



# Seguridad, Web y Java

**Daniel López Janáriz**

d.lopez@uib.es

## 1. Introducción: Puntos a tener en cuenta cuando hablamos de seguridad

### La seguridad al 100% no existe

Aunque resulte obvio, cabe resaltar de nuevo que la seguridad 100% no es viable y que intentar conseguirla puede llevarnos a desperdiciar muchos recursos que no serían necesarios. El objetivo es llegar a una seguridad adecuada al tipo de aplicación y la información que queremos proteger, optimizando el uso de recursos y el coste, a la vez que minimizando las interferencias de la seguridad en la funcionalidad de la aplicación.

Un atacante con recursos y tiempo ilimitados puede comprometer cualquier sistema, pero afortunadamente no todas las aplicaciones son susceptibles de ser atacadas con semejante generosidad de medios. Así pues, sería conveniente definir el perfil del posible atacante, en función de la aplicación e implementar la seguridad adecuada para evitar que ese tipo de atacantes logre su objetivo. Sea por falta de tiempo, de recursos o de interés por requerir un esfuerzo demasiado elevado para la recompensa obtenida.

En este sentido, en algunas aplicaciones será imprescindible implementar unos controles de seguridad exhaustivos y en otros simplemente no será necesario, dado que la información a proteger no es tan importante.

### Equilibrio entre seguridad y amigabilidad

Al hilo del primer punto y teniendo en cuenta que una de las ventajas de las aplicaciones web es la facilidad de uso que permite que los usuarios navegantes las utilicen, es importante hallar un equilibrio entre los controles de seguridad necesarios y los inconvenientes que ponemos a los usuarios legítimos de nuestro sitio.

Por ejemplo, algunas aplicaciones implementan la funcionalidad de “recordar mi identidad” para identificarse automáticamente al acceder de nuevo. En casos donde la usurpación de personalidad dentro de una aplicación no suponga un grave riesgo, esta funcionalidad puede ser aceptable. En otros muchos casos el riesgo puede ser demasiado alto y será imprescindible implementar la caducidad de la sesión y la obligatoriedad de identificarse en cada sesión, con los inconvenientes añadidos para el usuario.

Al mismo tiempo, dado el ritmo al que se trabaja en aplicaciones para Internet, conviene tener en cuenta los costes extras de desarrollo asociados al nivel de seguridad implementado. Un nivel de seguridad muy alto puede conllevar que hacer modificaciones en nuestro código sea muy costoso en tiempo, a la vez que puede disminuir el rendimiento de la aplicación. Por ello es conveniente también analizar las repercusiones que la seguridad tendrá sobre el desarrollo y la ejecución de nuestra aplicación.

### Asegurando nuestras aplicaciones

Teniendo en cuenta lo anterior, a la hora de implementar la seguridad en nuestras aplicaciones, además de analizar que nivel de seguridad debemos implementar en función de los requerimientos de cada aplicación, debemos estudiar los posibles problemas de seguridad y como solucionarlos con las herramientas con las que dispongamos.

## 2. Seguridad en Web: Retos a los que deben hacer frente las aplicaciones web

Las aplicaciones web tienen una serie de características especiales que afectan la forma en que debemos afrontar el problema de la seguridad. Las podemos resumir en:

- La comunicación cliente/servidor, es, a nivel básico, no orientada a la conexión y sin estado.
- El acceso a la aplicación se realiza desde máquinas y aplicaciones no controladas.

Estas características básicas definen de forma sucinta lo que podemos esperar al realizar una aplicación web: *En principio no nos podemos fiar de ninguna petición que llegue.*

### Autenticación y control de acceso

Como autenticación entenderemos el identificar que el usuario es quien dice ser y como control de acceso, la facultad de permitirle o no realizar peticiones a la aplicación en función de los privilegios que se la hayan otorgado a dicho usuario.

En el caso de aplicaciones web, dado que la comunicación es sin estado, deberemos proveer los mecanismos necesarios para asegurar que una vez se ha autenticado un usuario, las siguientes peticiones que realice mantengan las credenciales de seguridad adecuadas sin tener que volver a repetir el proceso

de autenticación, teniendo en cuenta que cualquier aspecto de la petición se puede falsear.

