticas de longitud, complejidad y rotación de contraseñas con Directrices del Instituto Nacional de Estándares y Tecnología (NIST) 800-63b en la sección 5.1.1 para secretos memorizados u otras políticas de contraseñas modernas basadas en evidencia.

- No obligue a seres humanos a rotar contraseñas a menos que sospeche de una violación. Si sospecha una violación, fuerce el restablecimiento de la contraseña inmediatamente.

- Asegúrese de que las vías de registro, recuperación de credenciales y API estén reforzadas contra ataques de enumeración de cuentas mediante el uso de los mismos mensajes para todos los resultados (“Nombre de usuario o contraseña no válidos”).

- Limite o retrase cada vez más los intentos fallidos de inicio de sesión, pero tenga cuidado de no crear un escenario de denegación de servicio. Registre todas las fallas y alerte a los administradores cuando se detecten o sospechen manipulación de credenciales, fuerza bruta u otros ataques.

- Utilice un administrador de sesiones integrado (session manager), seguro y del lado del servidor que genere un nuevo ID de sesión aleatorio con alta

entropía después de iniciar sesión. Los identificadores de sesión no deben estar en la URL, almacenarse de forma segura en una cookie segura e invalidarse después de cerrar sesión, estar inactivos y tener tiempos de espera absolutos.

- Lo ideal es utilizar un sistema “pre-built system” confiable para gestionar la autenticación, la identidad y la gestión de sesiones. Transfiera este riesgo siempre que sea posible comprando y utilizando un sistema reforzado y bien probado.

Ejemplos de escenarios de ataque.

Escenario #1: El relleno de credenciales, el uso de listas de combinaciones conocidas de nombres de usuario y contraseñas, es ahora un ataque muy común. Más recientemente, se ha descubierto que los atacantes ‘incrementan’ o ajustan de otro modo las contraseñas, basándose en el comportamiento humano común. Por ejemplo, cambiar ‘Winter2025’ a ‘Winter2026’, o ‘ILoveMyDog6’ a ‘ILoveMyDog7’ o ‘ILoveMyDog5’. Este ajuste de los intentos de contraseña se denomina ataque híbrido de relleno de credenciales o ataque de pulverización de contraseñas, y puede ser incluso más efectivo que la versión tradicional. Si una aplicación no implementa defensas contra amenazas automatizadas (fuerza bruta, scripts o bots) o relleno de credenciales, la aplicación se puede utilizar como un oráculo de contraseñas para determinar si las credenciales son válidas y obtener acceso no autorizado.

Escenario #2: La mayoría de los ataques de autenticación exitosos ocurren debido al uso continuo de contraseñas como único factor de autenticación. Una vez consideradas las mejores prácticas, la rotación de contraseñas y los requisitos de complejidad alientan a los usuarios tanto a reutilizar contraseñas como a utilizar contraseñas débiles. Se recomienda a las organizaciones que pongan fin a estas prácticas según NIST 800-63 y que impongan el uso de autenticación multifactor (MFA) en todos los sistemas importantes.

Escenario #3: Los tiempos de espera de las sesiones de la aplicación no se implementan correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación y en lugar de seleccionar “cerrar sesión”, simplemente cierra la pestaña del navegador y se aleja. Otro ejemplo de esto es si una sesión de inicio de sesión único (Single Sign-On “SSO”) no se puede cerrar mediante un cierre de sesión único (Single Logout “SLO”). Es decir, un único inicio de sesión le permite acceder, por ejemplo, a su lector de correo, a su sistema de documentos y

a su sistema de chat. Pero el cierre de sesión solo ocurre en el sistema actual. Si un atacante utiliza el mismo navegador después de que la víctima cree que ha cerrado sesión correctamente, pero con el usuario aún autenticado en algunas de las aplicaciones, puede acceder a la cuenta de la víctima.El mismo problema puede ocurrir en oficinas y empresas cuando no se ha salido correctamente de una aplicación confidencial y un colega tiene acceso (temporal) a la computadora desbloqueada.

Referencias.

- OWASP Authentication Cheat Sheet
- OWASP Secure Coding Practices

List of Mapped CWEs

- CWE-258 Empty Password in Configuration File
- CWE-259 Use of Hard-coded Password
- CWE-287 Improper Authentication
- CWE-288 Authentication Bypass Using an Alternate Path or Channel
- CWE-289 Authentication Bypass by Alternate Name
- CWE-290 Authentication Bypass by Spoofing
- CWE-291 Reliance on IP Address for Authentication
- CWE-293 Using Referer Field for Authentication
- CWE-294 Authentication Bypass by Capture-replay
- CWE-295 Improper Certificate Validation
- CWE-297 Improper Validation of Certificate with Host Mismatch
- CWE-298 Improper Validation of Certificate with Host Mismatch
- CWE-299 Improper Validation of Certificate with Host Mismatch
- CWE-300 Channel Accessible by Non-Endpoint

- CWE-302 Authentication Bypass by Assumed-Immutable Data
- CWE-303 Incorrect Implementation of Authentication Algorithm
- CWE-304 Missing Critical Step in Authentication
- CWE-305 Authentication Bypass by Primary Weakness
- CWE-306 Missing Authentication for Critical Function
- CWE-307 Improper Restriction of Excessive Authentication Attempts
- CWE-308 Use of Single-factor Authentication
- CWE-309 Use of Password System for Primary Authentication
- CWE-346 Origin Validation Error
- CWE-350 Reliance on Reverse DNS Resolution for a Security-Critical Action
- CWE-384 Session Fixation
- CWE-521 Weak Password Requirements
- CWE-613 Insufficient Session Expiration
- CWE-620 Unverified Password Change
- CWE-640 Weak Password Recovery Mechanism for Forgotten Password
- CWE-798 Use of Hard-coded Credentials
- CWE-940 Improper Verification of Source of a Communication Channel
- CWE-941 Incorrectly Specified Destination in a Communication Channel
- CWE-1390 Weak Authentication
- CWE-1391 Use of Weak Credentials
- CWE-1392 Use of Default Credentials
- CWE-1393 Use of Default Password

OwaspTop10-2025-A08-Fallos-de-integridad-de-software-o-datos-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A08:2025 Fallas de integridad de software o datos

Contexto.

