

# Problemas de seguridad en las aplicaciones Web



OWASP - The open web application security project

Juan Díez-Yanguas Barber

# Contenidos

- ⦿ Introducción
- ⦿ Riesgos en aplicaciones web
- ⦿ Soluciones y otros consejos
- ⦿ Breve demostración de la librería de seguridad ESAPI & Antisamy

# Introducción

- ⦿ Es necesaria una planificación de segura desde el inicio del proyecto.
- ⦿ El desarrollador no debe tomar ningún riesgo. Todo riesgo posible debe de ser por parte del usuario (si es que se le permite elegir)
- ⦿ Principio del mínimo privilegio
- ⦿ La utilización de servicios externos conlleva un riesgo añadido
- ⦿ No confiar en la seguridad a través de la oscuridad
- ⦿ Fallar de manera segura

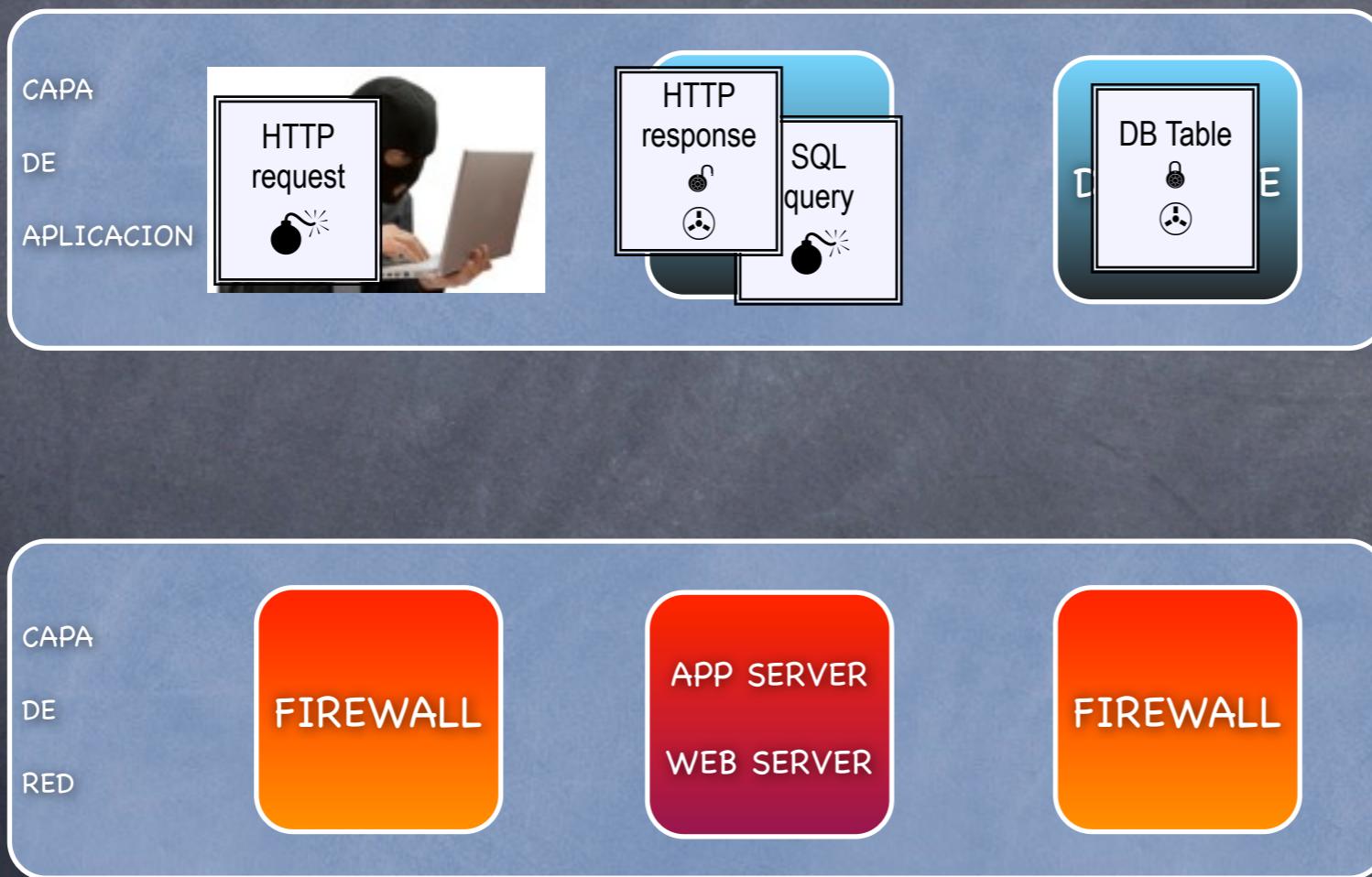
# Riesgos y vulnerabilidades

- ⦿ Inyección de código SQL
- ⦿ Cross Site Scripting (XSS)
- ⦿ Robo de sesión
- ⦿ Acceso a URLs restringidas