### **Confidencialidad de la información**

La confidencialidad en este caso se refiere a evitar que la información que maneja la aplicación, tanto las peticiones como las respuestas, puedan ser accedidas por personas que no tienen los privilegios suficientes.

Dado las características anteriormente explicadas, no sólo es importante en este caso encriptar las comunicaciones, si no que debemos asegurarnos que le mostramos la información sólo a los usuarios con los privilegios requeridos.

Dentro de este punto también podríamos integrar el no mostrar información sensible del sistema que no está destinada al usuario, como excesiva información en los mensajes de error, acceso a información de configuración de la aplicación o la base de datos, etc.

### **Integridad de la información**

Como integridad entenderemos no sólo el asegurar que los datos que envía el usuario son los mismos que recibimos, si no el no permitir la modificación de los datos que no deban ser modificados.

De nuevo teniendo en cuenta las características de las aplicaciones web, es sumamente fácil modificar los parámetros que se envían en una petición, o realizar peticiones creadas expresamente, por que lo que en algunos casos deberemos implementar medidas extra, necesarias para preservar la integridad de nuestro sistema y de la información en ella contenida.

### **Protección de los recursos del sistema**

Otra forma de ataque a una aplicación web es no atacar la aplicación en si misma pero atacar a través suyo los recursos del sistema donde la aplicación está alojada. Es por eso que debemos tener cuidado para que a través de nuestras aplicaciones no se pueda atacar a otros recursos, como bases de datos, la CPU, el sistema de ficheros...

### **Restricciones funcionales**

Además de restringir el acceso a los usuarios, también debemos controlar que utilicen la

aplicación para aquello para lo que están autorizados, restringiendo lo que pueden hacer.

Al mismo tiempo, a veces se hace necesario limitar lo que la propia aplicación puede hacer para evitar que en caso de problemas, pueda ser utilizada de puente para realizar acciones que de otro modo no estarían permitidas. Por ejemplo, si una aplicación únicamente se ha de conectar a un servidor auxiliar determinado por un puerto determinado, algunas veces será necesario limitar el acceso para no permitir que desde la aplicación podamos acceder a otros servidores u otros puertos en el mismo servidor. Este punto es especialmente sensible si estamos ejecutando aplicaciones que no hemos hecho nosotros. (Por ejemplo en casos de "hosting")

### **Control de responsabilidad (accountability)**

En algunos casos, también se hace necesario un complemento a la seguridad que es la capacidad de poder determinar quién/como/cuando ha realizado determinadas acciones en el sistema. No sólo para ayudar en las tareas de detección de intrusiones, si no en la de poder pedir responsabilidades a los usuarios por los actos realizados.

Dentro de este ámbito se engloban habitualmente técnicas de trazado de las acciones realizadas y mantenimiento de históricos, para poder reconstruir en caso necesario los pasos llevados a cabo en el sistema, cuándo, cómo y por quién.

## **3. Errores comunes: *Errores comunes que afectan a la seguridad al desarrollar aplicaciones web***

Cuando se plantea el desarrollo de aplicaciones para Internet, hay una serie de errores, muchos de ellos comunes a otras áreas de seguridad, que debemos evitar, y en los cuales suelen incurrir muchos desarrolladores.

### **Seguridad por ocultación**

Un error típico es pensar que lo que no se ve, no se sabe, o depender únicamente de la falta de conocimiento de un simple dato para nuestra seguridad. En este sentido debemos recordar que casi todo lo que recibe el usuario, es susceptible de ser analizado, manipulado y enviado de vuelta con manipulaciones.

Ejemplos de lo que no se debe hacer es usar campos ocultos para pasar información de sin verificarla

después; confiar que por que a un usuario no se le muestra una URL no accederá a ella, o que sólo accederá a ella con los parámetros que le hemos pasado; Confiar en procesos JavaScript sólo por que los hemos puesto en un fichero aparte., etc.

#### **Tratamiento incorrecto del acceso a BDD**

La mayor parte de las aplicaciones web acceden de una forma u otra a una base de datos. Dado que las aplicaciones web pueden tener muchos usuarios y no son orientadas a la conexión, hay que tratar con cuidado las sesiones que abrimos y como las usamos para no sobrecargar nuestra base de datos, ya que normalmente ésta tiene bastante más limitados sus recursos. Por ello es conveniente abrir las menores conexiones posibles y mantenerlas abiertas lo imprescindible, nunca confiando en que el poco tráfico o el comportamiento “benevolente” de los usuarios mantendrá nuestra base de datos a salvo.