Las fallas de integridad de software o datos continúan en el puesto n.° 8, con un ligero y esclarecedor cambio de nombre de “Software” *y* Fallas de integridad de datos”. Esta categoría se centra en la falta de mantenimiento de los límites de confianza y de verificación de la integridad del software, el código y los artefactos de datos a un nivel inferior al de las fallas de la cadena de suministro de software. Esta categoría se centra en hacer suposiciones relacionadas con actualizaciones de software y datos críticos, sin verificar la integridad. Las enumeraciones de debilidades comunes notables (CWE) incluyen: *CWE-829: Inclusión de funcionalidades de una esfera de control no confiable*, *CWE-915: Modificación mal controlada de atributos de objetos determinados dinámicamente*, y *CWE-502: Deserialización de datos no confiables*.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
14	8,98%	2,75%	78,52%	45,49%	7.11

Descripción.

Las fallas de integridad del software y de los datos se relacionan con código e infraestructura que no protegen contra código o datos no válidos o no confiables que se tratan como confiables y válidos. Un ejemplo de esto es cuando una aplicación depende de complementos, bibliotecas o módulos de fuentes, repositorios y redes de distribución de contenido (CDN) que no son de confianza.

Una pipeline de CI/CD insegura sin consumir ni proporcionar comprobaciones de integración de software puede generar la posibilidad de acceso no autorizado, código inseguro o malicioso o compromiso del sistema. Otro ejemplo de esto es un CI/CD que extrae código o artefactos de lugares que no son de confianza y/o no los verifica antes de su uso (verificando la firma o un mecanismo similar).

Por último, muchas aplicaciones ahora incluyen la funcionalidad de actualización automática,donde las actualizaciones se descargan sin suficiente verificación de integridad y se agregan al aplicativo, que era previamente confiable. Los atacantes podrían potencialmente cargar sus propias actualizaciones para distribuirlas y ejecutarlas en todas las instalaciones. Otro ejemplo es cuando los objetos o datos se codifican o serializan en una estructura que un atacante puede ver y modificar y es vulnerable a una deserialización insegura.

Cómo prevenir.

- Utilice firmas digitales o mecanismos similares para verificar que el software o los datos provengan de la fuente esperada y no hayan sido alterados.

- Asegúrese de que las bibliotecas y dependencias, como **npm** (Node Package Manager) o Maven, solo consuman repositorios confiables. Si tiene un perfil de riesgo más alto, considere alojar un repositorio interno que se sepa que es bueno y que esté examinado.

- Asegúrese de que exista un proceso de revisión de los cambios de código y configuración para minimizar la posibilidad de que se introduzca código o configuración maliciosa en su pipeline de software.

- Asegúrese de que su pipeline de CI/CD tenga segregación, configuración y control de acceso adecuados para garantizar la integridad del código que fluye a través de los procesos de compilación e implementación.

- Asegúrese de que los datos serializados no firmados o no cifrados no se reciban de clientes no confiables y posteriormente se utilicen sin algún tipo de verificación de integridad o firma digital para detectar manipulación o

reproducción de los datos serializados.

Ejemplos de escenarios de ataque.

Escenario #1: Inclusión de funcionalidad web de una fuente no confiable Una empresa utiliza un proveedor de servicios externo para proporcionar funcionalidad de soporte. Para mayor comodidad, tiene un mapeo DNS para myCompany.SupportProvider.com to support.myCompany.com. Esto significa que todas las cookies, incluidas las cookies de autenticación, están configuradas en el myCompany.com El dominio ahora se enviará al proveedor de soporte. Cualquier persona con acceso a la infraestructura del proveedor de soporte puede robar las cookies de todos sus usuarios que las hayan visitado support.myCompany.com y realizar un ataque de secuestro de sesión.

Escenario #2: Actualización sin firmar. Muchos enrutadores domésticos, decodificadores, firmware de dispositivos y otros no verifican las actualizaciones mediante firmware firmado. El firmware no firmado es un objetivo cada vez mayor para los atacantes y se espera que solo empeore. Esta es una preocupación importante ya que muchas veces no existe ningún mecanismo para remediarla más allá de arreglarla en una versión futura y esperar a que las versiones anteriores envejezcan.

Escenario #3 Un desarrollador tiene problemas para encontrar la versión actualizada de un paquete que está buscando, por lo que no la descarga desde el administrador de paquetes habitual y confiable, sino desde un sitio web en línea. El paquete no está firmado y, por tanto, no hay oportunidad de garantizar la integridad. El paquete incluye código malicioso.

Escenario #4: Deserialización insegura, Una aplicación React llama a un conjunto de microservicios Spring Boot. Al ser programadores funcionales, intentaron asegurarse de que su código fuera inmutable. La solución que se les ocurrió es serializar el estado del usuario y pasarlo de un lado a otro con cada solicitud. Un atacante detecta la firma del objeto Java “rO0” (en base64) y utiliza la Escáner de deserialización de Java para obtener la ejecución remota de código en el servidor de aplicaciones.

Referencias.

