

# Seguridad en la Red Informática Mundial

## Top-Ten Vulnerabilidades según OWASP

Semana 14 clases 27 y 28

Mtra. María Noemí Araiza Ramírez

---

# Top-Ten Vulnerabilidades según OWASP



## Objetivos:

¿Qué es OWASP Top 10?

¿Qué objetivos y motivaciones tiene?

¿Qué ha cambiado de 2013 a 2017?

¿Qué son los Riesgos de Seguridad de Aplicaciones?

¿Conocer cómo me afectan o cuál es mi riesgo?

¿Conocer los 10 Riesgos más importantes?

---

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

# Top-Ten Vulnerabilidades según OWASP



## ¿Cuál es mi Riesgo?

Top 10 2013

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	

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

Top 10 2017

A8

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

Recordemos que está ubicada en la tabla de amenazas de 2013, para 2017 esta amenaza desaparece y se integra una nueva.

A8  
:2017

Deserialización Insegura

---

# Top-Ten Vulnerabilidades según OWASP



**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.

## Top-Ten Vulnerabilidades según OWASP

A8

### Falsificación de Peticiones en Sitios Cruzados (CSRF)


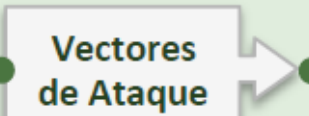
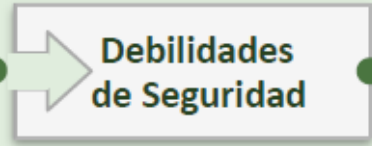
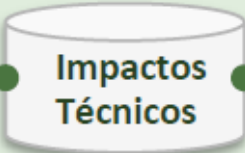
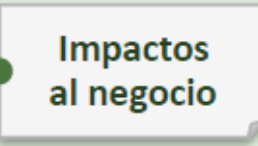




# Top-Ten Vulnerabilidades según OWASP

**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 <b>PROMEDIO</b>	Prevalencia <b>COMÚN</b>	Detección <b>FÁCIL</b>	Impacto <b>MODERADO</b>	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)

### ¿Soy vulnerable?

La forma más sencilla de revisar la vulnerabilidad en una aplicación, es verificando si cada enlace, y formulario, contiene un testigo token no predecible para cada usuario.

Si no se tiene dicho testigo, los atacantes pueden falsificar peticiones. Se debe concentrar el análisis en enlaces y formularios que invoquen funciones que permitan cambiar estados.

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### ¿Soy vulnerable?

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).

Se debe verificar transacciones que involucren múltiples pasos. Los atacantes pueden falsificar una serie de peticiones a través de múltiples etiquetas o posiblemente código JavaScript.

---

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### ¿Soy vulnerable?

Descartar como protección las cookies de sesión, las direcciones IP de la fuente y otro tipo de información, ya que está se encuentra incluida en las peticiones falsas.

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

---

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### ¿Cómo se puede evitar?

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

Tenemos tres alternativas:

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

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### ¿Cómo se puede evitar?

El Guardián CSRF de la OWASP, puede ser utilizado para incluir automáticamente los testigos en aplicaciones Java EE, NET o PHP.

La API ES de la OWASP, incluye generadores y validadores de testigos que los realizadores de software pueden usar para proteger sus transacciones.

---

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### ¿Cómo se puede evitar?

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.

**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### Ejemplos de escenarios de ataques

Escenario 1: La aplicación permite que los usuarios envíen peticiones de cambio de estado, que no incluyen nada secreto.

Por ejemplo:

[http://example.com/app/transferFunds?  
amount=1500&destinationAccount=4673243243](http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243)

El atacante puede construir una petición que transfiera dinero desde la cuenta de la víctima a su propia cuenta.

Podrá insertar su ataque dentro de una etiqueta de imagen o iframe en un sitio web que esté bajo su control, de la siguiente forma:

---



**A8**

## Falsificación de Peticiones en Sitios Cruzados (CSRF)

### Ejemplos de escenarios de ataques

```

```

Si la víctima visita alguno de los sitios controlados por el atacante, estando ya autenticado en [www ejemplo.com](http://www.ejemplo.com), en lugar de cargarse la imagen, se realizará la petición HTTP falsificada.

Aparte se incluirá automáticamente la información de la sesión del usuario, autorizando la petición del atacante entonces el ataque será exitoso.