Dentro de este apartado cabe mencionar el error recurrente de no gestionar el acceso a la base de datos, típicamente a través de un pool de conexiones, y confiar en la caducidad de la sesión para liberar las conexiones abiertas por el usuario.

#### **Concatenación de cadenas para crear instrucciones**

Volviendo al tema de no fiarse de nada que pueda introducir o manipular el usuario, uno de los grandes riesgos de seguridad es la creación de instrucciones ejecutables a partir de los parámetros enviados en la petición del usuario. Nunca se deben concatenar los parámetros enviados tal como vienen para crear instrucciones a ejecutar, sea para crear sentencias SQL o sea para crear comandos que después lanzaremos contra el sistema operativo.

Como ejemplos de ataques de este tipo podemos mencionar los casos de “SQL injection”, gracias a la concatenación incontrolada de cadenas para crear sentencias SQL, o algunos casos de “Cross Site Scripting”, al crear sentencias o redirecciones a través de JavaScript sin verificar que los parámetros de entrada nos mandan a otro sitio.

#### **Acceso incontrolado a recursos**

Otro de los errores típicos en aplicaciones web es el dar acceso a recursos, especificados por parámetros de entrada, sin controlar que realmente estamos accediendo únicamente a los recursos que tenemos permiso. Es una variante más de fiarse en exceso de

los parámetros que envía el usuario, sin pensar que pueden haber sido manipulados.

Los ejemplos en este caso pueden variar mucho: Desde conceder acceso a ficheros a través del un camino especificado como parámetro y no controlar si el camino real final se sale de los lugares que tenemos asignados (usando “../.” u otros caracteres para crear caminos alternativos), a permitir el envío de mensajes de correo por web con una dirección falsa, al aceptar por defecto que la dirección que viene como parámetro es la que nosotros pusimos allí.

### **4. Java: Características del lenguaje**

Antes de analizar como dar respuesta a los retos de seguridad en las aplicaciones web con Java, examinemos primero las características generales del lenguaje.

#### **Interpretado (compilado nativo “Just In Time”)**

Java es un lenguaje compilado e interpretado, ya que el código fuente Java se compilar para obtener el código compilado, compuesto de instrucciones simples conocidas como “bytecode”, que luego interpreta la maquina virtual Java. Sin embargo, las maquinas virtuales modernas compilan a código nativo el “bytecode”, aumentando de esta forma el rendimiento.

#### **Neutro en cuanto a plataforma**

Java como lenguaje es algo más que portable, puesto que la maquina virtual Java se encarga de emular la CPU de la maquina y abstraer al programa de detalles de la arquitectura como el orden de los bits, el numero de bytes que ocupan las primitivas, etc. Esta característica permite ejecutar un programa Java en una máquina distinta sin obtener comportamientos inesperados en el uso y acceso de la memoria, por ejemplo, lo cual es bastante importante.

#### **Múltiples controles de seguridad**

Desde que se escribe el programa hasta que se ejecuta, Java realiza múltiples controles de seguridad para controlar, por ejemplo, que el programa no realiza accesos incontrolados a posiciones de memoria y otros recursos. El primer control lo realiza el compilador, al transformar el código fuente en código compilado, después la

maquina virtual, a través del “Bytecode Verifier” comprueba que los bytecodes que carga siguen siendo correctos y no han sido manipulados. Por último, al ejecutar el programa las instrucciones son verificadas por un gestor de seguridad (SecurityManager) que vuelve a comprobar en tiempo de ejecución que el programa sólo realiza lo que le esta permitido hacer, según la política de seguridad que tenga definida (Security Policy).

## 5. Seguridad en Java: *Respuesta en Java a los retos de seguridad*

Teniendo en cuenta las características de Java y los retos de seguridad a los que debemos enfrentarnos ¿Cómo podemos afrontarlos desde éste lenguaje?

### Autenticación y control de acceso

La especificación para el desarrollo de aplicaciones web en Java, “Java Servlet Specification”, define la forma de proteger los recursos de nuestras aplicaciones web en base a roles, de forma que en cualquier contenedor de servlets donde ejecutemos nuestras aplicaciones web, éstas se encuentren protegidas.

