Desarrollo seguro de Aplicaciones Web basado en OWASP

Por: Carlos Carreño,

OCP, ScrumMaster, Solution Architect

Email: ccarrenovi@Gmail.com

Unidad 4 Seguridad en el diseño de software

- Criterios Básicos de Seguridad
- Prevención de divulgación de información
- Manejo de información sensible
- Almacenamiento seguro
- Transferencia segura
- Encriptación y Hashes

Autor: ccarrenovi@Gmail.com

… continua

- Auditoría y Logging
- Diseño de Autenticación y Autorización
- Seguridad en Web Services
- Diseño de protección contra Denial of Service (D.O.S)
- Errores de Lógica de negocio

Criterios Básicos de Seguridad

- Principio del menor privilegio
- Criterio de defensa en profundidad
- Manejo seguro de errores
- Criterio del “Fallo Seguro”
- Definición de mensajes de error

Principio del menor privilegio

- El principio del mínimo privilegio recomienda que las cuentas tengan la mínima cantidad de privilegios necesarios para realizar sus procesos de negocio. Esto abarca a los derechos de usuario, permisos de recursos tales como límites de CPU, memoria, red y permisos del sistema de archivos.

Ejemplo, si un servidor middleware requiere acceso sólo a la red, acceso de lectura a la tabla de una base de datos, y la habilidad para escribir en un log, esto describe todos los permisos que deben concederse. Bajo ninguna circunstancia debería darse privilegios administrativos al middleware.

Criterio de defensa en profundidad

- El principio de defensa en profundidad sugiere que donde con un control sería razonable, más controles contra diferentes tipos de riesgo serían mayores. Los controles, cuando se utilizan en profundidad, pueden hacérselo extraordinariamente difícil a severas vulnerabilidades y por lo tanto con poca probabilidad de que ocurran. Con la codificación segura, esto puede tomar la forma de validación basada en filas, controles de auditoria centralizados, y requerir a los usuarios hacer login en todas las páginas.

Ejemplo, una interfaz administrativa con defectos es poco probable que sea vulnerable a ataques anónimos si incorpora el acceso correctamente a redes de administración en producción, chequea la autorización administrativa del usuario, y hace log de todos los accesos.

Manejo seguro de errores

- Una vez que un fallo de seguridad ha sido identificado, es importante desarrollar un test para él y comprender la raíz del problema. Cuando se usan los patrones de diseño, es muy probable que el fallo de seguridad se encuentre muy extendido en todo el código base, por lo que desarrollar la solución correcta sin introducir regresiones es esencial.

Ejemplo, un usuario ha visto que es capaz de ver las cuentas de otro usuario simplemente ajustando su cookie. La solución parece ser relativamente sencilla, pero como el manejo de la cookie es compartido entre todas las aplicaciones, un cambio en una simple aplicación repercutirá en todas las demás. La solución por lo tanto debe testearse en todas las aplicaciones afectadas.

Criterio del “Fallo Seguro”

- Las aplicaciones fallan regularmente al procesar transacciones debido a diversas razones. De la manera en que fallan se puede determinar si una aplicación es segura o no.

```
isAdmin = true;
try {
      codeWhichMayFail();
      isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex) {
            log.write(ex.toString());
}
```

Sí el código codeWhichMayFail() falla, el usuario es administrador por defecto. Obviamente esto es un riesgo de seguridad.

Definición de mensajes de error

- Los mensajes de error detallados proveen a los atacantes con una enorme cantidad de información utilizada.
- Muchos lenguajes indican una condición de error por valor de retorno. ¿Hace uso el código de manipuladores de excepciones estructurados (try {} catch {} etc.) o de manipulación de errores basado en funciones?

Ejemplo:

· ¿Son todos los errores funcionales chequeados? Si no lo son, ¿que puede fallar?