# SQL Inyección – Introducción

- Consiste en hacer que la aplicación envíe datos introducidos por el usuario al intérprete de comandos
- Los intérpretes toman los datos de entrada y los ejecutan como comandos
- Es uno de los ataques más usados, a pesar de la simplicidad de su evitación
- Impacto: GRAVE. La base de datos se puede comprometer por completo

# SQL Inyection



## Account Summary

Acct:5424-6066-2134-4334  
Acct:4128-7574-3921-0192  
Acct:5424-9383-2039-4029  
Acct:4128-0004-1234-0293

- La aplicación presenta un formulario
- El atacante introduce un ataque en el formulario
- La aplicación envía el ataque a la base de datos en una consulta SQL
- La base de datos procesa la consulta con el ataque y envía los datos de vuelta encriptados a la aplicación
- La aplicación desencripta los datos como acostumbra a hacer y muestra los datos al usuario

# Riesgos y vulnerabilidades

- ⦿ Inyección de código SQL
- ⦿ Cross Site Scripting (XSS)
- ⦿ Robo de sesión
- ⦿ Acceso a URLs restringidas

# Cross Site Scripting (XSS) - Introducción

- Consistente en enviar datos sin procesar de un formulario
- Los datos sin procesar se almacenan en la base de datos
- Se muestran los datos almacenados en la base de datos en el navegador del usuario
- Impacto: Robar datos privados o la sesión de usuario, re-dirigir al usuario a páginas con malware
- Máximo impacto: Instalar mediante XSS un proxy y permitir al atacante re-dirigir al usuario a sitios distintos a los que desea visitar con el propósito de robar la identidad

# XSS



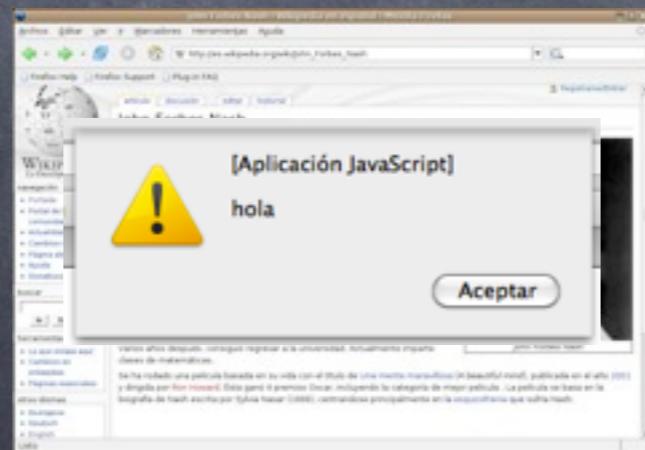
El atacante inserta script en una web que copie la información introducida por los usuarios a la base de datos



DATABASE



Un usuario visita la página y se ejecutará el script introducido por el atacante



Ejemplo meramente académico.  
Si lo que ha logrado introducir el atacante fuera en vez de eso un script para robar una cookie el atacante estaría recibiendo la cookie del usuario y con ella podría ver datos personales del mismo o robar una sesión

# Riesgos y vulnerabilidades

- ⦿ Inyección de código SQL
- ⦿ Cross Site Scripting (XSS)
- ⦿ Robo de sesión
- ⦿ Acceso a URLs restringidas

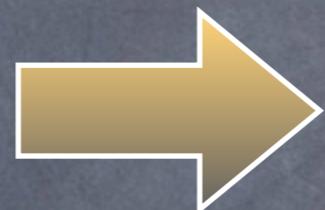
# Robo de sesión – Introducción

- HTTP es un protocolo sin estados, lo que significa que las credenciales han de ir en cada petición
- Dado que las credenciales aparecen en cada petición están más expuestas en los navegadores, la red y los logs
- Impacto: Sesiones de usuario comprometidas

# Robo de sesión

The screenshot shows a Windows Internet Explorer window. The address bar contains the URL [www.boi.com?JSESSIONID=9FA1DB9EA...](http://www.boi.com?JSESSIONID=9FA1DB9EA...). Below the address bar is a large image of a woman holding a coffee cup. To the right of the image is a login form titled "How to Hijack a Session". The form includes fields for "User Name" and "Password", and a "Login" button. The page is part of the OWASP WebGoat V5 project.

El usuario inicia sesión en un sitio y el sitio indica la ID de sesión en la URL



El usuario hace click en un enlace que lleva a la web del atacante  
<http://www.atacante.com/>

The screenshot shows a Firefox browser window. The address bar contains the URL <http://www.atacante.com/>. The page displays a login form with the title "H4x0r3d by [^0z0n3^]". It includes fields for "User Name" and "Password", and a "Login" button. The page is part of the OWASP WebGoat V5 project.

The screenshot shows a Windows Internet Explorer browser window. The address bar contains the URL <http://localhost/WebGoat/attack?Screen=45&menu=310>. The page displays a login form with the title "How to Hijack a Session". It includes fields for "User Name" and "Password", and a "Login" button. The page is part of the OWASP WebGoat V5 project.