- OWASP Cheat Sheet: Software Supply Chain Security

- OWASP Cheat Sheet: Infrastructure as Code
- OWASP Cheat Sheet: Deserialization
- SAFECode Software Integrity Controls
- A ‘Worst Nightmare’ Cyberattack: The Untold Story Of The SolarWinds Hack
- CodeCov Bash Uploader Compromise
- Securing DevOps by Julien Vehent
- Insecure Deserialization by Tenendo

List of Mapped CWEs

- CWE-345 Insufficient Verification of Data Authenticity
- CWE-353 Missing Support for Integrity Check
- CWE-426 Untrusted Search Path
- CWE-427 Uncontrolled Search Path Element
- CWE-494 Download of Code Without Integrity Check
- CWE-502 Deserialization of Untrusted Data
- CWE-506 Embedded Malicious Code
- CWE-509 Replicating Malicious Code (Virus or Worm)
- CWE-565 Reliance on Cookies without Validation and Integrity Checking
- CWE-784 Reliance on Cookies without Validation and Integrity Checking in a Security Decision
- CWE-829 Inclusion of Functionality from Untrusted Control Sphere
- CWE-830 Inclusion of Web Functionality from an Untrusted Source
- CWE-915 Improperly Controlled Modification of Dynamically-Determined Object Attributes
- CWE-926 Improper Export of Android Application Components

OwaspTop10-2025-A09-Fallas-de-registro-y-alerta-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A09:2025 Fallos de registro LOG y alertas

Contexto.

Logging & Alerting Failures mantiene su posición en el puesto número 9. Esta categoría tiene un ligero cambio de nombre para enfatizar la función de alerta necesaria para inducir la acción sobre eventos de registro relevantes. Esta categoría siempre estará subrepresentada en los datos y, por tercera vez, los participantes de la encuesta comunitaria votaron a favor de una posición en la lista. Esta categoría es increíblemente difícil de probar y tiene una representación mínima en los datos CVE/CVSS (solo 723 CVE); pero puede tener un gran impacto en la visibilidad, las alertas de incidentes y la ciencia forense. Esta categoría incluye problemas con *manejar adecuadamente la codificación de salida a archivos de registro (CWE-117), insertar datos confidenciales en archivos de registro (CWE-532) y registro insuficiente (CWE-778).*

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
5	11,33%	3,91%	85,96%	46,48%	7.19

Descripción.

Sin registro y monitoreo, no se pueden detectar ataques e infracciones, y sin alertas es muy difícil responder rápida y eficazmente durante un incidente de seguridad. En cualquier momento se produce un registro insuficiente, un seguimiento continuo, una detección y una alerta para iniciar respuestas activas:

- Los eventos auditables, como inicios de sesión, inicios de sesión fallidos y transacciones de alto valor, no se registran o se registran de manera inconsistente (por ejemplo, solo se registran inicios de sesión exitosos, pero no intentos fallidos).
- Las advertencias y los errores generan mensajes de registro nulos, inadecuados o poco claros.
- La integridad de los registros no está adecuadamente protegida contra manipulaciones.
- Los registros de aplicaciones y API no se monitorean para detectar actividades sospechosas.
- Los registros solo se almacenan localmente y no se realiza una copia de seguridad adecuada.
- No existen umbrales de alerta adecuados ni procesos de escalada de respuesta que sean eficaces. Las alertas no se reciben ni se revisan dentro de un período de tiempo razonable.
- Las pruebas de penetración y los escaneos realizados mediante herramientas de pruebas de seguridad de aplicaciones dinámicas (DAST) (como Burp o ZAP) no activan alertas.
- La aplicación no puede detectar, escalar ni alertar sobre ataques activos en tiempo real o casi en tiempo real.
- Es vulnerable a la fuga de información confidencial al hacer que los eventos de registro y alerta sean visibles para un usuario o un atacante (ver A01:2025-Control de acceso roto), o registrando información confidencial que no debe registrarse (como PII o PHI).
- Es vulnerable a inyecciones o ataques a los sistemas de registro o monitoreo si los datos de registro no están codificados correctamente.
- Falta la aplicación o maneja mal los errores y otras condiciones excepcionales, de modo que el sistema no sabe que hubo un error y, por lo tanto, no puede registrar que hubo un problema.
- Faltan casos de uso ‘adecuados’ para emitir alertas o están desactualizados para reconocer una situación especial.
- Demasiadas alertas falsas positivas hacen imposible distinguir las alertas importantes de las no importantes, lo que hace que se reconozcan demasiado tarde o no se reconozcan en absoluto (sobrecarga física del equipo SOC).
- Las alertas detectadas no se pueden procesar correctamente porque el manual del caso de uso está incompleto, desactualizado o falta.

Cómo prevenir.

Los desarrolladores deben implementar algunos o todos los siguientes controles, dependiendo del riesgo de la aplicación:

- Asegúrese de que todas las fallas de inicio de sesión, control de acceso y validación de entrada del lado del servidor puedan registrarse con suficiente contexto de usuario para identificar cuentas sospechosas o maliciosas y conservarse durante el tiempo suficiente para permitir un análisis forense retrasado.
- Asegúrese de que cada parte de su aplicación que contenga un control de seguridad esté registrada, ya sea que tenga éxito o falle.
- Asegúrese de que los registros se generen en un formato que las soluciones de gestión de registros puedan consumir fácilmente.
- Asegúrese de que los datos de registro estén codificados correctamente para evitar inyecciones o ataques a los sistemas de registro o monitoreo.
- Asegúrese de que todas las transacciones tengan un registro de auditoría con controles de integridad para evitar manipulaciones o eliminaciones, como tablas de bases de datos de solo anexión o similares.
- Asegúrese de que todas las transacciones que generen un error se reviertan y se inicien nuevamente. Siempre falla cerrado.
- Si su aplicación o sus usuarios se comportan de manera sospechosa, emita una alerta. Cree una guía para sus desarrolladores sobre este tema para que puedan codificar en contra de esto o comprar un sistema para esto.
- Los equipos de DevSecOps y de seguridad deben establecer un monitoreo y alerta efectivos de los casos de uso, incluidos manuales de estrategias para que el equipo del Centro de Operaciones de Seguridad (SOC) detecte y responda rápidamente a las actividades sospechosas.
- Agregue ‘honeytokens’ como trampas para atacantes en su aplicación, por ejemplo, en la base de datos, datos, como identidad de usuario real y/o técnica. Como no se utilizan en negocios normales, cualquier acceso genera datos de registro que pueden alertarse casi sin falsos positivos.
- El análisis del comportamiento y el soporte de IA podrían ser opcionalmente una técnica adicional para soportar bajas tasas de falsos positivos para las alertas.
- Establecer o adoptar un plan de respuesta y recuperación ante incidentes, como el Instituto Nacional de Estándares y Tecnología (NIST) 800-61r2 o posterior. Enseñe a sus desarrolladores de software cómo son los ataques e incidentes de aplicaciones para que puedan denunciarlos.