Prevención de divulgación de información

- Los usuarios se resisten a enviar **detalles privados** a un sistema. Es posible para un atacante revelar detalles de usuario, ya sea anónimamente o como un usuario autorizado.
- Las aplicaciones deben incluir **controles fuertes para prevenir manipulación de identificación de usuario**, particularmente cuando ellos usan una única cuenta para correr la aplicación entera.
- El navegador del usuario puede fugar información. **No todos los navegadores implementan correctamente políticas** de manejo de caché pedidos por las cabeceras HTTP
- Cada aplicación tiene la responsabilidad de **minimizar la cantidad de información almacenada por el navegador**, previendo que pueda divulgar información y pueda ser utilizada por un atacante

Manejo de información sensible

- El mejor desempeño y mayor seguridad a menudo se obtiene a través de **procedimientos almacenados** parametrizados, seguido de consultas parametrizadas (también conocidas como declaraciones preparadas) con una fuerte tipificación de los parámetros y esquemas. La principal razón para el uso de procedimientos almacenados es reducir al mínimo el tráfico de la red en transacciones de múltiples niveles o para evitar que **información sensible** sea transmitida por la red.

Solo tener cuidado de no atarse al vendedor de la base de datos.

Los sistemas externos son inseguros

- Diversas organizaciones utilizan las capacidades de procesamiento de **terceras compañías**, las cuales más que a menudo tienen **diferentes políticas de seguridad** y posturas que la suya.
- Es poco probable que pueda controlar o influenciar en una tercera parte externa, si ellas son usuarios domésticos o grandes suministradores o socios. De ahí que, **la confianza implícita de ejecutar sistemas externos, no está garantizada**. Todos los sistemas externos deberían ser tratados de un modo similar.

Separación de funciones

- Un control clave del fraude es la separación de funciones. Por ejemplo, alguien que solicita un ordenador no puede anunciarlo también, no debería recibir directamente el ordenador. Esto previene que el usuario solicite varios ordenadores y reclame que nunca le llegaron.
- Ciertos roles tienen **niveles diferentes de confianza** que los usuarios normales. En particular, los administradores son diferentes que los usuarios normales. En general, los administradores no deberían ser usuarios de la aplicación.

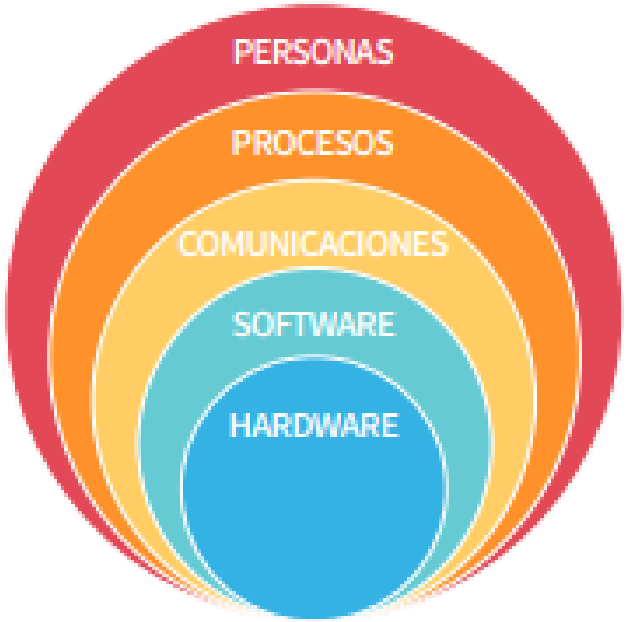
No confíes en la seguridad a través de la oscuridad

- La seguridad a través de la oscuridad es un control de seguridad débil, y además siempre fallan cuando son el único control. Esto no significa que mantener secretos es una mala idea, significa simplemente que la seguridad de los sistemas clave no debería basarse en mantener detalles ocultos

Ejemplo, la seguridad de una aplicación no debería basarse en mantener en secreto el conocimiento del código fuente. La seguridad debería basarse en muchos otros factores, incluyendo políticas razonables de contraseñas, defensa en profanidad, límites en las transacciones de negocios, arquitectura de red sólida, y controles de auditoria y fraude.