El atacante observa en la URL el JSESSIONID y se autentifica como si fuera la víctima



El hacker ve los log y se fija en el referer

# Riesgos y vulnerabilidades

- ⦿ Inyección de código SQL
- ⦿ Cross Site Scripting (XSS)
- ⦿ Robo de sesión
- ⦿ Acceso a URLs restringidas

# Acceso a URLs restringidas

- ⦿ No basar la seguridad únicamente en la ocultación
- ⦿ El atacante cambia parte de la URL y accede a páginas restringidas
- ⦿ Impacto: El atacante realiza acciones de administración, accede a datos de otros usuarios...

# Soluciones Inyección SQL

- ⦿ Usar variables que obliguen a un determinado formato. Permiten al intérprete distinguir entre código y datos (PreparedStatement)
- ⦿ Escapar las entradas del usuario. Es posible la utilización de la librería ESAPI
- ⦿ Usar una lista blanca para validar las entradas del usuario. Es posible el uso de ESAPI

# PreparedStatement

- Permite la definición exacta del formato de la sentencia SQL
- Ejemplo (No hay validación)

```
Connection conexion = pool.getConnection();
```

```
PreparedStatement statement = conexion.prepareStatement("INSERT INTO AGENDA VALUES  
    (?, ?, ?)");  
statement.setString(1, request.getParameter("name"));  
statement.setString(2, request.getParameter("phone"));  
statement.setInt(3, request.getParameter("age"));  
  
statement.executeUpdate();
```

# ESAPI - Escapar entradas de usuario

- Es posible escapar las entradas del usuario mediante ESAPI

```
protected String escapeUserEntries (String input){  
    Encoder encod = ESAPI.encoder();  
    return encod.encodeForSQL(new MySQLCodec(MySQLCodec.MYSQL_MODE),  
    input);  
}
```

# ESAPI – Validar entradas de usuario

- ⦿ Es posible validar las entradas del usuario mediante expresiones regulares mediante ESAPI
- ⦿ La validación previene ante ataques de Inyección SQL

# ESAPI – Validar entradas de usuario

- Es posible usar la librería configurando previamente un fichero de validación para validar las entradas del usuario
- Codificación UTF16. Usar hexadecimal para caracteres regionales. Ejemplo: á \u00e1
- Aprender con RegexBuddy
- Configurar el classpath
- Mirar API para más opciones de validación y mensajes de error

# ESAPI - Validar entradas de usuario

```
Validator.Name=^[A-Z][a-zA-Z -áéíóúüñ]+$  
Validator.Adress=^[A-Z][a-zA-Z0-9\-\ \ ,ºáéíóúüñ]+[0-9]{5}\-[A-Z][A-Za-z]\ \-áéíóúüñ]+$
```

```
public static String validateName (String input) throws IntrusionException, ValidationException {  
    Validator validador = ESAPI.validator();  
    //Contexto, entrada, expr, maxlen, null  
    return validador.getValidInput("Nombre", input, "Name", 100, false);  
}  
  
public static String validateAdress (String input) throws IntrusionException, ValidationException{  
    Validator validador = ESAPI.validator();  
    return validador.getValidInput("Dirección", input, "Adress", 200, false);  
}  
  
public static int validateNumber (String input, String context) throws IntrusionException, ValidationException{  
    Validator validador = ESAPI.validator();  
    //Contexto, entrada, minimo, maximo, null  
    return validador.getValidInteger(context, input, 0, 999999, false);  
}
```

# ESAPI – Validación HTML

- Usando el módulo Antisamy es posible la validación de HTML para la prevención de ataques de XSS
- Para la validación se usará un fichero de reglas, el cual lo podemos obtener online

# Antisamy – Validación HTML

- ⦿ Pensado para la validación de código html introducido por el usuario.
- ⦿ Evita etiquetas sospechosas
- ⦿ Fichero de configuración xml (Diferentes políticas de seguridad en la web)
- ⦿ Es posible su uso en filtros

# Antisamy - Validación HTML

- ⌚ Varios métodos para indicar errores

```
private boolean isStringSecure (String input, ServletRequest request) throws
ScanException, PolicyException{

    Policy politica = Policy.getInstance(this.getClass().getResourceAsStream("/
antisamy-slashdot-1.4.3.xml"));
    AntiSamy validator = new AntiSamy();
    CleanResults cr = validator.scan(ESAPI.encoder().canonicalize(input), politica);
    if (cr.getNumberOfErrors() == 0){
        return true;
    }else {
        return false;
    }
}
```

# Acceso a URLs restringidas - Filtros

- Comprobar siempre todos los accesos a las URLs restringidas, nunca suponer al usuario autenticado.  
Solución: Filtros
- Si el servidor de aplicaciones proporciona métodos de sesión usarlos
- Usar una buena organización de directorios y asegurarse de que están protegidas todas las peticiones
- ESAPI también proporciona herramientas:  
`isAuthorizedForURL()`

# Fallos seguros

- ⦿ Ante un fallo no dar información técnica del mismo. Páginas de error personalizadas.
- ⦿ No dar información técnica del sistema. No usar páginas de error predefinidas
- ⦿ Ante un fallo no puede quedar comprometida una operación en curso. Comprobar todo en cada paso de la operación

# Preguntas

