

Introducción

Bienvenido al OWASP Top 10 - 2021



TOP 10

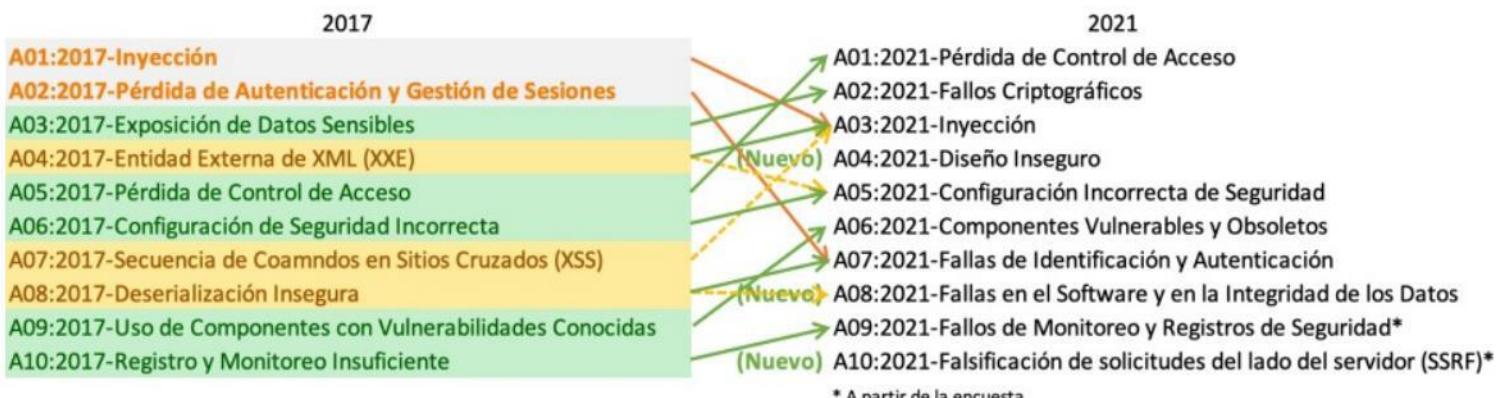


¡Bienvenido a la última entrega del OWASP Top 10! El OWASP Top 10 2021 ha sido totalmente renovado, con un nuevo diseño gráfico y una infografía de una sola página que puedes imprimir u obtener desde nuestra página web.

Un enorme agradecimiento a todos los que han contribuido con su tiempo y datos para esta iteración. Sin ustedes, esta entrega no sería posible.
GRACIAS!

Qué ha cambiado en el Top 10 de 2021

Hay tres nuevas categorías, cuatro categorías con cambios de nombre y alcance, y alguna consolidación en el Top 10 de 2021. Hemos cambiado los nombres cuando ha sido necesario para centrarnos en la causa raíz en lugar del síntoma.



- **A01:2021-Pérdida de Control de Acceso** sube de la quinta posición a la categoría con el mayor riesgo en seguridad de aplicaciones web; los datos proporcionados indican que, en promedio, el 3,81% de las aplicaciones analizadas poseían una o más de las debilidades identificadas en el Common Weakness Enumerations (CWEs), con más de 318 mil ocurrencias en estas categorías de riesgo. Las 34 CWEs relacionadas con la Pérdida de Control de Acceso tuvieron más apariciones en las aplicaciones que cualquier otra categoría.
- **A02:2021-Fallas Criptográficas** sube una posición ubicándose en la segunda, antes conocida como **A3:2017-Exposición de Datos Sensibles**, que era más una característica que una causa raíz. El nuevo nombre se centra en las fallas relacionadas con la criptografía, como se ha hecho implícitamente antes. Esta categoría frecuentemente conlleva a la exposición de datos confidenciales o al compromiso del sistema.
- **A03:2021-Inyección** descienden hasta la tercera posición. El 94% de las aplicaciones fueron analizadas en búsqueda de algún tipo de inyección, mostrando una tasa de incidencia máxima del 19% y de 3.37% de promedio. Las 33 CWEs relacionadas con esta categoría tienen la segunda mayor cantidad de ocurrencias en aplicaciones con 274.000. Las Secuencia de Comandos en Sitios Cruzados (Cross-Site Scripting o XSS), son parte de esta categoría en esta edición.
- **A04:2021-Diseño Inseguro** es nueva categoría en la edición 2021, con un foco en los riesgos relacionados a las fallas de diseño. Si realmente deseamos avanzar como industria, necesitamos más modelado de amenazas, utilizar más patrones y principios de diseño seguro y arquitecturas

de referencia. Un diseño inseguro no puede ser corregido con una implementación perfecta debido a que, por definición, los controles de seguridad necesarios nunca se crearon para defenderse de ataques específicos.

- [**A05:2021-Configuración de Seguridad Incorrecta**](#) asciende desde la sexta posición en la edición anterior. El 90% de las aplicaciones fueron probadas para detectar algún tipo de configuración incorrecta, con una tasa de incidencia promedio del 4,5% y más de 208 mil casos de CWEs relacionadas con esta categoría de riesgo. Con la adopción de software altamente configurable, no es sorprendente ver que esta categoría asciende. Las [**A4:2017-Entidades Externas XML\(XXE\)**](#) en esta edición, forman parte de esta categoría de riesgo.
- [**A06:2021-Componentes Vulnerables y Desactualizados**](#) antes denominado como Uso de Componentes con Vulnerabilidades Conocidas, ocupa el segundo lugar en el Top 10 de la encuesta a la comunidad, pero también tenía datos suficientes para estar en el Top 10 a través del análisis de datos. Esta categoría asciende desde la novena posición en la edición 2017 y es un problema conocido que cuesta probar y evaluar el riesgo. Es la única categoría que no tiene ninguna CVE (Common Vulnerability and Exposures) relacionada con las CWEs incluidas, por lo que una vulnerabilidad predeterminada y con ponderaciones de impacto de 5.0 son consideradas en sus puntajes.
- [**A07:2021-Fallas de Identificación y Autenticación**](#) previamente denominada como Pérdida de Autenticación, descendió desde la segunda posición, y ahora incluye CWEs que están más relacionadas con fallas de identificación. Esta categoría sigue siendo una parte integral del Top 10, pero el incremento en la disponibilidad de frameworks estandarizados parece estar ayudando.
- [**A08:2021-Fallas en la Integridad del Software y de los Datos**](#) es una nueva categoría para la edición 2021, enfocándose en realizar hipótesis relacionadas con actualizaciones de software, datos críticos y pipelines de CI/CD sin verificar su integridad. Uno de los mayores impactos ponderados de los datos del CVE/CVSS (Common Vulnerability and Exposures/Common Vulnerability Scoring System) asignados a los 10 CWE de esta categoría. El [**A8:2017-Deserialización Insegura**](#), pasa a formar parte de esta categoría mas amplia.
- [**A09:2021-Fallas en el Registro y Monitoreo de la Seguridad**](#) previamente denominada como [**A10:2017 Registro y Monitoreo Insuficientes**](#) es añadido a partir de la encuesta a la comunidad del Top 10 (tercer lugar), ascendiendo desde la décima posición de la edición anterior. Esta categoría se amplía para incluir más tipos de fallas que son difíciles de probar y no están bien representadas en los datos de CVE/CVSS. Igualmente, las fallas incluidas en esta categoría pueden afectar directamente la visibilidad, las alertas de incidentes y los análisis forenses.
- [**A10:2021-Falsificación de Solicitudes del Lado del Servidor \(SSRF\)**](#) es añadido a partir de la encuesta a la comunidad del Top 10 (primer lugar). Los datos muestran una tasa de incidencia relativamente baja con una cobertura de pruebas por encima del promedio, junto con calificaciones por encima del promedio para la capacidad de explotación e impacto. Esta categoría representa el escenario en el que los miembros de la comunidad de seguridad nos dicen que esto es importante, a pesar de no ser visible en los datos en este momento.

Metodología

Esta entrega del Top 10 está más orientada a los datos que nunca, pero no a ciegas. Hemos seleccionado ocho de las diez categorías a partir de los datos aportados y dos categorías a partir de la encuesta de la comunidad del Top 10 a un alto nivel. Hacemos esto por una razón fundamental, mirar los datos aportados es mirar al pasado. Los investigadores de seguridad en aplicaciones se toman su tiempo para encontrar nuevas vulnerabilidades y nuevas formas de probarlas. Se necesita tiempo para integrar estas pruebas en las herramientas y los procesos. Para el momento en que podamos probar de forma fiable una vulnerabilidad a escala, es probable que hayan pasado años. Para compensar este punto de vista, utilizamos una encuesta de la comunidad en la que preguntamos a los expertos en seguridad y desarrollo de aplicaciones que se encuentran en primera línea cuáles son, en su opinión, las vulnerabilidades esenciales que los datos aún no muestran.

Hay algunos cambios críticos que hemos adoptado para seguir madurando el Top 10.

Cómo se estructuran las categorías

Algunas categorías han cambiado con respecto a la anterior entrega del OWASP Top 10. A continuación se presenta un resumen de alto nivel de los cambios de categoría.

Los trabajos anteriores de recopilación de datos se centraban en un subconjunto prescrito de aproximadamente 30 CWEs, con un campo que solicitaba hallazgos adicionales. Nos dimos cuenta de que las organizaciones se centraban principalmente en esas 30 CWEs y rara vez añadían otras CWEs que veían. En esta iteración, lo abrimos y solo pedimos datos, sin ninguna restricción sobre las CWEs. Solicitamos el número de aplicaciones analizadas para un año determinado (a partir de 2017) y el número de aplicaciones con al menos una instancia de una CWE encontrada en las pruebas. Este formato nos permite rastrear la prevalencia de cada CWE dentro de la población de aplicaciones. Ignoramos la frecuencia para nuestros propósitos; aunque puede ser necesaria para otras situaciones, solo oculta la prevalencia real en la población de aplicaciones. El hecho de que una aplicación tenga cuatro instancias de una CWE o 4.000 instancias, no forma parte del cálculo para el Top 10. Hemos pasado de aproximadamente 30 a casi 400 CWEs para analizar en el conjunto de datos. Tenemos previsto realizar análisis de datos adicionales como complemento en el futuro. Este aumento significativo en el número de CWEs hace necesario cambiar la estructura de las categorías.