Existen productos de protección de aplicaciones comerciales y de código abierto,

como el conjunto de reglas básicas de OWASP ModSecurity, y software de correlación de registros de código abierto, como la pila Elasticsearch, Logstash, Kibana (ELK), que cuentan con paneles personalizados y alertas que pueden ayudarlo a combatir estos problemas. También existen herramientas de observabilidad comercial que pueden ayudarle a responder o bloquear ataques casi en tiempo real.

Ejemplos de escenarios de ataque.

Escenario #1: El operador del sitio web de un proveedor de planes de salud infantil no pudo detectar una violación debido a la falta de monitoreo y registro. Una parte externa informó al proveedor del plan de salud que un atacante había accedido y modificado miles de registros médicos confidenciales de más de 3,5 millones de niños. Una revisión posterior al incidente encontró que los desarrolladores del sitio web no habían abordado vulnerabilidades significativas. Como no hubo registro ni monitoreo del sistema, la violación de datos podría haber estado en curso desde 2013, un período de más de siete años.

Escenario #2: Una importante aerolínea india sufrió una violación de datos que involucró más de diez años de datos personales de millones de pasajeros, incluidos datos de pasaportes y tarjetas de crédito. La violación de datos ocurrió en un proveedor externo de alojamiento en la nube, quien notificó la violación a la aerolínea después de algún tiempo.

Escenario #3: Una importante aerolínea europea sufrió una infracción denunciable del RGPD. Según se informa, la violación fue causada por vulnerabilidades de seguridad de las aplicaciones de pago explotadas por atacantes, que recopilaron más de 400.000 registros de pagos de clientes. Como resultado, el regulador de privacidad multó a la aerolínea con 20 millones de libras.

Referencias.

- OWASP Proactive Controls: C9: Implement Logging and Monitoring
- OWASP Application Security Verification Standard: V16 Security Logging and Error Handling
- OWASP Cheat Sheet: Application Logging Vocabulary
- OWASP Cheat Sheet: Logging

- Data Integrity: Recovering from Ransomware and Other Destructive Events
- Data Integrity: Identifying and Protecting Assets Against Ransomware and Other Destructive Events
- Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events

List of Mapped CWEs

- CWE-117 Improper Output Neutralization for Logs
- CWE-221 Information Loss of Omission
- CWE-223 Omission of Security-relevant Information
- CWE-532 Insertion of Sensitive Information into Log File
- CWE-778 Insufficient Logging

OwaspTop10-2025-A10-Mal-manejo-de-condiciones-excepcionales-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

A10:2025 Mal manejo de condiciones excepcionales

Contexto.

El mal manejo de condiciones excepcionales es una nueva categoría para 2025. Esta categoría contiene 24 CWE y se centra en el manejo inadecuado de errores, errores lógicos, apertura fallida y otros escenarios relacionados derivados de condiciones y sistemas anormales que pueden encontrarse. Esta categoría tiene algunos CWE que anteriormente estaban asociados con una mala calidad del código. Esto era demasiado general para nosotros; en nuestra opinión, esta categoría más específica proporciona una mejor orientación.

CWE notables incluidos en esta categoría: *CWE-209 Generación de mensaje de error que contiene información confidencial, CWE-234 Error al manejar el parámetro faltante, CWE-274 Manejo inadecuado de privilegios insuficientes, CWE-476 Desreferencia de puntero NULL,* y *CWE-636 no falla de forma segura (‘Falla de apertura’), *

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
24	20,67%	2,95%	100,00%	37,95%	7.11

Descripción.

El mal manejo de condiciones excepcionales en el software ocurre cuando los programas no logran prevenir, detectar y responder a situaciones inusuales e impredecibles, lo que provoca fallas, comportamientos inesperados y, a veces, vulnerabilidades. Esto puede implicar una o más de las siguientes 3 fallas; la aplicación no evita que ocurra una situación inusual, no identifica la situación tal como está sucediendo y/o responde mal o no responde en absoluto a la situación posterior.

Las condiciones excepcionales pueden ser causadas por una validación de entrada faltante, deficiente o incompleta, o por un manejo tardío y de alto nivel de errores en las funciones donde ocurren, o por estados ambientales inesperados como problemas de memoria, privilegios o red, manejo inconsistente de excepciones o excepciones que no se manejan en absoluto, lo que permite que el sistema caiga en un estado desconocido e impredecible. Cada vez que una aplicación no está segura de su próxima instrucción, se ha manejado mal una condición excepcional. Los errores y excepciones difíciles de encontrar pueden amenazar la seguridad de toda la aplicación durante mucho tiempo.

Pueden ocurrir muchas vulnerabilidades de seguridad diferentes cuando manejamos mal condiciones excepcionales,

como errores lógicos, desbordamientos, condiciones de carrera, transacciones fraudulentas o problemas con la memoria, el estado, los recursos, el tiempo, la autenticación y la autorización. Este tipo de vulnerabilidades pueden afectar negativamente la confidencialidad, disponibilidad y/o integridad de un sistema o sus datos. Los atacantes manipulan el manejo defectuoso de errores de una aplicación para eliminar esta vulnerabilidad.