De todas formas, en caso de no ser suficiente éste tipo de seguridad, lo habitual es implementar un control de seguridad utilizando las características de modularidad de Java, que permiten definir filtros por los que han de pasar todas las peticiones antes de poder ser ejecutadas (ServletFilters) y que permiten que normalmente todas las peticiones pasen primero por un controlador, donde se realizarían los controles de seguridad, antes de ser redirigidas donde toca. Esta técnica se conoce como “Servlet Controller” y es muy común en la mayoría de “frameworks” web en Java.

Para mantener el estado de la conversación entre el cliente y el servidor, la misma especificación de servlets define una serie de mecanismos estándar, a través de cookies o de parámetros, que permiten almacenar el estado en lo que se conoce como la sesión.

### Confidencialidad de la información

Para mantener la confidencialidad de la información se pueden utilizar varios mecanismos. Para las comunicaciones no se suele utilizar nada especial y no lo normal es recurrir al SSL, con HTTPS, como el resto de aplicaciones web.

Sin embargo, en caso de querer encriptar información Java proporciona por defecto librerías completas de generación, gestión y utilización de certificados para la encriptación, por lo que podemos ocultar nuestros datos sin problemas.

Otro mecanismo interesante para garantizar la seguridad al almacenar nuestros datos es que en caso de guardar el estado de nuestra aplicación en disco, para recuperarlo posteriormente por ejemplo, podemos marcar algunos campos como no-almacenables (atributo “transient”) de forma que no se escriban nunca en disco o se puedan transmitir por la red, asegurándonos que no puedan ser leídos o interceptados fuera de la máquina virtual.

### Integridad de la información

A la hora de asegurar la integridad de la información, Java incluye por defecto librerías que permiten la creación de firmas digitales, que tanto se pueden usar para los contenidos como para firmar los programas y asegurarse de que los programas que se ejecutan en un sistema son los que nosotros desarrollamos y no han sido manipulados.

### Protección de los recursos del sistema

Para la protección de los recursos del sistema, existen diversas soluciones, según el tipo de problema que nos estemos planteando:

- Si estamos hablando de proteger la base de datos, Java provee mecanismos para definir y controlar el número de conexiones abiertas a una base de datos (DataSources), con lo que podemos asegurar que no la colapsaremos.
- Para proteger la memoria del sistema, la maquina virtual permita limitar al memoria máxima que puede reservar, con lo que no podemos consumir memoria indefinidamente.
- Para otros niveles de protección como el uso de CPU, Java no ofrece ninguna protección, al igual que la mayoría de lenguajes.

### Restricciones funcionales

A la hora de restringir la funcionalidad de los usuarios, el primer paso es implementar correctamente la seguridad a nivel de nuestra

aplicación, pero si aun así queremos estar seguros, o si el código no lo hemos hecho nosotros, para impedir el acceso a ficheros que no debamos, cortar el acceso desde nuestro programa a otros servidores, o no permitir el acceso desde otros servidores a nuestros programas etc., Java permite ejecutar los programas bajo el control de un gestor de seguridad (Security Manager) al cual se le puede especificar exactamente qué acciones puede realizar cada librería, incluso según el certificado con el que este firmada.

Para niveles aun más detallados, se puede llegar a especificar bajo qué credenciales se ejecutan ciertos trozos de código.

#### **Control de responsabilidad (accountability)**

En el tema de control de responsabilidad, es la aplicación la responsable de guardar trazas de los pasos seguidos por el usuario y aparte de librerías estándar para la generación de trazas, Java no puede ofrecernos mucho más puesto que es una cuestión particular de cada aplicación.

#### **Conclusiones**

En esta presentación hemos expuesto brevemente los retos a los que nos debemos enfrentar desde el punto de vista de la seguridad cuando desarrollamos una aplicación web, y como el lenguaje Java nos proporciona algunas herramientas para afrontar dichos retos. En algunos casos nos ayudará más y en otros menos, sin embargo la mayor responsabilidad recae del lado del programador o administrador y del buen uso de los mecanismos que Java, o cualquier otro lenguaje, le proporcionan.