Pasamos varios meses agrupando y clasificando las CWEs y podríamos haber seguido durante más tiempo. En algún momento tuvimos que detenernos. Hay tipos de CWE de *causa raíz* y de *síntoma*, donde los de *causa raíz* serían "Falla Criptográfica" y "Configuración de Seguridad Incorrecta" en contraste con los de *síntoma* como "Exposición de Datos Sensibles" y "Denegación de Servicio". Hemos decidido centrarnos en la *causa raíz* siempre que sea posible, ya que es más lógico para proporcionar orientación sobre la identificación y la remediación. Centrarse en la *causa raíz* en lugar de en el *síntoma* no es un concepto nuevo; el Top 10 ha sido una mezcla de *síntoma* y *causa raíz*. Las CWEs también son una mezcla de *síntoma* y *causa raíz*; simplemente estamos siendo más explícitos al respecto y lo señalamos. Hay un promedio de 19,6 CWEs por categoría en esta entrega, con el límite inferior de 1 CWE para [**A10:2021-Falsificación de Solicitudes del Lado del Servidor \(SSRF\)**](#) a 40 CWEs en [**A04:2021-Diseño Inseguro**](#). Esta actualización en la estructura de las categorías ofrece beneficios adicionales de capacitación, ya que las empresas pueden centrarse en las CWE que tienen sentido para cierto lenguaje/framework.

Cómo se utilizan los datos para seleccionar las categorías

En 2017, seleccionamos las categorías según la tasa de incidencia para determinar la probabilidad, y luego las clasificamos según la discusión del equipo basada en décadas de experiencia respecto a la *explotabilidad*, la *detectabilidad* (también *probabilidad*) y el *impacto técnico*. Para 2021, queremos utilizar los datos de *explotabilidad* e *impacto* (*técnico*), si es posible.

Descargamos OWASP Dependency Check y extrajimos las puntuaciones de CVSS de Explotación e Impacto agrupadas por CWEs relacionadas. Nos costó bastante investigación y esfuerzo, ya que todas las CVEs tienen puntuaciones CVSSv2, pero hay fallos en CVSSv2 que CVSSv3 debería solucionar. A partir de cierto momento, a todas las CVEs se les asignó también una puntuación CVSSv3. Además, los rangos y fórmulas se han actualizado entre CVSSv2 y CVSSv3.

En CVSSv2, tanto *explotabilidad* como *impacto técnico* podían llegar a 10,0, pero la fórmula los rebajaba al 60% para *explotabilidad* y al 40% para *impacto técnico*. En CVSSv3, el máximo teórico se limitó a 6,0 para *explotabilidad* y 4,0 para *impacto técnico*. Con la ponderación considerada, la puntuación de impacto técnico se elevó, casi un punto y medio en promedio en CVSSv3, y la explotabilidad se redujo casi medio punto.

Hay 125.000 registros de una CVE mapeadas a una CWE en los datos de la NVD (National Vulnerability Database) extraídos del OWASP Dependency Check y hay 241 CWEs únicas mapeadas a una CVE. 62.000 mapeos de CWE tienen una puntuación CVSSv3, lo cual representa aproximadamente la mitad de la población en el conjunto de datos.

Para el Top 10 2021, calculamos las puntuaciones promedio de *explotabilidad* e *impacto técnico* de la siguiente manera. Agrupamos todas las CVEs con puntuaciones CVSS por CWE y ponderamos las puntuaciones de *explotabilidad* e *impacto técnico* por el porcentaje de la población que tenía CVSSv3 + la población restante de puntuaciones CVSSv2 para obtener una media global. Mapeamos estos promedios a las CWEs en el conjunto de datos para utilizarlos como puntuación de *explotabilidad* e *impacto (Técnico)* para la otra mitad de la ecuación del riesgo.

¿Por qué no únicamente los datos estadísticos puros?

Los resultados de los datos se limitan principalmente a lo que podemos comprobar de forma automatizada. Si hablas con un profesional experimentado de la seguridad de las aplicaciones, te hablará de las cosas que encuentra y de las tendencias que ve y que aún no están en los datos. Se necesita tiempo para desarrollar metodologías de pruebas para ciertos tipos de vulnerabilidades y más tiempo aún para que esas pruebas sean automatizadas y ejecutadas contra una gran población de aplicaciones. Todo lo que hallamos es mirando al pasado y puede que falten algunas tendencias del último año, que aún no están presentes en los datos.

Es por eso que solo recogemos ocho de las diez categorías de los datos, porque están incompletos. Las otras dos categorías proceden de la encuesta a la comunidad del Top 10. Esto permite a los profesionales de primera línea votar por los que ellos consideran que son los mayores riesgos y que podrían no estar en los datos (y puede que nunca se expresen en ellos).

¿Por qué la tasa de incidencia en lugar de la frecuencia?

Hay tres fuentes principales de datos. Las identificamos como Herramienta asistida por Humanos (HaT), Humano asistido por Herramientas (TaH) y Herramientas en bruto.

Herramientas y HaT son generadores de hallazgos de alta frecuencia. Las herramientas buscarán vulnerabilidades específicas e intentarán incansablemente encontrar cada instancia de esa vulnerabilidad y generarán un elevado número de hallazgos para algunos tipos de vulnerabilidades. Por ejemplo el Cross-Site Scripting que suelen ser de dos tipos: o bien es un error menor y aislado, o bien es un problema sistémico. Cuando se trata de un problema sistémico, el número de hallazgos puede ser de miles para una sola aplicación. Esta alta frecuencia ahoga a la mayoría de las demás vulnerabilidades encontradas en los informes o datos.

Por otro lado, TaH encontrará una gama más amplia de tipos de vulnerabilidades, pero con una frecuencia mucho menor debido a las limitaciones de tiempo. Cuando se prueba manualmente una aplicación y se ve algo como un Cross-Site Scripting, normalmente se encontrarán tres o cuatro instancias y se detendrá, siendo suficiente para determinar un hallazgo sistémico y escribir una recomendación para corregirlo en la aplicación en general. No hay necesidad (ni tiempo) de encontrar cada instancia en particular.

Supongamos que tomamos estos dos conjuntos de datos distintos y tratamos de fusionarlos por su frecuencia. En ese caso, los datos de Herramientas y HaT enmascararán los datos más precisos (aunque amplios) de TaH y es en buena parte la razón por la que algo como el Cross-Site Scripting ha sido clasificado tan alto en muchas listas, cuando el impacto es generalmente bajo o moderado. Esto se debe al gran volumen de hallazgos. El Cross-Site Scripting es relativamente fácil de probar, por lo que hay muchas más pruebas para ello.

En 2017, introdujimos el uso de la tasa de incidencia en su lugar para dar una nueva mirada y combinar de manera transparente los datos de Herramientas y HaT con los de TaH. La tasa de incidencia se refiere al porcentaje de la población de aplicaciones que tiene al menos una instancia de un tipo de vulnerabilidad. No nos importa si fue algo puntual o sistémico. Eso es irrelevante para nuestros objetivos, solo necesitamos saber cuántas aplicaciones tenían al menos una instancia, lo cual ayuda a proporcionar una visión más clara de los hallazgos de las pruebas a través de múltiples tipos de pruebas sin enmascarar los datos en resultados de alta frecuencia. Esto corresponde a una visión relacionada con el riesgo, ya que un atacante solo necesita una instancia para atacar una aplicación con éxito a través de dicha categoría.

¿Cuál es el proceso de recopilación y análisis de datos?

En el Open Security Summit de 2017 formalizamos el proceso de recopilación de datos del OWASP Top 10. Los líderes del OWASP Top 10 y la comunidad pasaron dos días trabajando en la formalización de un proceso de recopilación de datos transparente. La edición de 2021 es la segunda en la que utilizamos esta metodología.

Publicamos la solicitud de datos a través de las redes sociales que disponemos, tanto del proyecto como de OWASP. En la página del proyecto, enumeramos los elementos y la estructura de los datos que buscamos y cómo presentarlos. En el proyecto de GitHub, disponemos de archivos de ejemplo que sirven como plantillas. También, en caso necesario, trabajamos con las organizaciones para ayudarles a determinar la estructura y el mapeo con las CWEs.

Recibimos datos de organizaciones que son proveedores de servicios de seguridad, proveedores de bug bounty y organizaciones que aportan datos de pruebas internas. Una vez que recibimos los datos, los cargamos y realizamos un análisis fundamental de las categorías de riesgo de las CWEs. Hay un solapamiento entre algunas CWEs, y otros están muy relacionados (por ejemplo, las fallas criptográficas). Toda decisión relacionada con los datos en bruto se ha documentado y publicado para ser abiertos y transparentes en cuanto a su la normalización.

Analizamos las ocho categorías con las tasas de incidencia más altas para incluirlas en el Top 10. También miramos los resultados de la encuesta a la comunidad del Top 10 para ver cuáles ya se encuentran presentes en los datos. Las dos más votadas que no estén ya presentes fueron seleccionadas para los otros dos puestos del Top 10. Una vez seleccionados las diez, aplicamos factores generalizados de explotabilidad e impacto; para así poder ordenar el Top 10 2021 en función del riesgo.

Factores de datos

Existen factores de datos que se enumeran para cada una de las categorías, aquí se explica su significado:

- CWEs mapeadas: El número de CWEs asignadas a una categoría por el equipo del Top 10.
- Tasa de incidencia: La tasa de incidencia es el porcentaje de aplicaciones vulnerables a esa CWE de la población analizada por esa organización para ese año.
- Cobertura (de pruebas): El porcentaje de aplicaciones que han sido testadas por todas las organizaciones para una determinada CWE.
- Explotabilidad ponderada: La sub-puntuación de Explotabilidad de las puntuaciones CVSSv2 y CVSSv3 asignadas a las CVEs mapeadas a las CWEs, normalizados y colocados en una escala de 10 puntos.
- Impacto ponderado: La sub-puntuación de Impacto de las puntuaciones CVSSv2 y CVSSv3 asignadas a las CVEs mapeadas a las CWEs, normalizados y colocados en una escala de 10 puntos.
- Total de ocurrencias: Número total de aplicaciones en las que se han encontrado las CWEs asignados a una categoría.
- Total de CVEs: Número total de CVEs en la base de datos del NVD que fueron asignadas a las CWEs asignados a una categoría.

Agradecimiento a nuestros proveedores de datos

Las siguientes organizaciones (junto con algunos donantes anónimos) han tenido la amabilidad de donar datos de más de 500.000 aplicaciones para hacer de este el mayor y más completo conjunto de datos sobre seguridad de las aplicaciones. Sin ustedes, esto no sería posible.

- AppSec Labs
- Cobalt.io
- Contrast Security
- GitLab
- HackerOne
- HCL Technologies
- Micro Focus
- PenTest-Tools
- Probely
- Sqreen
- Veracode
- WhiteHat (NTT)

Gracias a nuestros patrocinadores

El equipo del OWASP Top 10 2021 agradece el apoyo financiero de Secure Code Warrior y Just Eat.