Cómo prevenir.

Para manejar adecuadamente una condición excepcional debemos planificar este tipo de situaciones (esperar lo peor). Debemos ‘captar’ cada posible error del sistema directamente en el lugar donde ocurre y luego manejarlo (lo que significa hacer algo significativo para resolver el problema y asegurarnos de recuperarnos del problema). Como parte del manejo, debemos incluir el lanzamiento de un error (para informar al usuario de forma comprensible), el registro del evento, así como emitir una alerta si lo consideramos justificado. También deberíamos contar con un controlador de excepciones global en caso de que alguna vez nos hayamos perdido algo. Idealmente, también tendríamos herramientas o funcionalidades de

monitoreo y/o observabilidad que vigilen errores repetidos o patrones que indiquen un ataque en curso, que podrían emitir una respuesta, defensa o bloqueo de algún tipo.Esto puede ayudarnos a bloquear y responder a scripts y bots que se centran en nuestras debilidades en el manejo de errores.

Captar y manejar condiciones excepcionales garantiza que la infraestructura subyacente de nuestros programas no tenga que lidiar con situaciones impredecibles. Si se encuentra a mitad de una transacción de cualquier tipo, es extremadamente importante que revierta cada parte de la transacción y comience de nuevo (también conocido como cierre fallido). Intentar recuperar una transacción a mitad de camino es a menudo cuando creamos errores irrecuperables.

Siempre que sea posible, agregue limitaciones de tarifas, cuotas de recursos, limitaciones y otros límites siempre que sea posible, para evitar condiciones excepcionales en primer lugar. Nada en la tecnología de la información debe ser ilimitado, ya que esto conduce a una falta de resiliencia de las aplicaciones, denegación de servicio, ataques exitosos de fuerza bruta y facturas extraordinarias en la nube. Considere si los errores repetidos idénticos, por encima de una determinada tasa, solo deben generarse como estadísticas que muestren con qué frecuencia han ocurrido y en qué período de tiempo. Esta información debe agregarse al mensaje original para no interferir con el registro y monitoreo automatizados, consulte A09:2025 Fallas de registro y alerta TJ.

Además de esto, nos gustaría incluir una validación de entrada estricta (con desinfección o escape para caracteres potencialmente peligrosos que debemos aceptar) y *centralizado* manejo de errores, registro, monitoreo y alertas, y un controlador de excepciones global. Una aplicación no debe tener múltiples funciones para manejar condiciones excepcionales, debe realizarse en un solo lugar, de la misma manera cada vez. También debemos crear requisitos de seguridad del proyecto para todos los consejos de esta sección, realizar actividades de modelado de amenazas y/o revisión de diseño seguro en la fase de diseño de nuestros proyectos, realizar revisión de código o análisis estático, así como ejecutar pruebas de estrés, rendimiento y penetración del sistema final.

Si es posible, toda su organización debe manejar las condiciones excepcionales de la misma manera, ya que facilita la revisión y auditoría del código para detectar errores en este importante control de seguridad.

Ejemplos de escenarios de ataque.

FIXME: Actualizar ejemplos para que sean más modernos

Escenario #1: El agotamiento de recursos debido al mal manejo de condiciones excepcionales (denegación de servicio) podría deberse a que la aplicación detecta excepciones cuando se cargan archivos, pero no libera recursos correctamente después. Cada nueva excepción deja los recursos bloqueados o no disponibles hasta que se agoten todos los recursos.

Escenario #2: Exposición de datos confidenciales mediante un manejo inadecuado o errores en la base de datos que revelan el error completo del sistema al usuario. El atacante continúa forzando errores para utilizar la información confidencial del sistema para crear un mejor ataque de inyección SQL. Los datos confidenciales en los mensajes de error del usuario son de reconocimiento. **Escenario #3:**La corrupción estatal en las transacciones financieras podría ser causada por un atacante que interrumpe una transacción de varios pasos mediante interrupciones de la red. Imagínese que el orden de transacción fuera: cuenta de usuario de débito, cuenta de destino de crédito, transacción de registro. Si el sistema no revierte correctamente toda la transacción (falla cerrada) cuando hay un error a mitad de camino, el atacante podría potencialmente vaciar la cuenta del usuario o posiblemente una condición de carrera que le permita enviar dinero al destino varias veces.

Referencias.

OWASP MASVS-RESILIENCE

- OWASP Cheat Sheet: Logging
- OWASP Cheat Sheet: Error Handling
- OWASP Application Security Verification Standard (ASVS): V16.5 Error Handling
- OWASP Testing Guide: 4.8.1 Testing for Error Handling
- Best practices for exceptions (Microsoft, .Net)
- Clean Code and the Art of Exception Handling (Toptal)
- General error handling rules (Google for Developers)

List of Mapped CWEs

- CWE-209 Generation of Error Message Containing Sensitive Information
- CWE-215 Insertion of Sensitive Information Into Debugging Code
- CWE-234 Failure to Handle Missing Parameter
- CWE-235 Improper Handling of Extra Parameters
- CWE-248 Uncaught Exception
- CWE-252 Unchecked Return Value
- CWE-274 Improper Handling of Insufficient Privileges
- CWE-280 Improper Handling of Insufficient Permissions or Privileges
- CWE-369 Divide By Zero
- CWE-390 Detection of Error Condition Without Action
- CWE-391 Unchecked Error Condition
- CWE-394 Unexpected Status Code or Return Value
- CWE-396 Declaration of Catch for Generic Exception
- CWE-397 Declaration of Throws for Generic Exception
- CWE-460 Improper Cleanup on Thrown Exception
- CWE-476 NULL Pointer Dereference
- CWE-478 Missing Default Case in Multiple Condition Expression
- CWE-484 Omitted Break Statement in Switch
- CWE-550 Server-generated Error Message Containing Sensitive Information
- CWE-636 Not Failing Securely (‘Failing Open’)
- CWE-703 Improper Check or Handling of Exceptional Conditions
- CWE-754 Improper Check for Unusual or Exceptional Conditions
- CWE-755 Improper Handling of Exceptional Conditions
- CWE-756 Missing Custom Error Page