Almacenamiento seguro

- Clasificación de la información y almacenamiento adecuado
- Métodos de recuperación
- Procedimientos de borrado seguro
- Conservación y archivado



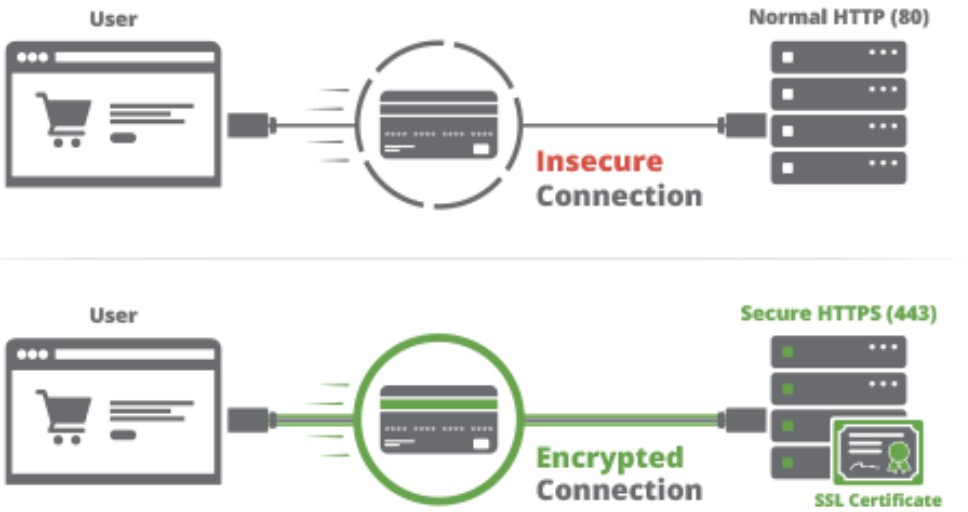
Elementos de un sistema de información

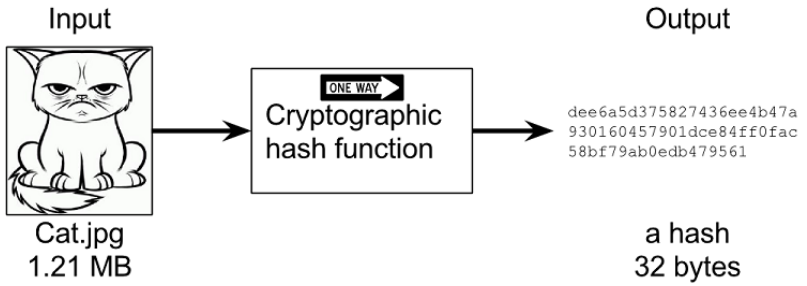
Transferencia segura

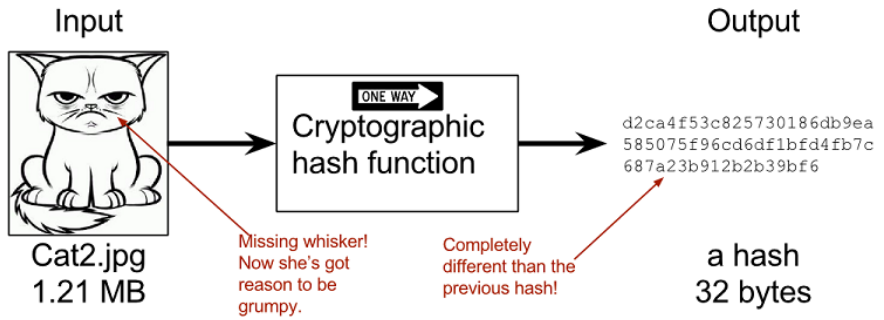
- Secure Socket Layer (SSL) es un protocolo desarrollado por Netscape en 1996 que pronto se convirtió en el método elegido para asegurar las transmisiones de datos por Internet.
- La aplicación más corriente de los certificados SSL es la de asegurar la transferencia de datos entre exploradores y servidores web.

Encriptación y Hashes

- Cifrado
- Hash







Autor: ccarrenovi@Gmail.com

Auditoría y Logging

- Una tarea común, típicamente requerida par las auditorias, es reconstruir la cadena de eventos que llevó a un problema en particular. Usualmente, esto se logra al guardar **registro de eventos** en el servidor en una ubicación segura, disponible solo para los administradores de TI y los auditores de sistema, para crear lo que comúnmente se conoce como un **rastro de auditoria**.
- Las aplicaciones Web y los servicios web no son excepción a esta práctica y sigan la solución común a otros tipos de aplicación Web.