Acerca de OWASP

El proyecto abierto de seguridad en aplicaciones Web (OWASP por sus siglas en inglés) es una comunidad abierta dedicada a facultar a las organizaciones a desarrollar, adquirir y mantener aplicaciones y APIS que pueden ser confiables.

En OWASP, encontrará gratuitas y abiertas:

- Herramientas y estándares de seguridad en aplicaciones.
- Investigación de vanguardia.
- Controles estándar de seguridad y bibliotecas.
- Libros completos de revisiones de seguridad en aplicaciones, desarrollo de código fuente seguro, y revisiones de seguridad en código fuente.
- Presentaciones y [videos](#).
- [Hojas de ayuda \(Cheat sheets\)](#) en varios tópicos comunes.
- [Reuniones de Capítulos](#)
- [Eventos, entrenamientos y conferencias](#).
- [Grupos de Google](#)

Conozca más en: <https://www.owasp.org>.

Todas las herramientas, documentos, videos, presentaciones y capítulos de OWASP son gratuitos y están abiertos a cualquier interesado en mejorar la seguridad en aplicaciones.

Abogamos por resolver la seguridad en aplicaciones como un problema de personas, procesos y tecnología, ya que los enfoques más efectivos para la seguridad en aplicaciones requieren mejoras en todas estas áreas.

OWASP es un nuevo tipo de organización. Nuestra libertad de presiones comerciales nos permite proveer información sobre seguridad en aplicaciones sin sesgos, práctica y efectiva.

OWASP no está afiliada con ninguna compañía de tecnología, aunque apoyamos el uso instruido de tecnologías de seguridad comercial. OWASP produce muchos tipos de materiales en una manera colaborativa, transparente y abierta.

La fundación OWASP es una entidad sin fines de lucro para asegurar el éxito a largo plazo del proyecto. Casi todos los asociados con OWASP son voluntarios, incluyendo la junta directiva de OWASP, líderes de capítulos, líderes y miembros de proyectos. Apoyamos la investigación innovadora sobre seguridad a través de becas e infraestructura.

¡Únase a nosotros!

Derechos de Autor y Licencia



Copyright © 2003-2021 The OWASP™ Foundation. Este documento se publica bajo la licencia Creative Commons Attribution Share-Alike 4.0. Para cualquier reutilización o distribución, debe dejar en claro los términos de la licencia de este trabajo.

A01:2021 – Pérdida de Control de Acceso



Factores

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
34	55.97%	3.81%	6.92	5.93	94.55%	47.72%	318,487	19,013

Resumen

Subiendo desde la quinta posición, el 94% de las aplicaciones fueron probadas para detectar algún tipo de pérdida de control de acceso con una tasa de incidencia promedio del 3,81%. Tuvo la mayor cantidad de ocurrencias en el conjunto de datos analizado con más de 318.000. Las CWE (Common Weakness Enumerations) más importantes incluidas son *CWE-200: Exposición de información sensible a un actor no autorizado*, *CWE-201: Exposición de información confidencial a través de datos enviados*, y *CWE-352: Falsificación de Peticiones en Sitios Cruzados (Cross Site Request Forgery, CSRF por su siglas en inglés)*.

Descripción

El control de acceso implementa el cumplimiento de política de modo que los usuarios no pueden actuar fuera de los permisos que le fueron asignados. Las fallas generalmente conducen a la divulgación de información no autorizada, la modificación o la destrucción de todos los datos o la ejecución de una función de negocio fuera de los límites del usuario. Las vulnerabilidades comunes de control de acceso incluyen:

- Violación del principio de mínimo privilegio o denegación por defecto, según el cual el acceso sólo debe ser permitido para capacidades, roles o usuarios particulares, y no disponible para cualquier persona.
- Eludir las comprobaciones de control de acceso modificando la URL (alteración de parámetros o navegación forzada), el estado interno de la aplicación o la página HTML, o mediante el uso de una herramienta que modifique los pedidos a APIs.
- Permitir ver o editar la cuenta de otra persona, con tan solo conocer su identificador único (referencia directa insegura a objetos)
- Acceder a APIs con controles de acceso inexistentes para los métodos POST, PUT y DELETE.
- Elevación de privilegios. Actuar como usuario sin haber iniciado sesión o actuar como administrador cuando se inició sesión como usuario regular.
- Manipulación de metadatos, como reutilizar o modificar un token de control de acceso JSON Web Token (JWT), una cookie o un campo oculto, manipulándolos para elevar privilegios o abusar de la invalidación de tokens JWT.
- Configuraciones incorrectas de CORS (uso compartido de recursos de origen cruzado) que permiten el acceso a APIs desde orígenes no autorizados o confiables.
- Forzar la navegación a páginas autenticadas siendo usuario no autenticado o a páginas privilegiadas siendo usuario regular.

Cómo se previene

El control de acceso solo es efectivo si es implementado en el servidor (server-side) o en la API (caso serverless), donde el atacante no puede modificarlo ni manipular metadatos.

- A excepción de los recursos públicos, denegar por defecto.
- Implemente mecanismos de control de acceso una única vez y reutilícelos en toda la aplicación, incluyendo la minimización del uso de CORS.
- El control de acceso debe implementar su cumplimiento a nivel de dato y no permitir que el usuario pueda crear, leer, actualizar o borrar cualquier dato.
- Los modelos de dominio deben hacer cumplir los requisitos únicos de límite de negocio de aplicaciones.
- Deshabilite el listado de directorios del servidor web y asegúrese de que los archivos de metadatos (por ejemplo una carpeta .git) y archivos de

respaldo no puedan ser accedidos a partir de la raíz del sitio web.

- Registre las fallas de control de acceso (loggin), alertando a los administradores cuando sea apropiado (por ejemplo, fallas repetidas).
- Estableza límites a la tasa de accesos permitidos a APIs y controladores de forma de poder minimizar el daño provocado por herramientas automatizadas de ataque.
- Los identificadores de sesiones deben invalidarse en el servidor luego de cerrar la sesión. Los tokens JWT deberían ser preferiblemente de corta duración para minimizar la ventana de oportunidad de ataque. Para tokens JWT de mayor duración, es sumamente recomendable seguir los estándares de OAuth de revocación de acceso.

Tanto desarrolladores como personal de control de calidad deben incluir pruebas funcionales de control de acceso tanto a nivel unitario como de integración.

Ejemplos de escenarios de ataque

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

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

Un atacante simplemente modifica el parámetro 'acct' en el navegador para enviar el número de cuenta que desee. Si no es verificado correctamente, el atacante puede acceder a la cuenta de cualquier usuario.

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

Escenario #2: Un atacante simplemente navega a una URL específica. Se deberían requerir 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 una persona que no es administrador puede acceder a la página de administración, esto es también una falla.

Referencias

- [OWASP Proactive Controls: Enforce Access Controls](#)
- [OWASP Application Security Verification Standard: V4 Access Control](#)
- [OWASP Testing Guide: Authorization Testing](#)
- [OWASP Cheat Sheet: Access Control](#)
- [OWASP Cheat Sheet: Authorization](#)
- [PortSwigger: Exploiting CORS misconfiguration](#)
- [OAuth: Revoking Access](#)

Lista de CWEs mapeadas

[CWE-22 Improper Limitation of a Pathname to a Restricted Directory \('Path Traversal'\)](#)

[CWE-23 Relative Path Traversal](#)

[CWE-35 Path Traversal: '.../...//'](#)

[CWE-59 Improper Link Resolution Before File Access \('Link Following'\)](#)

[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-264 Permissions, Privileges, and Access Controls \(should no longer be used\)](#)

[CWE-275 Permission Issues](#)

[CWE-276 Incorrect Default Permissions](#)

[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-402 Transmission of Private Resources into a New Sphere \('Resource Leak'\)](#)

[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-639 Authorization Bypass Through User-Controlled Key](#)

[CWE-651 Exposure of WSDL File Containing Sensitive Information](#)

[CWE-668 Exposure of Resource to Wrong Sphere](#)

[CWE-706 Use of Incorrectly-Resolved Name or Reference](#)

[CWE-862 Missing Authorization](#)

[CWE-863 Incorrect Authorization](#)

[CWE-913 Improper Control of Dynamically-Managed Code Resources](#)

[CWE-922 Insecure Storage of Sensitive Information](#)

[CWE-1275 Sensitive Cookie with Improper SameSite Attribute](#)

A02:2021 – Fallas Criptográficas



Factores

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
29	46.44%	4.49%	7.29	6.81	79.33%	34.85%	233,788	3,075

Resumen

Subiendo una posición al número 2, anteriormente conocido como *Exposición de datos sensibles*, que es más un amplio síntoma que una causa raíz, la atención se centra en las fallas relacionadas con la criptografía (o la falta de ésta). Esto a menudo conduce a la exposición de datos sensibles. Las CWE incluidas son *CWE-259: Uso de contraseña en código fuente*, *CWE-327: Algoritmo criptográfico vulnerado o inseguro* y *CWE-331: Entropía insuficiente*.

Descripción

Lo primero es determinar las necesidades de protección de los datos en tránsito y en reposo. Por ejemplo, contraseñas, números de tarjetas de crédito, registros médicos, información personal y secretos comerciales requieren protección adicional, principalmente si están sujetos a leyes de privacidad (por ejemplo, el Reglamento General de Protección de Datos -GDPR- de la UE), o regulaciones, (por ejemplo, protección de datos financieros como el Estándar de Seguridad de Datos de PCI -PCI DSS-). Para todos esos datos:

- ¿Se transmiten datos en texto claro? Esto se refiere a protocolos como HTTP, SMTP, FTP que también utilizan actualizaciones de TLS como STARTTLS. El tráfico externo de Internet es peligroso. Verifique todo el tráfico interno, por ejemplo, entre balanceadores de carga, servidores web o sistemas de back-end.
- ¿Se utilizan algoritmos o protocolos criptográficos antiguos o débiles de forma predeterminada o en código antiguo?
- ¿Se utilizan claves criptográficas predeterminadas, se generan o reutilizan claves criptográficas débiles, o es inexistente la gestión o rotación de claves adecuadas? ¿Se incluyen las claves criptográficas en los repositorios de código fuente?
- ¿No es forzado el cifrado, por ejemplo, faltan las directivas de seguridad de los encabezados HTTP (navegador) o los encabezados?
- ¿El certificado de servidor recibido y la cadena de confianza se encuentran debidamente validados?
- ¿Los vectores de inicialización se ignoran, se reutilizan o no se generan de forma suficientemente seguros para el modo de operación criptográfico? ¿Se utiliza un modo de funcionamiento inseguro como el ECB? ¿Se utiliza un cifrado cuando el cifrado autenticado es más apropiada?
- ¿Las contraseñas se utilizan como claves criptográficas en ausencia de una función de derivación de claves a partir de contraseñas?