OwaspTop10-2025-A11-Próximos-pasos-RC8

Traducido por Carlos Allendes & AIDA-5 - www.qualityfactory.cl

Por diseño, el Top10 de OWASP se limita innatamente a los diez riesgos más importantes. Todos los Top 10 de OWASP tienen “en la cima” riesgos considerados en profundidad para su inclusión, pero al final no lograron ser incluidos. Los demás riesgos fueron más frecuentes e impactantes.

Vale la pena el esfuerzo de identificar y remediar las dos cuestiones siguientes: organizaciones que trabajan para lograr un programa de seguridad de aplicaciones maduro, consultorías de seguridad o proveedores de herramientas que desean ampliar la cobertura de sus ofertas.

X01:2025 Falta de resiliencia de la aplicación

Contexto.

Este es un cambio de nombre de la denegación de servicio de 2021’. Se le cambió el nombre porque describía un síntoma en lugar de una causa raíz. Esta categoría se centra en los CWE que describen debilidades relacionadas con cuestiones de resiliencia. La puntuación de esta categoría fue muy ajustada con A10:2025-Mal manejo de condiciones excepcionales. Los CWE relevantes incluyen: *CWE-400 Consumo de recursos no controlado, CWE-409 Manejo inadecuado de datos altamente comprimidos (amplificación de datos), CWE-674 Recursión no controlada*, y *Bucle CWE-835 con condición de salida inalcanzable (‘Bucle infinito’).*

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
16	20,05%	4,55%	86,01%	41,47%	7,92

Descripción.

Esta categoría representa una debilidad sistémica en la forma en que las aplicaciones responden al estrés, las fallas y los casos extremos de los que no pueden recuperarse de una falla. Cuando una aplicación no maneja, resiste o se recupera con elegancia de condiciones inesperadas, limitaciones de recursos y otros eventos adversos, puede fácilmente generar problemas de disponibilidad (más comúnmente), pero también corrupción de datos, divulgación de datos confidenciales, fallas en cascada y/o elusiones de controles de seguridad.

Además X02:2025 Fallos en la gestión de memoria También puede provocar fallos en la aplicación o incluso en todo el sistema.

Cómo prevenir

Para prevenir este tipo de vulnerabilidad debes diseñar para fallas y recuperación de tus sistemas.

- Agregue límites, cuotas y funcionalidad de conmutación por error, prestando especial atención a las operaciones que consumen más recursos
- Identifique páginas que consumen muchos recursos y planifique con anticipación: reduzca la superficie de ataque, especialmente sin exponer ‘dispositivos’ innecesarios y funciones que requieren muchos recursos (por ejemplo, CPU, memoria) para usuarios desconocidos o no confiables
- Realice una validación de entrada estricta con listas permitidas y limitaciones de tamaño, luego pruebe exhaustivamente
- Limite el tamaño de las respuestas y nunca envíe respuestas sin procesar al cliente (proceso en el lado del servidor)
- El valor predeterminado es seguro/cerrado (nunca abierto), denegar de forma predeterminada y revertir si hay un error
- Evite bloquear llamadas sincrónicas en subprocesos de solicitud (use llamadas asincrónicas/sin bloqueo, tenga tiempos de espera, tenga límites de concurrencia, etc.)
- Pruebe cuidadosamente su funcionalidad de manejo de errores

- Implementar patrones de resiliencia como disyuntores, mamparos, lógica de reintento y degradación elegante
- Realice pruebas de rendimiento y carga; agregue ingeniería del caos si tiene apetito por el riesgo
- Implementar y diseñar para redundancia cuando sea razonable y asequible
- Implementar monitoreo, observabilidad y alertas
- Filtrar direcciones de remitente no válidas de acuerdo con RFC 2267
- Bloquee botnets conocidas mediante huellas dactilares, IP o dinámicamente mediante comportamiento
- Prueba de trabajo: iniciar operaciones que consumen recursos en el *atacantes* lado que no tiene grandes impactos en los usuarios normales pero sí afecta a los bots que intentan enviar una gran cantidad de solicitudes. Haga que la prueba de trabajo sea más difícil si aumenta la carga general del sistema, especialmente para sistemas que son menos confiables o parecen ser bots
- Limite el tiempo de sesión del lado del servidor en función de la inactividad y el tiempo de espera final
- Limitar el almacenamiento de información vinculada a la sesión

Ejemplos de escenarios de ataque.

Escenario #1: Los atacantes consumen intencionalmente recursos de la aplicación para provocar fallas dentro del sistema, lo que resulta en una denegación de servicio. Esto podría deberse al agotamiento de la memoria, al llenado de espacio en disco, a la saturación de la CPU o a la apertura de conexiones infinitas.

Escenario #2: Fuzzing de entrada que conduce a respuestas diseñadas que rompen la lógica empresarial de la aplicación.

Escenario #3: Los atacantes se centran en las dependencias de la aplicación, eliminan API u otros servicios externos y la aplicación no puede continuar.

Referencias.