Diseño de Autenticación y Autorización

- La autenticación es solo tan fuerte como sus procesos de administración de usuarios.
- Use la forma más apropiada de autenticación adecuada para su clasificación.
- Re-autenticar al usuario para transacciones de alto valor y acceso a áreas protegidas
- Autenticar la transacción, no el usuario.
- Las contraseñas son trivialmente rotas y no son adecuadas para sistemas de alto valor

Seguridad en Web Services

Los servicios Web, como otras aplicaciones distribuidas, requieren protección en múltiples niveles:

- Los mensajes SOAP que son enviados en la red deben ser entregados confiablemente y sin modificaciones
- El servidor necesita saber con confianza con quien esta hablando y a que tienen derecho los clientes
- Los clientes necesitan saber que están hablando al servidor correcto y no a un sitio de "**phishing**"
- El registro de eventos debe contener suficiente información para reconstruir confiablemente la cadena de eventos y rastrear de vuelta a los que llamaron funciones sin autentificación previa

Diseño de protección contra Denial of Service (D.O.S)

- Los ataques de denegación de servicio (Denial of Service – DoS) se dirigen principalmente contra software conocido, mediante vulnerabilidades que permiten que estos ataques resulten satisfactorios.
- Consumo excesivo de la CPU
- Consumo excesivo de disco de Entrada/Salida
- Consumo excesivo de entrada/salida en red
- Bloqueo de cuentas de usuario

Errores de Lógica de negocio

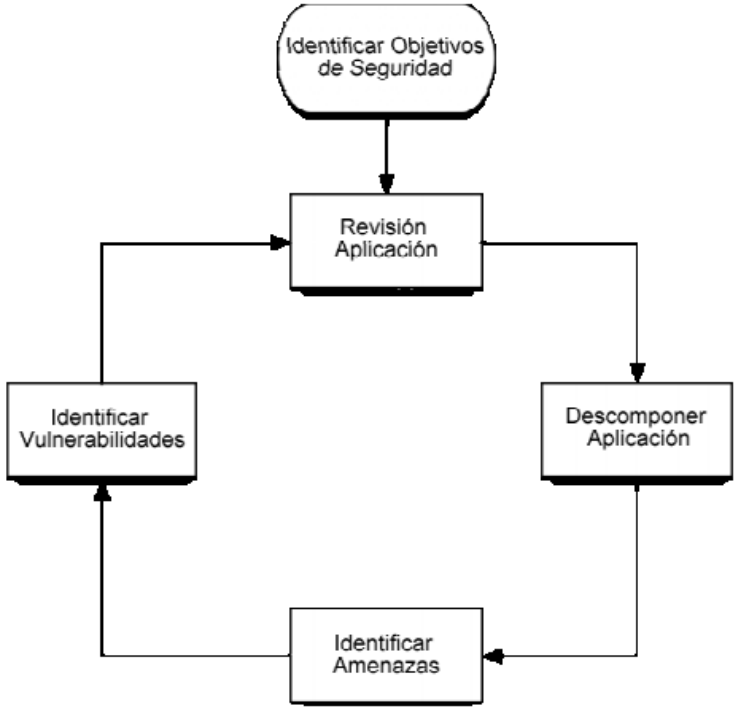
- La mejor manera tratar los errores de lógica de negocio es obtener la **documentación de arquitectura y diseño de la aplicación**. Busque diagramas de componentes UML. Los diagramas de componentes de alto nivel son generalmente todo lo que se requiere para entender como y porque la información fluye a distintos lugares.
- Información que cruza un límite de confianza , desde el Internet al código de la interfaz o desde la **lógica de negocio** al servidor de base de datos, necesita ser analizado cuidadosamente, mientras que los flujos dentro del mismo nivel de confianza no necesitan tanto escrutinio.

Modelado de Riesgo de Amenaza

- **Durante el diseño de su aplicación, es esencial que la diseñe utilizando controles evaluados de riesgo de amenaza**, de otra forma malgastara recursos, tiempo y dinero en controles inútiles y no suficiente en los riesgos reales. El método que utilice para determinar riesgo no es tan importante como hacer modelado de riesgo de amenaza estructurado.
- Microsoft señala que la mejora sencilla en su programa de mejora de seguridad fue **la adopción universal de modelado de amenaza**.
- **OWASP ha elegido el proceso de modelado de amenaza de Microsoft** ya que trabaja bien para los retos únicos enfrentando seguridad en aplicaciones, y es fácil de aprender y adoptar por diseñadores, desarrolladores y revisores de código.