---

**A8**  
:2017

# Deserialización Insegura

**A8**  
:2017

## Deserialización Insegura

### ¿Qué es la deserialización?

La serialización es el proceso de convertir algún objeto en un formato de datos que se puede restaurar más tarde.

Las personas a menudo serializan objetos para guardarlos en el almacenamiento o enviarlos como parte de las comunicaciones.

La deserialización es el reverso de ese proceso, tomando datos estructurados de algún formato y reconstruyéndolos en un objeto.

Hoy, el formato de datos más popular para serializar datos es JSON. Antes de eso, era XML.

---

**A8**  
:2017

## Deserialización Insegura

Sin embargo, muchos lenguajes de programación ofrecen una capacidad nativa para serializar objetos.

Estos formatos nativos generalmente ofrecen más funciones que JSON o XML, incluida la personalización del proceso de serialización.

Desafortunadamente, las características de estos mecanismos de deserialización nativos se pueden reutilizar para un efecto malicioso cuando se opera con datos no confiables.

Se ha descubierto que los ataques contra deserializadores permiten ataques de denegación de servicio, control de acceso y ejecución remota de código (RCE).

---

**A8**  
:2017

## Deserialización Insegura

Es una vulnerabilidad de difícil explotación.

La serialización es un concepto que implica convertir datos de un formato a otro formato concreto (por ejemplo de un formato admitido a formato XML) que permita su transmisión o guardado.

No aceptar datos serializados de fuentes no confiables o sólo aceptar para su serialización a datos de tipos primitivos, son dos buenas prácticas que nuestra arquitectura debería cumplir en todos los casos.

Esta vulnerabilidad podría permitir la ejecución remota de esos códigos serializados.

---

# Top-Ten Vulnerabilidades según OWASP



**A8**  
:2017

## Deserialización Insegura

Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución.

En el peor de los casos, la deserialización insegura puede conducir a la ejecución remota de código en el servidor.

# Top-Ten Vulnerabilidades según OWASP

**A8**  
:2017

## Deserialización Insegura

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



**A8**  
:2017

## Deserialización Insegura

### ¿La aplicación es vulnerable?

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

Esto da como resultado dos tipos primarios de ataques:

- Ataques relacionados con la estructura de datos y objetos; donde el atacante modifica la lógica de la aplicación o logra una ejecución remota de código que puede cambiar el comportamiento de la aplicación durante o después de la deserialización.

**A8**  
:2017

## Deserialización Insegura

### ¿La aplicación es vulnerable?

- Ataques típicos de manipulación de datos; como ataques relacionados con el control de acceso, en los que se utilizan estructuras de datos existentes pero se modifica su contenido.

La serialización puede ser utilizada en aplicaciones para:

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

**A8**  
:2017

## Deserialización Insegura

### ¿Cómo se previene?

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

Si esto no es posible, considere alguno de los siguientes puntos:

- Implemente verificaciones de integridad tales como firmas digitales en cualquier objeto serializado, con el fin de detectar modificaciones no autorizadas.

**A8**  
:2017

## Deserialización Insegura

### ¿Cómo se previene?

Durante la deserialización y antes de la creación del objeto, exija el cumplimiento estricto de verificaciones de tipo de dato, ya que el código normalmente espera un conjunto de clases definibles.

Se ha demostrado que se puede pasar por alto esta técnica, por lo que no es aconsejable confiar sólo en ella.

- Aísle el código que realiza la deserialización, de modo que se ejecute en un entorno con los mínimos privilegios posibles.
-

**A8**  
:2017

## Deserialización Insegura

### ¿Cómo se previene?

- Registre las excepciones y fallas en la deserialización, tales como cuando el tipo recibido no es el esperado, o la deserialización produce algún tipo de error.
- Restrinja y monitoree las conexiones (I/O) de red desde contenedores o servidores que utilizan funcionalidades de deserialización.
- Monitoree los procesos de deserialización, alertando si un usuario deserializa constantemente.

**A8**  
:2017

## Deserialización Insegura

### Ejemplos de escenarios de ataques

Escenario 1: Una aplicación React invoca a un conjunto de microservicios Spring Boot.

Siendo programadores funcionales, intentaron asegurar que su código sea inmutable. La solución a la que llegaron es serializar el estado del usuario y pasarlo en ambos sentidos con cada solicitud.

Un atacante advierte la firma “Roo” del objeto Java, y usa la herramienta Java Serial Killer para obtener ejecución de código remoto en el servidor de la aplicación.