- Hoja de trucos de OWASP: denegación de servicio
- OWASP MASVS-RESILIENCIA
- Mejores prácticas de ASP.NET Core (Microsoft)
- Resiliencia en microservicios: mamparo versus disyuntor (parser)
- Patrón de mamparo (Geeks para Geeks)
- Marco de ciberseguridad (CSF) del NIST

- Evite bloquear llamadas: vaya a Async en Java (Devlane)

Lista de CWE mapeados

- CWE-73 Control externo del nombre o ruta del archivo
- CWE-183 Lista permisiva de entradas permitidas
- CWE-256 Almacenamiento de texto sin formato de una contraseña
- CWE-266 Asignación incorrecta de privilegios
- CWE-269 Gestión inadecuada de privilegios
- CWE-286 Gestión incorrecta de usuarios
- CWE-311 Falta cifrado de datos confidenciales
- CWE-312 Almacenamiento de información confidencial en texto sin cifrar
- CWE-313 Almacenamiento de texto sin cifrar en un archivo o en un disco
- CWE-316 Almacenamiento en texto sin cifrar de información confidencial en la memoria
- Ejecución concurrente de CWE-362 utilizando un recurso compartido con sincronización incorrecta (‘Condición de carrera’)
- CWE-382 J2EE Malas prácticas: uso de System.exit()
- Canal primario desprotegido CWE-419
- CWE-434 Carga sin restricciones de archivos con tipo peligroso
- Conflicto de interpretación CWE-436
- CWE-444 Interpretación inconsistente de solicitudes HTTP (‘Contrabando de solicitudes/respuestas HTTP’)
- Interfaz de usuario (UI) CWE-451 Tergiversación de información crítica
- CWE-454 Inicialización externa de variables o almacenes de datos confiables
- CWE-472 Control externo de parámetros web supuestamente inmutables
- Violación del límite de confianza CWE-501
- CWE-522 Credenciales insuficientemente protegidas
- CWE-525 Uso de caché de navegador web que contiene información confidencial
- CWE-539 Uso de cookies persistentes que contienen información confidencial
- CWE-598 Uso del método de solicitud GET con cadenas de consulta confidenciales
- CWE-602 Aplicación de la seguridad del lado del cliente al lado del servidor
- Llamada a función CWE-628 con argumentos especificados incorrectamente
- CWE-642 Control externo de datos de estado crítico
- CWE-646 Dependencia del nombre del archivo o extensión del archivo suministrado externamente
- CWE-653 Aislamiento o compartimentación inadecuados
- CWE-656 Confianza en la seguridad a través de la oscuridad

- CWE-657 Violación de los principios de diseño seguro
- CWE-676 Uso de una función potencialmente peligrosa
- Fallo del mecanismo de protección CWE-693
- CWE-799 Control inadecuado de la frecuencia de interacción
- CWE-807 Confianza en aportaciones no confiables en una decisión de seguridad
- CWE-841 Aplicación inadecuada del flujo de trabajo conductual
- CWE-1021 Restricción incorrecta de capas o marcos de interfaz de usuario renderizados
- CWE-1022 Uso de un enlace web a un destino no confiable con acceso a window.opener
- CWE-1125 Superficie de ataque excesiva

X02:2025 Fallos en la gestión de memoria

Fondo.

Los lenguajes como Java, C#, JavaScript/TypeScript (node.js), Go y Rust “seguro” son seguros para la memoria. Los problemas de gestión de memoria tienden a ocurrir en lenguajes que no son seguros para la memoria, como C y C++. Esta categoría obtuvo la puntuación más baja en la encuesta comunitaria y baja en los datos a pesar de tener el tercer CVE más relacionado. Creemos que esto se debe al predominio de las aplicaciones web sobre las aplicaciones de escritorio más tradicionales. Las vulnerabilidades de gestión de memoria suelen tener las puntuaciones CVSS más altas.

Tabla de puntuación.

CWE mapeados	Tasa máxima de incidencia	Tasa de incidencia promedio	Cobertura máxima	Cobertura promedio	Explotación ponderada promedio
24	2,96%	1,13%	55,62%	28,45%	6,75

Descripción.

Cuando una aplicación se ve obligada a gestionar la memoria por sí misma, es muy fácil cometer errores. Los lenguajes seguros para la memoria se utilizan con

mayor frecuencia, pero todavía hay muchos sistemas heredados en producción en todo el mundo, nuevos sistemas de bajo nivel que requieren el uso de lenguajes no seguros para la memoria y aplicaciones web que interactúan con mainframes, dispositivos IoT, firmware y otros sistemas que pueden verse obligados a administrar su propia memoria. Los CWE representativos son *Copia de buffer CWE-120 sin verificar el tamaño de la entrada ('Desbordamiento de buffer clásico')* y *Desbordamiento de buffer basado en pila CWE-121*.

Las fallas en la gestión de la memoria pueden ocurrir cuando:

- No asignas suficiente memoria para una variable
- No validas la entrada, lo que provoca un desbordamiento del montón, la pila y un buffer
- Almacena un valor de datos que es mayor que el tipo de variable que puede contener
- Intenta utilizar memoria no asignada o espacios de direcciones
- Creas errores uno por uno (contando desde 1 en lugar de cero)
- Intentas acceder a un objeto después de que haya sido liberado
- Utilizas variables no inicializadas
- Pierde memoria o utiliza toda la memoria disponible por error hasta que nuestra aplicación falla

Los fallos en la gestión de la memoria pueden provocar fallos en la aplicación o incluso en todo el sistema, véase también X01:2025 Falta de resiliencia de la aplicación

Cómo prevenir.

La mejor manera de evitar fallos en la gestión de la memoria es utilizar un lenguaje seguro para la memoria. Los ejemplos incluyen Rust, Java, Go, C#, Python, Swift, Kotlin, JavaScript, etc. Al crear nuevas aplicaciones, intente convencer a su organización de que vale la pena la curva de aprendizaje para cambiar a un lenguaje seguro para la memoria. Si realiza una refactorización completa, presione para reescribir en un lenguaje seguro para la memoria cuando sea posible y factible.