Ejecutando modelado de riesgo de amenazas

- Flujo del Modelo de Amenaza



Autor: ccarrenovi@Gmail.com

Identificar objetivos de seguridad

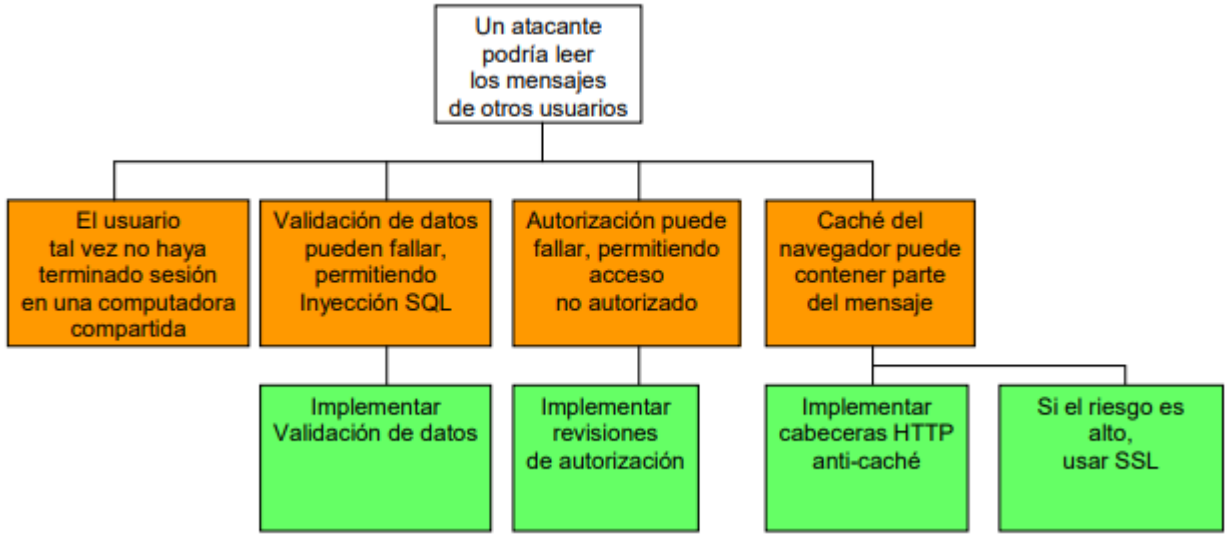
- Identidad
- Reputación
- Financiero
- Privacidad y regulaciones
- Disponibilidad de garantias

Descomponer la aplicación

- Componentes
- Flujos de datos
- Límites de confianza
- Modelos
- Arquitectura

Identificar Amenazas

- Documentar la amenazas



Que amenaza al sistema de software

- **Descubrimiento accidental**: Usuarios autorizados pueden toparse con un error en la lógica de su aplicación utilizando simplemente un navegador.
- **Malware automatizado** (buscando vulnerabilidades conocidas pero con un poco de malicia e inteligencia)
- **Atacante Curioso** (como investigadores de seguridad o usuarios que notaron algo mal en su aplicación y prueban más allá)
- **Script kiddie**: criminales computacionales atacando o desfigurando aplicaciones por respeto o motivos políticos – utilizando técnicas descritas aquí o en la Guía de Pruebas de OWASP para comprometer su aplicación.
- **Atacantes motivados** (como personal disgustado o un atacante pagado).
- **Crimen organizado** (generalmente para aplicaciones de alto riesgo, como comercio electrónico o bancario

Identificar la Vulnerabilidades

- Documentar las vulnerabilidades que pueden ser explotadas por las amenazas.
- Documentar las estrategias de mitigación del riesgo