---

**A8**  
:2017

## Deserialización Insegura

### Ejemplos de escenarios de ataques (Top 10 2017)

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

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

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

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

---



**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

Recordemos que está ubicada en la tabla de amenazas de 2013, para 2017 esta amenaza continua en el mismo lugar 9.

**A9**  
:2017

Uso de Componentes con Vulnerabilidades Conocidas

---

# Top-Ten Vulnerabilidades según OWASP



**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

Es una vulnerabilidad a través de la cual el atacante identifica o conoce mediante escaneos o análisis manuales, los componentes débiles de una aplicación web como los frameworks y ejecuta el ataque.

# Top-Ten Vulnerabilidades según OWASP



**A9**

## 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.

# Top-Ten Vulnerabilidades según OWASP



**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

**A9**  
:2017

## Uso de Componentes con Vulnerabilidades Conocidas

Los componentes como bibliotecas, frameworks y otros módulos se ejecutan con los mismos privilegios que la aplicación.




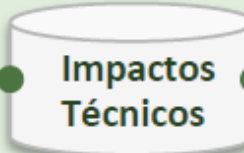

Si se explota un componente vulnerable, el ataque puede provocar una pérdida de datos o tomar el control del servidor.

Las aplicaciones y API que utilizan componentes con vulnerabilidades conocidas pueden debilitar las defensas de las aplicaciones y permitir diversos ataques e impactos.

# Top-Ten Vulnerabilidades según OWASP

**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 <b>PROMEDIO</b>	Prevalencia <b>DIFUNDIDO</b>	Detectabilidad <b>DIFÍCIL</b>	Impacto <b>MODERADO</b>	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.

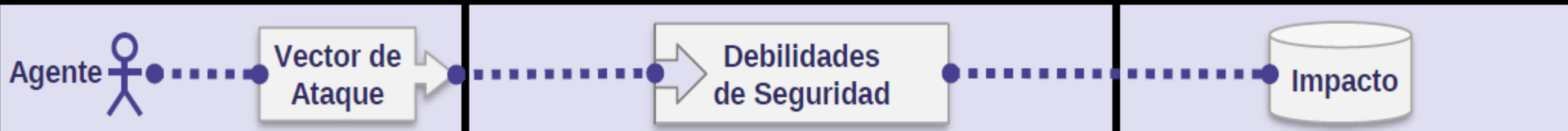
# Top-Ten Vulnerabilidades según OWASP

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

**A9**  
:2017

Uso de Componentes con  
Vulnerabilidades Conocidas

					
App. Específica	Explotabilidad: 2	Prevalencia: 3	Detectabilidad: 2	Técnico: 2	¿Negocio?
<p>Es sencillo obtener <i>exploits</i> para vulnerabilidades ya conocidas pero la explotación de otras requieren un esfuerzo considerable, para su desarrollo y/o personalización.</p>		<p>Estos defectos están muy difundidos. El desarrollo basado fuertemente en componentes de terceros, puede llevar a que los desarrolladores no entiendan qué componentes se utilizan en la aplicación o API y, mucho menos, mantenerlos actualizados. Esta debilidad es detectable mediante el uso de analizadores tales como <a href="https://retire.js.org/">retire.js</a> o la inspección de cabeceras. La verificación de su explotación requiere de la descripción de un posible ataque.</p>		<p>Mientras que ciertas vulnerabilidades conocidas conllevan impactos menores, algunas de las mayores brechas registradas han sido realizadas explotando vulnerabilidades conocidas en componentes comunes. Dependiendo del activo que se está protegiendo, este riesgo puede ser incluso el principal de la lista.</p>	

**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

### ¿Soy vulnerable?

Teóricamente se dice que debiera ser fácil distinguir si se está usando un componente o biblioteca vulnerable.

Desafortunadamente, los reportes de vulnerabilidades para soNware comercial o de código abierto no siempre especifica exactamente que versión de un componente es vulnerable en un estándar, de forma accesible.

Además, no todas las bibliotecas usan un sistema numérico de versiones entendible.

---



# Top-Ten Vulnerabilidades según OWASP



**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## ¿Soy vulnerable?

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 es necesario buscar en estas bases de datos, también mantenerse al tanto de la lista de correos del proyecto y anuncios de cualquier cosa que nos de indicio de 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.