Si no puede utilizar un lenguaje seguro para la memoria, realice lo siguiente:

- Habilite las siguientes funciones del servidor que hacen que los errores de administración de memoria sean más difíciles de explotar: aleatorización del diseño del espacio de direcciones (ASLR), protección de ejecución de datos (DEP) y protección contra sobrescritura con manejo estructurado de

excepciones (SEHOP).
- Supervise su aplicación para detectar fugas de memoria.
- Valide todas las entradas a su sistema con mucho cuidado y rechace todas las entradas que no cumplan con las expectativas.
- Estudia el lenguaje que estás usando y haz una lista de funciones inseguras y más seguras, luego comparte esa lista con todo tu equipo. Si es posible, agréguelo a su guía o estándar de codificación segura. Por ejemplo, en C, prefiera strncpy() sobre strcpy() y strncat() sobre strcat().
- Si su lenguaje o marco ofrece bibliotecas de seguridad de memoria, utilícelas. Por ejemplo: Safestringlib o SafeStr.
- Utilice buffers y cadenas administrados en lugar de matrices y punteros sin formato siempre que sea posible.
- Realice una capacitación en codificación segura que se centre en problemas de memoria y/o el idioma de su elección. Informe a su entrenador que le preocupan las fallas en la gestión de la memoria.
- Realizar revisiones de código y/o análisis estáticos.
- Utilice herramientas de compilación que ayuden con la administración de memoria, como StackShield, StackGuard y Libsafe.
- Realice fuzzing en cada entrada a su sistema.
- Si le realizan una prueba de penetración, informe a su evaluador que le preocupan las fallas en la administración de la memoria y que le gustaría que prestaran especial atención a esto durante la prueba.
- Corrija todos los errores del compilador *y* advertencias. No ignore las advertencias porque su programa se compila.
- Asegúrese de que su infraestructura subyacente sea parcheada, escaneada y reforzada periódicamente.
- Supervise su infraestructura subyacente específicamente para detectar posibles vulnerabilidades de memoria y otras fallas.
- Considere usar canarios para proteger su pila de direcciones de ataques de desbordamiento.

Ejemplos de escenarios de ataque.

Escenario #1: Los desbordamientos de buffer son la vulnerabilidad de memoria más famosa, una situación en la que un atacante envía más información a un campo de la que puede aceptar, de modo que desborda el buffer creado para la variable subyacente. En un ataque exitoso, los caracteres de desbordamiento sobrescriben el puntero de la pila, lo que permite al atacante insertar instrucciones maliciosas en su programa.

Escenario #2: El uso después de la liberación (UAF) ocurre con tanta frecuencia

que se trata de un envío de recompensas por errores del navegador semicomún. Imagine un navegador web que procesa JavaScript y manipula elementos DOM. El atacante crea un payload de JavaScript que crea un objeto (como un elemento DOM) y obtiene referencias a él. Mediante una manipulación cuidadosa, activan el navegador para liberar la memoria del objeto mientras mantienen un puntero colgante en él. Antes de que el navegador se dé cuenta de que se ha liberado la memoria, el atacante asigna un nuevo objeto que ocupa el *igual* espacio de memoria. Cuando el navegador intenta utilizar el puntero original, ahora apunta a datos controlados por el atacante. Si este puntero fuera para una tabla de funciones virtuales, el atacante puede redirigir la ejecución del código a su payload.

Escenario #3: Un servicio de red que acepta la entrada del usuario, no la valida ni desinfecta adecuadamente y luego la pasa directamente a la función de registro. La entrada del usuario se pasa a la función de registro como syslog(user_input) en lugar de syslog("%s", user_input), que no especifica el formato. El atacante envía cargas útiles maliciosas que contienen especificadores de formato como %x para leer la memoria de la pila (divulgación de datos confidenciales) o %n para escribir en direcciones de memoria. Al encadenar múltiples especificadores de formato, podrían mapear la pila, localizar direcciones importantes y luego sobrescribirlas. Esta sería una vulnerabilidad de cadena de formato (formato de cadena no controlado).

Nota: los navegadores modernos utilizan muchos niveles de defensas para defenderse de este tipo de ataques, incluidos sandbox del navegador ASLR, DEP/NX, RELRO y PIE. Un ataque de falla de administración de memoria en un navegador no es un ataque sencillo de llevar a cabo.

Referencias.

- Páginas de la comunidad OWASP: Fuga de memoria, Liberando doblemente la memoria,& Desbordamiento de buffer
- Awesome Fuzzing: una lista de recursos de fuzzing
- Blog del Proyecto Cero
- Blog de Microsoft MSRC

Lista de CWE mapeados

- Eliminación de código para borrar bufferes del compilador CWE-14
- CWE-119 Restricción inadecuada de operaciones dentro de los límites de un buffer de memoria

- Copia de buffer CWE-120 sin verificar el tamaño de la entrada (‘Desbordamiento de buffer clásico’)
- Desbordamiento de buffer basado en pila CWE-121
- CWE-122 Desbordamiento de buffer basado en montón
- Suscripción de buffer CWE-124 (‘Subflujo de buffer’)
- CWE-125 Lectura fuera de límites
- Sobrelectura del buffer CWE-126
- Desbordamiento o envoltura de enteros CWE-190
- 191 Subflujo de enteros (envolver o encapsular)
- CWE-196 Error de conversión de no firmado a firmado
- CWE-367 Tiempo de verificación Tiempo de uso (TOCTOU) Condición de carrera
- CWE-415 Doble Gratis
- CWE-416 Uso después de la liberación
- CWE-457 Uso de variable no inicializada
- Limpieza incompleta del CWE-459
- CWE-467 Uso de sizeof() en un tipo de puntero
- CWE-787 Escritura fuera de límites
- CWE-788 Acceso a la ubicación de la memoria después del final del buffer
- CWE-824 Acceso a puntero no inicializado