Clasificación de Amenazas: STRIDE

- STRIDE

Matriz DREAD

- Calculo de Matriz DREAD

Escala de Riesgo

Bajo = 1
Medio = 2
Alto = 3

Cálculo de Riesgo

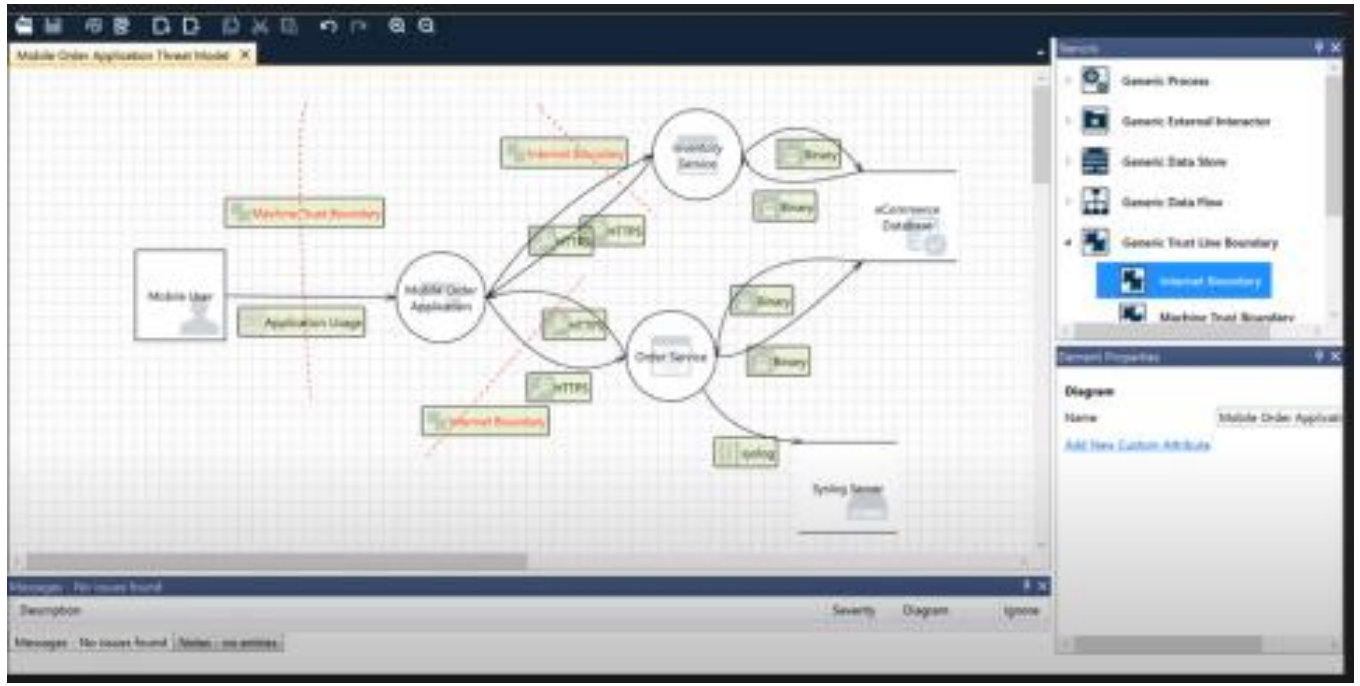
$$\frac{D_a+R+E+A+D_i}{5} = Riesgo$$

	Amenaza	Da	R	E	A	Di	Riesgo
1	Agente externo envía enlace falso a usuario real para intentar obtener credenciales	3	2	2	2	1	**2**
2	Envió de ataque XSS con intención de alterar sistema de archivos del servidor.	1	1	2	1	1	**1.2**
3	Ya que usuarios anónimos pueden subir imágenes, estos pueden negar que conocían los términos y condiciones de uso	1	1	3	1	3	**1.8**
4	s de comunicación (HTTP), descifrar o revisar credenciales	3	3	3	2	1	**2.4**
5	Sniffer podría interceptar imágenes y utilizarlos con otros fines	1	3	2	3	2	**2.2**
6	Almacen de imágenes llega al límite, no puede recibir más archivos	1	1	1	3	1	**1.4**
7	DDOS con imágenes al Sistema Web (Ataque a recursos)	1	3	3	3	3	**2.6**
8	Usuario anónimo podría intentar realizar acciones de un usuario administrador	3	1	2	3	1	**2**

Autor: ccarrenovi@Gmail.com

Microsoft threat modeling tool

- Herramienta para el modelado de amenazas



Autor: ccarrenovi@Gmail.com

Lab

- Lab 2 Crear el modelo de amenazas de una aplicación de compra en línea

Referencias

- https://owasp.org/
- https://www.microsoft.com/en-us/download/details.aspx?id=49168