- ¿Se utiliza con fines criptográficos generadores de aleatoriedad que no fueron diseñados para dicho fin? Incluso si se elige la función correcta, debe ser inicializada (seed) por el desarrollador y, de no ser así, ¿el desarrollador ha sobrescrito la funcionalidad de semilla fuerte incorporada con una semilla que carece de suficiente entropía/imprevisibilidad?
- ¿Se utilizan funciones hash en obsoletas, como MD5 o SHA1, o se utilizan funciones hash no criptográficas cuando se necesitan funciones hash criptográficas?
- ¿Se utilizan métodos criptográficos de relleno(padding) obsoletos, como PKCS número 1 v1.5?
- ¿Se pueden explotar los mensajes de errores criptográficos como un canal lateral, por ejemplo, en forma de ataques de criptoanálisis por modificación relleno (Oracle Padding)?

Consulte ASVS Crypto (V7), Data Protection (V9) y SSL/TLS (V10)

Cómo se previene

Haga lo siguiente como mínimo, y consulte las referencias:

- Clasifique los datos procesados, almacenados o transmitidos por una aplicación. Identifique qué datos son confidenciales de acuerdo con las leyes de privacidad, los requisitos reglamentarios o las necesidades comerciales.
- No almacene datos sensibles innecesariamente. Descártelos lo antes posible o utilice una utilización de tokens compatible con PCI DSS o incluso el truncamiento. Los datos que no se conservan no se pueden robar.
- Asegúrese de cifrar todos los datos sensibles en reposo (almacenamiento).
- Garantice la implementación de algoritmos, protocolos y claves que utilicen estándares sólidos y actualizados; utilice una gestión de claves adecuada.
- Cifre todos los datos en tránsito con protocolos seguros como TLS con cifradores de confidencialidad adelantada (forward secrecy, o FS), priorización de cifradores por parte del servidor y parámetros seguros. Aplique el cifrado mediante directivas como HTTP Strict Transport Security (HSTS).
- Deshabilite el almacenamiento en caché para respuestas que contengan datos sensibles.
- Aplique los controles de seguridad requeridos según la clasificación de los datos.
- No utilice protocolos antiguos como FTP y SMTP para transportar datos sensibles.
- Almacene las contraseñas utilizando funciones robustas, flexibles, que utilicen sal en los hashes y use un factor de retraso (factor de trabajo), como Argon2, scrypt, bcrypt o PBKDF2.
- Elija vectores de inicialización apropiados para el modo de operación. Para muchos modos, esto significa usar 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 solo cifrado.
- Las claves deben generarse criptográficamente al azar y almacenarse en la memoria como arrays de bytes. Si se utiliza una contraseña, debe convertirse en una clave mediante una función adecuada de derivación de claves basada en contraseña.
- Asegúrese de que se utilice la aleatoriedad criptográfica cuando sea apropiado y que no se utilice una semilla de una manera predecible o con baja entropía. La mayoría de las API modernas no requieren que el desarrollador genere el CSPRNG para obtener seguridad.
- Evite las funciones criptográficas y los esquemas de relleno(padding) en desuso, como MD5, SHA1, PKCS número 1 v1.5.
- Verifique de forma independiente la efectividad de la configuración y los ajustes.

Ejemplos de escenarios de ataque

Escenario #1: Una aplicación cifra los números de tarjetas de crédito en una base de datos mediante el cifrado automático de la base de datos. Sin embargo, estos datos se descifran automáticamente cuando se recuperan, lo que permite que por una falla de inyección SQL se recuperen números de tarjetas de crédito en texto sin cifrar.

Escenario #2: Un sitio no utiliza ni aplica TLS para todas sus 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. El atacante luego reutiliza 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 #3: La base de datos de contraseñas utiliza hashes simples o sin un valor inicial aleatorio único(salt) para almacenar todas las contraseñas. Una falla en la carga de archivos permite a un atacante recuperar la base de datos de contraseñas. Todos los hashes sin salt se pueden calcular a partir de una rainbow table de hashes precalculados. Los hash generados por funciones hash simples o rápidas pueden ser descifrados a través de cálculos intensivos provistos por una o mas GPUs, incluso si utilizan un salt.

Referencias

- [OWASP Proactive Controls: Protect Data Everywhere](#)
- [OWASP Application Security Verification Standard \(V7, 9, 10\)](#)

- [OWASP Cheat Sheet: Transport Layer Protection](#)
- [OWASP Cheat Sheet: User Privacy Protection](#)
- [OWASP Cheat Sheet: Password and Cryptographic Storage](#)
- [OWASP Cheat Sheet: HSTS](#)
- [OWASP Testing Guide: Testing for weak cryptography](#)

Lista de CWEs mapeadas

[CWE-261 Weak Encoding for Password](#)

[CWE-296 Improper Following of a Certificate's Chain of Trust](#)

[CWE-310 Cryptographic Issues](#)

[CWE-319 Cleartext Transmission of Sensitive Information](#)

[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-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-347 Improper Verification of Cryptographic Signature](#)

[CWE-523 Unprotected Transport of Credentials](#)

[CWE-720 OWASP Top Ten 2007 Category A9 - Insecure Communications](#)

[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-818 Insufficient Transport Layer Protection](#)

[CWE-916 Use of Password Hash With Insufficient Computational Effort](#)

A03:2021 – Inyección

Factores



CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
33	19.09%	3.37%	7.25	7.15	94.04%	47.90%	274,228	32,078

Resumen

La Inyección se desliza hasta la tercera posición. El 94% de las aplicaciones fueron probadas sobre alguna forma de inyección con una tasa de incidencia máxima del 19%, una tasa de incidencia promedio del 3% y 274 mil ocurrencias. Las CWE notables incluidas son *CWE-79: Cross-site Scripting*, *CWE-89: SQL Injection*, y la *CWE-73:Control Externo de Nombre de archivos o ruta*.

Descripción

Una aplicación es vulnerable a este ataque cuando:

- Los datos proporcionados por el usuario no son validados, filtrados ni sanitizados por la aplicación.
- Se invocan consultas dinámicas o no parametrizadas, sin codificar los parámetros de forma acorde al contexto.
- Se utilizan datos dañinos dentro de los parámetros de búsqueda en consultas Object-Relational Mapping (ORM), para extraer registros adicionales sensibles.
- Los datos dañinos se usan directamente o se concatenan, de modo que el SQL o comando resultante contiene datos y estructuras con consultas dinámicas, comandos o procedimientos almacenados.

Algunas de las inyecciones más comunes son SQL, NoSQL, comandos de SO, Object-Relational Mapping (ORM), LDAP, expresiones de lenguaje u Object Graph Navigation Library (OGNL). El concepto es idéntico entre todos los intérpretes. La revisión del código fuente es el mejor método para detectar si las aplicaciones son vulnerables a inyecciones, seguido de cerca por pruebas automatizadas de todos los parámetros, encabezados, URL, cookies, JSON, SOAP y entradas de datos XML.

Las organizaciones pueden incluir herramientas de análisis estático (SAST) y pruebas dinámicas (DAST) para identificar errores de inyecciones recientemente introducidas y antes del despliegue de la aplicación en producción.

Cómo se previene

Para prevenir inyecciones, se requiere separar los datos de los comandos y las consultas.

- La opción preferida es utilizar una API segura, que evite el uso de un intérprete por completo y proporcione una interfaz parametrizada. Se debe migrar y utilizar una herramienta de Mapeo Relacional de Objetos (ORM).
Nota: Incluso cuando se parametrizan, los procedimientos almacenados pueden introducir una inyección SQL si el procedimiento PL/SQL o T-SQL concatena consultas y datos, o se ejecutan parámetros utilizando EXECUTE IMMEDIATE o exec().
- Realice validaciones de entradas de datos en el servidor, utilizando "listas blancas". De todos modos, esto no es una defensa completa, ya que muchas aplicaciones requieren el uso de caracteres especiales, como en campos de texto, APIs o aplicaciones móviles.
- Para cualquier consulta dinámica residual, escape caracteres especiales utilizando la sintaxis de caracteres específica para el intérprete que se trate.
Nota: La estructura de SQL como nombres de tabla, nombres de columna, etc. no se pueden escapar y, por lo tanto, los nombres de estructura suministrados por el usuario son peligrosos. Este es un problema común en el software de redacción de informes.
- Utilice LIMIT y otros controles SQL dentro de las consultas para evitar la fuga masiva de registros en caso de inyección SQL.

Ejemplos de escenarios de ataque

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

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

Escenario #2: Del mismo modo, la confianza total de una aplicación en su framework puede resultar en consultas que aún son vulnerables a inyección, (por ejemplo: Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID=' " + request.getParameter("id") + "'");
```

En ambos casos, el atacante modifica el valor del parámetro "id" en su navegador para enviar: ' or '1'='1. Por ejemplo:

```
http://example.com/app/accountView?id=' or '1'='1
```

Esto cambia el significado de ambas consultas, devolviendo todos los registros de la tabla "accounts". Ataques más peligrosos podrían modificar los 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](#)

Lista de CWEs mapeadas

[CWE-20 Improper Input Validation](#)

[CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component \('Injection'\)](#)

[CWE-75 Failure to Sanitize Special Elements into a Different Plane \(Special Element Injection\)](#)

[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-87 Improper Neutralization of Alternate XSS Syntax](#)

[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-100 Deprecated: Was catch-all for input validation issues](#)

[CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers \('HTTP Response Splitting'\)](#)

[CWE-116 Improper Encoding or Escaping of Output](#)

[CWE-138 Improper Neutralization of Special Elements](#)

[CWE-184 Incomplete List of Disallowed Inputs](#)

[CWE-470 Use of Externally-Controlled Input to Select Classes or Code \('Unsafe Reflection'\)](#)

[CWE-471 Modification of Assumed-Immutable Data \(MAID\)](#)

[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-652 Improper Neutralization of Data within XQuery Expressions \('XQuery Injection'\)](#)

[CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement \('Expression Language Injection'\)](#)

A04:2021 – Diseño Inseguro

Factores



CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
40	24.19%	3.00%	6.46	6.78	77.25%	42.51%	262,407	2,691

Resumen

Una nueva categoría para 2021 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. Como comunidad, debemos ir más allá de solo abandonar las metodologías tradicionales (movimiento "shift-left" en inglés) en el espacio de codificación para precodificar actividades que son críticas para los principios de Secure by Design. Las CWE notables incluidas son *CWE-209: Generación de mensaje de error que contiene información confidencial*, *CWE-256: Almacenamiento desprotegido de credenciales*, *CWE-501: Infracción de límites de confianza* y *CWE-522: Credenciales protegidas insuficientemente*.

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 otras 10 categorías de riesgo principales. Existe una diferencia entre un diseño inseguro y una implementación insegura. Distinguimos entre fallas de diseño y defectos de implementación por una razón, tienen diferentes causas y soluciones. Un diseño seguro aún puede tener defectos de implementación que conduzcan a vulnerabilidades que pueden explotarse. Un diseño inseguro no se puede arreglar con una implementación perfecta, ya que, por definición, los controles de seguridad necesarios nunca se crearon para defenderse de ataques específicos. Uno de los factores que contribuyen al diseño inseguro es la falta de perfiles de riesgo empresarial inherentes al software o sistema que se está desarrollando y, por lo tanto, la falta de determinación del nivel de diseño de seguridad que se requiere.

Gestión de requerimientos y recursos

Recopile y negocie los requerimientos 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 de negocio esperada. Tenga en cuenta qué tan expuesta estará su aplicación y si necesita segregación de tenants (además del control de acceso). Compile los requisitos técnicos, incluidos los requerimientos de seguridad funcionales y no funcionales. Planifique y negocie el presupuesto que cubra todo el diseño, construcción, prueba 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 estar integrado en sesiones de refinamiento (o actividades similares); buscar cambios en los flujos de datos y el control de acceso u otros controles de seguridad. Durante el desarrollo de la historia de usuario, determine el flujo correcto y los estados de falla, asegúrese de que sean bien entendidos y acordados por las partes responsables e impactadas. Analice las suposiciones y las condiciones para los flujos esperados y de falla, asegúrese de que aún sean precisos y deseables. Determine cómo validar las suposiciones y hacer cumplir las condiciones necesarias para los comportamientos adecuados. 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 pueda agregar al software.

Ciclo de Vida de Desarrollo Seguro (S-SDLC)

El software seguro requiere un ciclo de vida de desarrollo seguro, alguna forma de patrón de diseño seguro, metodologías "Paved Road", biblioteca de componentes seguros, herramientas y modelado de amenazas. Comuníquese con sus especialistas en seguridad al comienzo de un proyecto de software durante todo el proyecto y el mantenimiento de su software. Considere aprovechar el [Modelo de Madurez para el Aseguramiento del Software \(SAMM\)](#) para ayudar a estructurar sus esfuerzos de desarrollo de software seguro.

Cómo se previene

- Establezca y use un ciclo de vida de desarrollo seguro con Aplicaciones de Seguridad profesionales para ayudar a evaluar y diseñar la seguridad y controles relacionados con la privacidad.
- Establecer y utilizar una biblioteca de patrones de diseño seguros o componentes de "Paved Road"
- Utilice el modelado de amenazas para autenticación crítica, control de acceso, lógica empresarial y flujos clave.
- Integre el lenguaje y los controles de seguridad en las "historias de usuario".
- Integre verificaciones de plausibilidad en cada nivel de su aplicación (de frontend a backend)
- Escribir pruebas de integración y pruebas unitarias para validar que todos los flujos críticos son resistentes al modelo de amenazas. Compilar casos de uso y casos de uso indebido para cada nivel de su aplicación.
- Separe las capas de niveles en el sistema y las capas de red según las necesidades de exposición y protección.
- Separe a los tenants de manera robusta por diseño en todos los niveles.
- Limitar el consumo de recursos por usuario o servicio.

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 preguntas y respuestas como evidencia de identidad como más de una persona puede conocer las respuestas, por lo que están prohibidas. Dicho código debe eliminarse y reemplazarse por un diseño más seguro.

Escenario #2: Una cadena de cines permite descuentos en la reserva de grupos y tiene un máximo de quince asistentes antes de solicitar un depósito. Los atacantes podrían modelar este flujo y probar si podían reservar seiscientos asientos 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 revender sitios web de subastas. Esto crea una publicidad terrible para los fabricantes de tarjetas de video y los propietarios de cadenas minoristas y una mala sangre duradera con entusiastas que no pueden obtener estas tarjetas a ningún precio. El diseño cuidadoso de anti-bot y las reglas de lógica de dominio, como las compras realizadas a los pocos segundos de disponibilidad, pueden identificar compras no auténticas y rechazar dichas transacciones.

Referencias

- [OWASP Cheat Sheet: Secure Design Principles](#)
- [OWASP SAMM: Design:Security Architecture](#)
- [OWASP SAMM: Design:Threat Assessment](#)
- [NIST – Guidelines on Minimum Standards for Developer Verification of Software](#)
- [The Threat Modeling Manifesto](#)
- [Awesome Threat Modeling](#)

Lista de CWEs mapeadas

[CWE-73 External Control of File Name or Path](#)

[CWE-183 Permissive List of Allowed Inputs](#)

[CWE-209 Generation of Error Message Containing Sensitive Information](#)

[CWE-213 Exposure of Sensitive Information Due to Incompatible Policies](#)

[CWE-235 Improper Handling of Extra Parameters](#)

[CWE-256 Unprotected Storage of Credentials](#)

[CWE-257 Storing Passwords in a Recoverable Format](#)

[CWE-266 Incorrect Privilege Assignment](#)

[CWE-269 Improper Privilege Management](#)

[CWE-280 Improper Handling of Insufficient Permissions or Privileges](#)

[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-419 Unprotected Primary Channel](#)

[CWE-430 Deployment of Wrong Handler](#)

[CWE-434 Unrestricted Upload of File with Dangerous Type](#)

[CWE-444 Inconsistent Interpretation of HTTP Requests \('HTTP Request Smuggling'\)](#)

[CWE-451 User Interface \(UI\) Misrepresentation of Critical Information](#)

[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-579 J2EE Bad Practices: Non-Serializable Object Stored in Session](#)

[CWE-598 Use of GET Request Method With Sensitive Query Strings](#)

[CWE-602 Client-Side Enforcement of Server-Side Security](#)

[CWE-642 External Control of Critical State Data](#)

[CWE-646 Reliance on File Name or Extension of Externally-Supplied File](#)

[CWE-650 Trusting HTTP Permission Methods on the Server Side](#)

[CWE-653 Insufficient Compartmentalization](#)

[CWE-656 Reliance on Security Through Obscurity](#)

[CWE-657 Violation of Secure Design Principles](#)

[CWE-799 Improper Control of Interaction Frequency](#)

[CWE-807 Reliance on Untrusted Inputs in a Security Decision](#)

[CWE-840 Business Logic Errors](#)

[CWE-841 Improper Enforcement of Behavioral Workflow](#)

[CWE-927 Use of Implicit Intent for Sensitive Communication](#)

[CWE-1021 Improper Restriction of Rendered UI Layers or Frames](#)

[CWE-1173 Improper Use of Validation Framework](#)

A05:2021 – Configuración de Seguridad Incorrecta



Factores

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
20	19.84%	4.51%	8.12	6.56	89.58%	44.84%	208,387	789

Resumen

Pasando del puesto #6 en la edición anterior, el 90% de las aplicaciones se probaron para detectar algún tipo de configuración incorrecta, con una tasa de incidencia promedio del 4.% y más de 208k ocurrencias de una enumeración de debilidad común (CWE) en esta categoría de riesgo. Con más cambios en software altamente configurable, no es sorprendente ver que esta categoría asciende. Las CWE notables incluidas son *CWE-16 Configuración* y *CWE-611 Restricción incorrecta de la referencia de entidad externa XML*.

Descripción

La aplicación puede ser vulnerable si la aplicación:

- Le falta el refuerzo de seguridad adecuado en cualquier parte de la pila de aplicaciones o permisos configurados incorrectamente en los servicios en la nube
- Tiene funciones innecesarias habilitadas o instaladas (por ejemplo, puertos, servicios, páginas, cuentas o privilegios innecesarios).
- Las cuentas predeterminadas y sus contraseñas aún están habilitadas y sin cambios.
- El manejo de errores revela a los usuarios rastros de pila u otros mensajes de error demasiado informativos.
- Para sistemas actualizados, las últimas funciones de seguridad están deshabilitadas o no configuradas de forma segura.
- Las configuraciones de seguridad en los servidores de aplicaciones, frameworks de aplicaciones (por ejemplo, Struts, Spring, ASP.NET), bibliotecas, bases de datos, etc., no tienen configurados valores seguros.
- El servidor no envía encabezados o directivas de seguridad, o no tienen configurados valores seguros.
- El software está desactualizado o es vulnerable (consulte [A06:2021-Componentes Vulnerables y Desactualizados](#)).

Sin un proceso de configuración de seguridad de aplicaciones coordinado y repetible, los sistemas corren un mayor riesgo.

Cómo se previene

Deben implementarse procesos de instalación seguros, incluyendo:

- Un proceso de endurecimiento repetible agiliza y facilita la implementación de otro entorno que esté debidamente bloqueado. Los entornos de desarrollo, control de calidad y producción deben configurarse de forma idéntica, con diferentes credenciales utilizadas en cada entorno. Este proceso debe automatizarse para minimizar el esfuerzo necesario para configurar un nuevo entorno seguro.