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## ¿La aplicación es vulnerable? (Top 10 2017)

Es potencialmente vulnerable si:

- No conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del servidor). Esto incluye componentes utilizados directamente como sus dependencias anidadas.
- El software es vulnerable, no posee soporte o se encuentra desactualizado. Esto incluye el sistema operativo, servidor web o de aplicaciones, DBMS, APIs y todos los componentes, ambientes de ejecución y bibliotecas.

**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

### ¿La aplicación es vulnerable? (Top 10 2017)

- No se analizan los componentes periódicamente ni se realiza seguimiento de los boletines de seguridad de los componentes utilizados.
  - No se parchea o actualiza la plataforma subyacente, frameworks y dependencias, con un enfoque basado en riesgos. Esto sucede comúnmente en ambientes en los cuales la aplicación de parches se realiza de forma mensual o trimestral bajo control de cambios, lo que deja a la organización abierta innecesariamente a varios días o meses de exposición a vulnerabilidades ya solucionadas.
  - No asegura la configuración de los componentes correctamente.
-

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## ¿Cómo se puede evitar?

Una opción es no usar componentes que no ha codificado, sin embargo esto no suena muy realista y 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 soNware debieran tener un proceso para:

---

**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

### ¿Cómo se puede evitar?

1. Identificar todos los componentes y la versión que están utilizando, incluyendo dependencias.
  2. Revisar la seguridad del componente en bases de datos públicas, lista de correos del proyecto y listas de correo de seguridad, y mantenerlos actualizados.
  3. Establecer políticas de seguridad que regulen el uso de componentes, como requerir 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
-

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## ¿Cómo se previene? (Top 10 2017)

Remover dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada.

- Utilizar una herramienta para mantener un inventario de versiones de componentes (por ej. frameworks o bibliotecas) tanto del cliente como del servidor. Por ejemplo, Dependency Check y retire.js.

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## ¿Cómo se previene? (Top 10 2017)

- Monitorizar continuamente fuentes como CVE y NVD en búsqueda de vulnerabilidades en los componentes utilizados. Utilizar herramientas de análisis automatizados. Suscribirse a alertas de seguridad de los componentes utilizados.
  - Obtener componentes únicamente de orígenes oficiales utilizando canales seguros. Utilizar preferentemente paquetes firmados con el fin de reducir las probabilidades de uso de versiones manipuladas maliciosamente.
-

**A9**

## Uso de Componentes con Vulnerabilidades Conocidas

### ¿Cómo se previene? (Top 10 2017)

- Supervisar bibliotecas y componentes que no poseen mantenimiento o no liberan parches de seguridad para sus versiones obsoletas o sin soporte. Si el parcheo no es posible, considere desplegar un parche virtual para monitorizar, detectar o protegerse contra la debilidad detectada.

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

---



**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## 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, por tanto, cualquier falla en un componente puede ser serio.

Los siguientes componentes vulnerables fueron descargados 22 millones de veces en el año 2011:

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios de ataques

Escenario 1: Apache CSF Authentication Bypass es un framework de servicios (no el servidor de aplicaciones Apache) que no otorgaba un token de identidad.

De este modo, los atacantes podían invocar cualquier servicio web con todos los permisos.

Nota: es importante no confundir Apache CXF con el servidor de aplicaciones de Apache, ya que este es un framework de servicios.

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios de ataques

### Escenario 2: Spring Remote Code Execution

El abuso de la implementación en Spring del componente « Expression Lenguaje » permitió a los atacantes ejecutar código arbitrario, tomando el control del servidor.

Esto es que, cualquier aplicación que utilice cualquiera de esas bibliotecas vulnerables es susceptible de ataques.

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios 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.

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios de ataques (Top 10 2017)

Escenario 1: Típicamente, los componentes se ejecutan con los mismos privilegios de la aplicación que los contienen y, como consecuencia, fallas en éstos pueden resultar en impactos serios.

Estas fallas pueden ser accidentales (por ejemplo, errores de codificación) o intencionales (una puerta trasera en un componente).

Algunos ejemplos de vulnerabilidades en componentes explotables son:

---

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios de ataques (Top 10 2017)

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

**A9**

Uso de Componentes con  
Vulnerabilidades Conocidas

## Ejemplos de escenarios de ataques (Top 10 2017)

Existen herramientas automáticas que ayudan a los atacantes a descubrir sistemas mal configurados o desactualizados.

A modo de ejemplo, el motor de búsqueda Shodan ayuda a descubrir dispositivos que aún son vulnerables a Heartbleed, la cual fue parcheada en abril del 2014.

---

## Referencias

<https://www.owasp.org>