- Una plataforma mínima sin funciones, componentes, documentación ni ejemplos innecesarios. Elimine o no instale características y frameworks 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 (consulte [A06: 2021-Componentes Vulnerables y Desactualizados](#)). Revise los permisos de almacenamiento en la nube (por ejemplo, Permisos de bucket de S3).
- Una arquitectura de aplicación segmentada proporciona una separación efectiva y segura entre componentes o instancias, con segmentación, organización en contenedores o grupos de seguridad en la nube (ACLs).
- Envío de directivas de seguridad a los clientes, por ejemplo, encabezados de seguridad.
- Un proceso automatizado para verificar la efectividad de las configuraciones y ajustes en todos los entornos.

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 no se modificaron las cuentas predeterminadas. En ese caso, el atacante inicia sesión con las contraseñas predeterminadas y toma el control.

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

Escenario #3: La configuración del servidor de aplicaciones permite que se devuelvan a los usuarios mensajes de error detallados, por ejemplo, seguimientos de pila(stack traces). Esto potencialmente expone información confidencial o fallas subyacentes, como versiones de componentes que se sabe que son vulnerables.

Escenario #4: Un proveedor de servicios en la nube tiene permisos de uso compartido predeterminados abiertos a Internet por otros usuarios del encabezado de política de seguridad de contenido (CSP). Esto permite acceder a los 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 V14 Configuration](#)
- [NIST Guide to General Server Hardening](#)
- [CIS Security Configuration Guides/Benchmarks](#)
- [Amazon S3 Bucket Discovery and Enumeration](#)

Lista de CWEs mapeadas

[CWE-27PK - Environment](#)

[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-520 .NET Misconfiguration: Use of Impersonation](#)

[CWE-526 Exposure of Sensitive Information Through Environmental Variables](#)

[CWE-537 Java Runtime Error Message Containing Sensitive Information](#)

[CWE-541 Inclusion of Sensitive Information in an Include File](#)

[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-756 Missing Custom Error Page](#)

[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-1032 OWASP Top Ten 2017 Category A6 - Security Misconfiguration](#)

[CWE-1174 ASP.NET Misconfiguration: Improper Model Validation](#)

A06:2021 – Componentes Vulnerables y Desactualizados

Factores



CWEs	Tasa de	Tasa de incidencia	Explotabilidad	Impacto ponderado	Cobertura	Cobertura	Incidencias	Total
------	---------	--------------------	----------------	-------------------	-----------	-----------	-------------	-------

mapeadas	incidencia máx	prom	ponderada prom	prom	máx	prom	totales	CVEs
3	27.96%	8.77%	51.78%	22.47%	5.00	5.00	30,457	0

Resumen

Era el #2 de la encuesta de la comunidad Top 10, pero también tuvo datos suficientes para llegar al Top 10 a través del análisis de datos. Los componentes vulnerables son un problema conocido que es difícil de testear y evaluar el riesgo y es la única categoría que no tiene enumeraciones de debilidades comunes (CWE) asignadas a las CWE incluidas, por lo que se utiliza un peso de impacto/exploits predeterminado de 5.0. Las CWE notables incluidas son *CWE-1104: Uso de componentes de terceros no mantenidos* y las dos CWE del Top 10 2013 y 2017.

Descripción

Usted probablemente sea vulnerable:

- Si no conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del lado del servidor). Esto incluye los componentes que usa directamente, así como las dependencias anidadas.
- Si el software es vulnerable, carece de soporte o no está actualizado. Esto incluye el sistema operativo, el servidor web/de aplicaciones, el sistema de administración de bases de datos (DBMS), las aplicaciones, las API y todos los componentes, los entornos de ejecución y las bibliotecas.
- Si no se actualiza sobre nuevas vulnerabilidades con regularidad y se suscribe a los boletines de seguridad relacionados con los componentes que utiliza.
- Si no repara o actualiza la plataforma subyacente, los frameworks y las dependencias de manera oportuna y basada en el riesgo. Esto suele ocurrir en entornos en los que la aplicación de parches de seguridad es una tarea mensual o trimestral bajo control de cambios, lo que deja a las organizaciones abiertas a días o meses de exposición innecesaria a vulnerabilidades reparadas.
- Si los desarrolladores de software no testean la compatibilidad de las bibliotecas actualizadas, actualizadas o parchadas.
- Si no asegura las configuraciones de los componentes (consulte [A05: 2021-Configuración de Seguridad Incorrecta](#)).

Cómo se previene

Debe existir un proceso de administración de parches que:

- Elimine las dependencias que no son utilizadas, las funciones, los componentes, los archivos y la documentación innecesarios.
- Realice un inventario continuo de las versiones de los componentes del lado del cliente y del lado del servidor (por ejemplo, frameworks, bibliotecas) y sus dependencias utilizando herramientas como Versions Maven Plugin, OWASP Dependency Check, retire.js, etc. Supervise continuamente fuentes como Common Vulnerability and Exposures (CVE) y National Vulnerability Database (NVD) para detectar vulnerabilidades en los componentes. Utilice herramientas de análisis de composición de software para automatizar el proceso. Suscríbase para recibir alertas por correo electrónico sobre vulnerabilidades de seguridad relacionadas con los componentes que utiliza.
- Solo obtenga componentes de fuentes oficiales a través de enlaces seguros. Prefiera los paquetes firmados para reducir la posibilidad de incluir un componente malicioso modificado (consulte A08: 2021-Fallos de integridad de datos y software).
- Supervise las bibliotecas y los componentes que no se mantienen o no crean parches de seguridad para versiones anteriores. Si la aplicación de parches no es posible, considere implementar un parche virtual para monitorear, detectar o protegerse contra el problema descubierto.

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 portafolio de aplicaciones.

Ejemplos de escenarios de ataque

Escenario #1: 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. Tales fallas pueden ser accidentales (por ejemplo, error de codificación) o intencionales (por ejemplo, una puerta trasera en un componente). Algunos ejemplos de vulnerabilidades de componentes explotables descubiertos son:

- CVE-2017-5638, una vulnerabilidad de ejecución remota de código de Struts 2 que permite la ejecución de código arbitrario en el servidor, ha sido acusada de infracciones importantes.
- Si bien el Internet de las cosas (IoT) es con frecuencia difícil o imposible de parchear, la importancia de parchearlo 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 motor de búsqueda Shodan IoT puede ayudarlo a encontrar dispositivos que aún sufren la vulnerabilidad Heartbleed parchada en abril de 2014.

Referencias

- 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
- Node Libraries Security Advisories
- [Ruby Libraries Security Advisory Database and Tools](#)
- https://safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf

Lista de CWEs mapeadas

CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities

CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

CWE-1104 Use of Unmaintained Third Party Components

A07:2021 – Fallas de Identificación y Autenticación



Factores

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
22	14.84%	2.55%	7.40	6.50	79.51%	45.72%	132,195	3,897

Resumen

Previamente denominada como *Pérdida de Autenticación*, descendió desde la segunda posición, y ahora incluye CWEs que están más relacionados con fallas de identificación. Las CWE notables incluidas son *CWE-297: Validación incorrecta de Certificado con discrepancia de host*, *CWE-287: Autenticación incorrecta* y *CWE-384: Fijación de sesiones*.

Descripción

La confirmación de la identidad, la autenticación y la gestión de sesiones del usuario son fundamentales para protegerse contra ataques relacionados con la autenticación. Puede haber debilidades de autenticación si la aplicación:

- Permite ataques automatizados como la reutilización de credenciales conocidas, donde el atacante posee una lista de pares de usuario y contraseña válidos.
- Permite ataques de fuerza bruta u otros ataques automatizados.
- Permite contraseñas por defecto, débiles o bien conocidas, como "Password1" o "admin/admin".
- Posee procesos débiles o no efectivos para las funcionalidades de olvido de contraseña o recuperación de credenciales, como "respuestas basadas en el conocimiento", las cuales no se pueden implementar de forma segura.
- Almacena las contraseñas en texto claro, cifradas o utilizando funciones de hash débiles (consulte [A02: 2021-Fallas Criptográficas](#)).
- No posee una autenticación multi-factor o la implementada es ineficaz.
- Expone el identificador de sesión en la URL.
- Reutiliza el identificador de sesión después de iniciar sesión.
- No inválida correctamente los ID de sesión. Las sesiones de usuario o los tokens de autenticación (principalmente tokens de inicio de sesión único (SSO)) no son correctamente invalidados durante el cierre de sesión o luego de un período de inactividad.

Cómo se previene

- Cuando sea posible, implemente la autenticación multi-factor para evitar ataques automatizados de reutilización de credenciales conocidas, fuerza bruta y reuso de credenciales robadas.
- No incluya o implemente en su software credenciales por defecto, particularmente para usuarios administradores.
- Implemente un control contra contraseñas débiles, tal como verificar que una nueva contraseña o la utilizada en el cambio de contraseña no esté incluida en la lista de las 10,000 peores contraseñas.
- Alinear las políticas de largo, complejidad y rotación de las contraseñas con las pautas de la sección 5.1.1 para Secretos Memorizados de la guía del NIST 800-63b u otras políticas de contraseñas modernas, basadas en evidencias.
- Asegúrese que el registro, la recuperación de las credenciales y el uso de APIs, no permiten los ataques de enumeración de usuarios, mediante la utilización de los mismos mensajes genéricos en todas las salidas.

- Limite o incremente el tiempo de espera entre intentos fallidos de inicio de sesión, pero tenga cuidado de no crear un escenario de denegación de servicio. Registre todos los fallos y avise a los administradores cuando se detecten ataques de rellenos automatizados de credenciales, fuerza bruta u otros.
- Utilice un gestor de sesión en el servidor, integrado, seguro y que genere un nuevo ID de sesión aleatorio con alta entropía después de iniciar sesión. Los identificadores de sesión no deben incluirse en la URL, deben almacenarse de forma segura y deben ser invalidados después del cierre de sesión, luego de un tiempo de inactividad o por un tiempo de espera absoluto.

Ejemplos de escenarios de ataque

Escenario #1: Relleno de credenciales, el uso de listas de contraseñas conocidas, es un ataque común. Supongamos que una aplicación no se implementa protección automatizada de relleno de credenciales. En ese caso, la aplicación puede usarse como oráculo de contraseñas para determinar si las credenciales son válidas.

Escenario #2: La mayoría de los ataques de autenticación ocurren debido al uso de contraseñas como único factor. Las consideradas mejores prácticas de requerir de una rotación y complejidad de las contraseñas, son vistos como alentadoras del uso y reúso de contraseñas débiles por parte de los usuarios. Se le recomienda a las organizaciones que detengan dichas prácticas y utilicen las prácticas recomendadas en la guía NIST 800-63 y utilicen autenticación multi-factor.

Escenario #3: Los tiempos de espera (timeouts) de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar "cerrar sesión", el usuario simplemente cierra la pestaña del navegador y se aleja. Un atacante usa el mismo navegador una hora más tarde, y el usuario continúa autenticado.

Referencias

- [OWASP Proactive Controls: Implement Digital Identity](#)
- [OWASP Application Security Verification Standard: V2 authentication](#)
- [OWASP Application Security Verification Standard: V3 Session Management](#)
- [OWASP Testing Guide: Identity, Authentication](#)
- [OWASP Cheat Sheet: Authentication](#)
- [OWASP Cheat Sheet: Credential Stuffing](#)
- [OWASP Cheat Sheet: Forgot Password](#)
- [OWASP Cheat Sheet: Session Management](#)
- [OWASP Automated Threats Handbook](#)
- NIST 800-63b: 5.1.1 Memorized Secrets

Lista de CWEs mapeadas

- [CWE-255 Credentials Management Errors](#)
- [CWE-259 Use of Hard-coded Password](#)
- [CWE-287 Improper Authentication](#)
- [CWE-288 Authentication Bypass Using an Alternate Path or Channel](#)
- [CWE-290 Authentication Bypass by Spoofing](#)
- [CWE-294 Authentication Bypass by Capture-replay](#)
- [CWE-295 Improper Certificate Validation](#)
- [CWE-297 Improper Validation of Certificate with Host Mismatch](#)
- [CWE-300 Channel Accessible by Non-Endpoint](#)
- [CWE-302 Authentication Bypass by Assumed-Immutable Data](#)
- [CWE-304 Missing Critical Step in Authentication](#)
- [CWE-306 Missing Authentication for Critical Function](#)
- [CWE-307 Improper Restriction of Excessive Authentication Attempts](#)
- [CWE-346 Origin Validation Error](#)
- [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-1216 Lockout Mechanism Errors](#)

A08:2021 – Fallas en el Software y en la Integridad de los Datos



Factores

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
10	16.67%	2.05%	6.94	7.94	75.04%	45.35%	47,972	1,152

Resumen

Una nueva categoría en la versión 2021 que se centra en hacer suposiciones relacionadas con las actualizaciones de software, los datos críticos y los pipelines de CI/CD sin verificación de integridad. Corresponde a uno de los mayores impactos según los sistemas de ponderación de vulnerabilidades (CVE/CVSS, siglas en inglés para Common Vulnerability and Exposures/Common Vulnerability Scoring System). Entre estos, se destacan las siguientes CWEs: *CWE-829: Inclusión de funcionalidades provenientes de fuera de la zona de confianza*, *CWE-494: Ausencia de verificación de integridad en el código descargado*, y *CWE-502: Deserialización de datos no confiables*.

Descripción

Los fallos de integridad del software y de los datos están relacionados con código e infraestructura no protegidos contra alteraciones (integridad). Ejemplos de esto son cuando una aplicación depende de plugins, bibliotecas o módulos de fuentes, repositorios o redes de entrega de contenidos (CDN) no confiables. Un pipeline CI/CD inseguro puede conducir a accesos no autorizados, la inclusión de código malicioso o el compromiso del sistema en general. Además, es común en la actualidad que las aplicaciones implementen funcionalidades de actualización, a través de las cuales se descargan nuevas versiones de la misma sin las debidas verificaciones integridad que fueron realizadas previamente al instalar la aplicación. Los atacantes potencialmente pueden cargar sus propias actualizaciones para que sean distribuidas y ejecutadas en todas las instalaciones. Otro ejemplo es cuando objetos o datos son codificados o serializados en estructuras que un atacante puede ver y modificar, produciéndose una deserialización insegura.

Cómo se previene

- Utilice firmas digitales o mecanismos similares para verificar que el software o datos provienen efectivamente de la fuente esperada y no fueron alterados.
- Asegúrese que las bibliotecas y dependencias, tales como npm o maven son utilizadas desde repositorios confiables. Si su perfil de riesgo es alto, considere alojarlas en un repositorio interno cuyo contenido ha sido previamente analizado.
- Asegúrese que se utilice una herramienta de análisis de componentes de terceros, como OWASP Dependency Check u OWASP CycloneDX, con el fin de verificar la ausencia de vulnerabilidades conocidas.
- Asegúrese que se utilice un proceso de revisión de cambios de código y configuraciones para minimizar las posibilidades de que código o configuraciones maliciosas sean introducidos en su pipeline.
- Asegúrese que su pipeline CI/CD posee adecuados controles de acceso, segregación y configuraciones que permitan asegurar la integridad del código a través del proceso de build y despliegue.
- Asegúrese que datos sin cifrar o firmar no son enviados a clientes no confiables sin alguna forma de verificación de integridad o firma electrónica con el fin de detectar modificaciones o la reutilización de datos previamente serializados.

Ejemplos de escenarios de ataque

Escenario #1 Actualizaciones no firmadas: Muchos routers domésticos, decodificadores de televisión, firmware de dispositivos, entre otros, no verifican las firmas de sus actualizaciones de firmware. El firmware sin firmar es un objetivo creciente para los atacantes y se espera que empeore. Esto es una gran preocupación, ya que muchas veces no existe otro mecanismo para remediarlo que corregirlo en una versión futura y esperar a que las versiones anteriores caduquen.

Escenario #2 Actualización maliciosa de SolarWinds: Se sabe que los Estados-Naciones utilizan como vector de ataque los mecanismos de actualización, siendo un caso reciente de pública notoriedad el sufrido por SolarWinds Orion. La compañía que desarrolla el software poseía procesos seguros de construcción y mecanismos de integridad en sus actualizaciones. Sin embargo, estos fueron comprometidos y, durante varios meses, la firma distribuyó una actualización maliciosa a más de 18.000 organizaciones, de las cuales alrededor de un centenar se vieron afectadas. Se trata de una de las brechas de este tipo de mayor alcance y más importantes de la historia.

Escenario #3 Deserialización insegura: Una aplicación React utiliza un conjunto de microservicios implementados en Spring Boot. Tratándose de programadores funcionales, intentaron asegurarse de que su código sea inmutable. La solución implementada consistió en serializar el estado de la sesión para el usuario y enviarlo entre los componentes con cada solicitud. Un atacante advierte el uso de un objeto Java serializado y codificado en base64 (identifica un string que comienza con "rO0") y utiliza la herramienta Java Serial Killer para obtener una ejecución remota de código en el servidor de aplicación.

Referencias

- *OWASPCheatSheet : Software Supply Chain Security*
(Coming Soon)
- *OWASPCheatSheet : Secure build and deployment*
(Coming Soon)
- [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](#)

Lista de CWEs mapeadas

[CWE-345 Insufficient Verification of Data Authenticity](#)

[CWE-353 Missing Support for Integrity Check](#)

[CWE-426 Untrusted Search Path](#)

[CWE-494 Download of Code Without Integrity Check](#)

[CWE-502 Deserialization of Untrusted Data](#)

[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](#)

A09:2021 – Fallas en el Registro y Monitoreo

Factores



CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
4	19.23%	6.51%	6.87	4.99	53.67%	39.97%	53,615	242

Resumen

Monitoreo y registro de seguridad provienen de la encuesta de la comunidad TOP 10, subió levemente desde la décima posición en el OWASP Top 10 2017. El registro y monitoreo pueden ser desafiantes para ser testeados, a menudo implica entrevistas o preguntas si los ataques fueron detectados durante una prueba de penetración. No hay muchos datos de CVEs para esta categoría, pero detectar y responder a las brechas es crítico. Aun así, puede tener un gran impacto para la responsabilidad, visibilidad, alertas de incidentes y forense. Esta categoría se expande más allá de [CWE-117 Neutralización de salida incorrecta para registros](#), [CWE-223 Omisión de información relevante para la seguridad](#), y [CWE-532 Inserción de información sensible en el archivo de registro](#).

Descripción

Volviendo al OWASP Top 10 2021, la intención es detectar, escalar y responder ante brechas activas. Sin registros y monitoreo, las brechas no pueden ser detectadas. Registros, detecciones, monitoreo y respuestas activas insuficientes pueden ocurrir en cualquier momento:

- Eventos auditables, tales como los inicios de sesión, fallas en el inicio de sesión y transacciones de alto valor, no son registradas.
- Advertencias y errores generan registros poco claros, inadecuados y en algunos casos ni se generan.
- Registros en aplicaciones y API no son monitoreados para detectar actividades sospechosas.
- Los registros son únicamente almacenados en forma local.
- Los umbrales de alerta y procesos de escalamiento no están correctamente implementados o no son efectivos.
- Las pruebas de penetración y los escaneos utilizando herramientas de pruebas dinámicas de seguridad en aplicaciones (como ser OWASP ZAP) no generan alertas.

- Las aplicaciones no logran detectar, escalar, o alertar sobre ataques activos en tiempo real ni cercanos al tiempo real.

Se es vulnerable a la fuga de información haciendo registros y eventos de alertas que sean visibles para un usuario o un atacante (consulte [A01: 2021-Pérdida de Control de Acceso](#)).

Cómo se previene

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

- Asegúrese de que todos los errores de inicio de sesión, de control de acceso y de validación de entradas de datos del lado del servidor se pueden registrar con suficiente contexto como para identificar cuentas sospechosas o maliciosas y mantenerlo durante el tiempo suficiente para permitir un posterior análisis forense.
- Asegúrese de que los registros se generen en un formato fácil de procesar por las herramientas de gestión de registros.
- Asegúrese de que los datos de registros estén codificados correctamente para prevenir inyecciones o ataques en el sistema de monitoreo o registros.
- Asegúrese de que las transacciones de alto valor poseen una traza de auditoria con controles de integridad para evitar la modificación o el borrado, tales como permitir únicamente la inserción en las tablas de base de datos o similares.
- El equipo de DevSecOps debe establecer alertas y monitoreo efectivo tal que se detecte actividades sospechosas y responderlas rápidamente.
- Establecer o adoptar un plan de respuesta y recuperación, tal como NIST 800-61r2 o posterior.

Existen frameworks de protección de aplicaciones comerciales y de código abierto, tales como el conjunto de reglas de ModSecurity de OWASP y el conjunto de programas de correlación de registros de código abierto como ser ELK (Elasticsearch, Logstash, Kibana) con paneles personalizados y alertas.

Ejemplos de escenarios de ataque

Escenario #1: Un operador de salud que provea un plan de salud para niños no pudieron detectar una brecha debido a la falta de monitoreo y registro. Alguien externo informó al proveedor de salud que un atacante había accedido y modificados miles de registros médicos sensibles de más de 3.5 millones de niños. Una revisión post incidente encontró que los desarrolladores del sitio web no habían encontrado vulnerabilidades significativas. Como no hubo ni registro ni monitores del sistema, la brecha de datos pudo haber estado en proceso desde el 2013, un período de más de 7 años.

Escenario #2: Una gran aerolínea India tuvo una brecha de seguridad que involucró a la pérdida de datos personales de millones de pasajeros por más de 10 años, incluyendo pasaportes y tarjetas de crédito. La brecha se produjo por un proveedor de servicios de almacenamiento en la nube, quien notificó a la aerolínea después de un cierto tiempo.

Escenario #3: Una gran aerolínea Europea sufrió un incumplimiento de la GRPD. Se reporta que la causa de la brecha se debió a que un atacante explotó una vulnerabilidad en una aplicación de pago, obteniendo más de 400,000 registros de pagos de usuarios. La aerolínea fue multada con 20 millones de libras como resultado del regulador de privacidad.

Referencias

- [OWASP Proactive Controls: Implement Logging and Monitoring](#)
- [OWASP Application Security Verification Standard: V8 Logging and Monitoring](#)
- [OWASP Testing Guide: Testing for Detailed Error Code](#)
- [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](#)

Lista de CWEs mapeadas

[CWE-117 Improper Output Neutralization for Logs](#)

[CWE-223 Omission of Security-relevant Information](#)

[CWE-532 Insertion of Sensitive Information into Log File](#)

[CWE-778 Insufficient Logging](#)

A10:2021 – Falsificación de Solicitudes del Lado del Servidor (SSRF)

Factores



CWEs	Tasa de	Tasa de incidencia	Explotabilidad	Impacto ponderado	Cobertura	Cobertura	Incidencias	Total
------	---------	--------------------	----------------	-------------------	-----------	-----------	-------------	-------

mapeadas	incidencia máx	prom	ponderada prom	prom	máx	prom	totales	CVEs
1	2.72%	2.72%	8.28	6.72	67.72%	67.72%	9,503	385

Resumen

Esta categoría se agrega de la encuesta de la comunidad Top 10(#1). Los datos muestran una tasa de incidencia relativamente baja con una cobertura de pruebas por encima del promedio y calificaciones de potencial de Explotación e Impacto por encima del promedio. Como es probable que estas nuevas entradas sean una única o un pequeño grupo de Enumeraciones de debilidades comunes (CWE) para tomar en cuenta y concientizar sobre ellas, la esperanza es que se enfoque la atención en ellas y puedan integrarse en una categoría más grande en una edición futura.

Descripción

Las fallas de SSRF ocurren cuando una aplicación web está obteniendo un recurso remoto sin validar la URL proporcionada por el usuario. Permite que un atacante coaccione a la aplicación para que envíe una solicitud falsificada a un destino inesperado, incluso cuando está protegido por un firewall, VPN u otro tipo de lista de control de acceso a la red (ACL).

Dado que las aplicaciones web modernas brindan a los usuarios finales funciones convenientes, la búsqueda de una URL se convierte en un escenario común. Como resultado, la incidencia de SSRF está aumentando. Además, la gravedad de SSRF es cada vez mayor debido a los servicios en la nube y la complejidad de las arquitecturas.

Cómo se previene

Los desarrolladores pueden prevenir SSRF implementando algunos o todos los siguientes controles de defensa en profundidad:

Desde la capa de red

- Segmenta la funcionalidad de acceso a recursos remotos en redes separadas para reducir el impacto de SSRF
- Haga cumplir las políticas de firewall "denegar por defecto" o las reglas de control de acceso a la red para bloquear todo el tráfico de la intranet excepto el esencial.
Consejos:
 - ~ Establezca la propiedad y un ciclo de vida para las reglas de firewall basadas en aplicaciones.
 - ~ Registre en logs todos los flujos de red aceptados y bloqueados en firewalls (consulte [A09: 2021-Fallas en el Registro y Monitoreo](#)).

Desde la capa de aplicación:

- Sanitize y valide todos los datos de entrada proporcionados por el cliente
- Haga cumplir el esquema de URL, el puerto y el destino con una lista positiva de ítems permitidos
- No envíe respuestas en formato "crudo" a los clientes
- Deshabilite las redirecciones HTTP
- Tenga en cuenta la coherencia de la URL para evitar ataques como el enlace de DNS y las condiciones de carrera de "tiempo de verificación, tiempo de uso" (TOCTOU)

No mitigue SSRF mediante el uso de una lista de denegación o una expresión regular. Los atacantes tienen listas de payloads, herramientas y habilidades para eludir las listas de denegación.

Medidas adicionales a considerar:

- No implemente otros servicios relevantes para la seguridad en los sistemas frontales (por ejemplo, OpenID). Controle el tráfico local en estos sistemas (por ejemplo, localhost)
- Para frontends con grupos de usuarios dedicados y manejables, use el cifrado de red (por ejemplo, VPN) en sistemas independientes para considerar necesidades de protección muy altas

Ejemplos de escenarios de ataque

Los atacantes pueden usar SSRF para atacar sistemas protegidos detrás de firewalls de aplicaciones web, firewalls o ACLs de red, utilizando escenarios tales como:

Escenario #1: Escaneo de puertos de servidores internos – Si la arquitectura de la red no está segmentada, los atacantes pueden trazar un mapa de las redes internas y determinar si los puertos están abiertos o cerrados en los servidores internos a partir de los resultados de la conexión o del tiempo transcurrido para conectar o rechazar las conexiones de payload SSRF.

Escenario #2: Exposición de datos sensibles: los atacantes pueden acceder a archivos locales como servicios internos para obtener información confidencial como `file:///etc/passwd` y `http://localhost:28017/`.

Escenario #3: Acceso al almacenamiento de metadatos de los servicios en la nube: la mayoría de los proveedores de la nube tienen almacenamiento de metadatos como `http://169.254.169.254/`. Un atacante puede leer los metadatos para obtener información confidencial.

Escenario #4: Exposición de los servicios internos: el atacante puede abusar de los servicios internos para realizar más ataques, como la ejecución remota de código (RCE) o la denegación de servicio (DoS).

Referencias

- [OWASP - Server-Side Request Forgery Prevention Cheat Sheet](#)
- [PortSwigger - Server-side request forgery \(SSRF\)](#)
- [Acunetix - What is Server-Side Request Forgery \(SSRF\)?](#)
- [SSRF bible](#)
- [A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages!](#)

Lista de CWEs mapeadas

[CWE-918 Server-Side Request Forgery \(SSRF\)](#)

A11:2021 – Siguientes pasos

Por diseño, el Top 10 de OWASP se limita de forma innata a los diez riesgos más importantes. Cada Top 10 de OWASP tuvo riesgos "en el umbral" considerados detenidamente para su inclusión, pero al final, no fueron incluidos. No importando cuanto intentáramos interpretar o tergiversar los datos, los otros riesgos fueron más frecuentes e impactantes.

Para aquellas organizaciones que trabajan en pos de un programa de appsec maduro o consultores de seguridad o proveedores de herramientas que deseen ampliar la cobertura de sus ofertas, vale la pena el esfuerzo de identificar y solucionar los siguientes tres problemas.

Problemas de calidad de código

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Total de Incidencias	Total de CVEs
38	49.46%	2.22%	7.1	6.7	60.85%	23.42%	101736	7564

- **Descripción.** Los problemas de calidad del código incluyen patrones o defectos de seguridad conocidos, reutilización de variables para múltiples propósitos, exposición de información sensible en la salida de depuración, errores uno por uno, condiciones de carrera de tiempo de verificación/tiempo de uso(TOC/TOU), errores de conversión firmados o no firmados, uso de memoria liberada y más. El sello distintivo de esta sección es que generalmente se pueden identificar con estrictas marcas de compilación, herramientas de análisis de código estático y complementos de IDE para análisis de tipo lint. Los lenguajes modernos por diseño eliminaron muchos de estos problemas, como la propiedad de la memoria de Rust y el concepto de préstamo, el diseño de hilos de subprocesso de Rust y la tipificación estricta y la verificación de límites de Go.
- **Cómo se previene.** Habilite y use las opciones de análisis de código estático de su editor específicos para cada lenguaje. Considere el uso de una herramienta de análisis de código estático. Considere si fuera posible usar o migrar a un lenguaje o marco que elimine este tipo de errores, como Rust o Go.
- **Ejemplos de escenarios de ataque.** Un atacante puede obtener o actualizar información confidencial aprovechando una condición de carrera utilizando una variable compartida estáticamente en varios subprocessos.
- **Referencias**
- [OWASP Code Review Guide](#)
- [Google Code Review Guide](#)

Denegación de servicio

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Explotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidentes totales	Total CVEs
8	17.54%	4.89%	8.3	5.9	79.58%	33.26%	66985	973

- **Descripción.** La denegación de servicio siempre es posible con suficientes recursos. Sin embargo, las prácticas de diseño y codificación tienen una influencia significativa en la magnitud de la denegación de servicio. Supongamos que cualquier persona con el enlace puede acceder a un archivo grande, o que se produce una transacción computacionalmente costosa en cada página. En ese caso, la denegación de servicio requiere menos esfuerzo para llevarse a cabo.
- **Cómo se previene.** Realizar prueba de rendimiento para el uso de CPU, E/S y memoria, rediseñar, optimizar o almacenar en caché las operaciones pesadas. Considere los controles de acceso para objetos más grandes para asegurarse de que solo las personas autorizadas puedan acceder a archivos u objetos grandes o servirlos a través de una red de almacenamiento en caché perimetral.
- **Ejemplos de escenarios de ataque.** Un atacante podría determinar que una operación tarda entre 5 y 10 segundos en completarse. Cuando se ejecutan cuatro subprocessos simultáneos, el servidor parece dejar de responder. El atacante entonces usa 1000 subprocessos y saca de servicio todo el sistema.
- **Referencias**
- [OWASP Cheet Sheet: Denial of Service](#)
- [OWASP Attacks: Denial of Service](#)

Errores de administración de memoria

CWEs mapeadas	Tasa de incidencia máx	Tasa de incidencia prom	Eplotabilidad ponderada prom	Impacto ponderado prom	Cobertura máx	Cobertura prom	Incidencias totales	Total CVEs
14	7.03%	1.16%	6.7	8.1	56.06%	31.74%	26576	16184

- **Descripción.** Las aplicaciones web usualmente se escriben en lenguajes de memoria administrada, como Java, .NET o node.js (JavaScript o TypeScript). Sin embargo, estos lenguajes están escritos en lenguajes de sistema que tienen problemas de administración de memoria, como desbordamientos de búfer o de heap, uso de memoria liberada, desbordamiento de números enteros y más. A lo largo de los años, ha habido muchos escapes de espacio aislado(sandbox escapes) que demuestran que aunque el lenguaje de la aplicación web es nominalmente "seguro" para la memoria, la estructura de base no siempre es segura.
 - **Cómo se previene.** Muchas API modernas ahora están escritas en lenguajes seguros para la memoria como Rust o Go. En el caso de Rust, la seguridad de la memoria es una característica crucial del lenguaje. Para el código existente, el uso de banderas de compilador estrictas, fuerte tipado, análisis de código estático y pruebas de fuzz puede ser beneficioso para identificar pérdidas de memoria, desbordamientos de matrices y memoria, y más.
 - **Ejemplos de escenarios de ataque.** Los desbordamientos de búfer y heap han sido un pilar de los atacantes a lo largo de los años. El atacante envía datos a un programa, que almacena en un búfer de pila de tamaño insuficiente. El resultado es que se sobrescribe la información de la pila de llamadas, incluido el puntero de retorno de la función. Los datos establecen el valor del puntero de retorno para que cuando la función regrese, transfiera el control al código malicioso contenido en los datos del atacante.
- **Referencias**
- [OWASP Vulnerabilities: Buffer Overflow](#)
 - [OWASP Attacks: Buffer Overflow](#)
 - [Science Direct: Integer Overflow](#)