Espacio del divulgador

Métodos de programación segura en Java para aplicaciones móviles en Android

Rodrigo Pimienta García*, Gualberto Aguilar Torres*, Manuel Ramírez Flores* y Gina Gallegos García*

Recepción: 26 de agosto de 2013
Aceptación: 2 de diciembre de 2013

***Instituto Politécnico Nacional, México.**
Correo electrónico: rodrigopg.lv99@gmail.com; autg79y@yahoo.com; manuel300688@hotmail.com y gganig@hotmail.com
Se agradecen los comentarios de los árbitros de la revista.

Resumen. Hoy en día la mayoría de las personas posee un teléfono celular, ya sea con un sistema operativo clásico o un *smartphone*, que no sólo tienen la capacidad de realizar llamadas, sino también de instalar *software* o navegar por internet. Estos dispositivos se han convertido en un medio de almacenamiento de información, por lo cual son vulnerables a robo. Por lo anterior, se presentan las vulnerabilidades que afectan a Android y se propone el uso de práctica de programación segura. Los resultados mostraron mitigar satisfactoriamente las vulnerabilidades, con lo cual se crean métodos eficientes de programación segura.
Palabras clave: Android, vulnerabilidades, programación segura.

Safe Programming Methods in Java for Mobile Applications in Android

Abstract. Nowadays most of the people have a cell phone with a classic operating system or a *Smartphone*, which does not have only the capacity of making calls, but also has functions like installing *software*, surfing on the Internet, etc.
These devices have become a data storage media, and can be targeted by malicious people to steal information. One way to reduce this risk is by developing secure applications. In this article, the vulnerabilities which affect Android are presented and a use of safe programming practices is proposed.
Key words: Android, vulnerabilities, secure programming.

Introducción

Actualmente, 80% de la población mundial posee un teléfono móvil, 78.4% son teléfonos convencionales con sistema operativo Symbian y el restante 21.6% es un teléfono inteligente o *Smartphone* (Go-Globe, 2012), con un sistema operativo de última generación de los cuales Android es el líder del mercado (46.3% de los dispositivos), seguido por iOS (30%) y RIM(14.9%) (Nielsen, 2012).

Con la evolución de las tecnologías tanto de *hardware* como de *software* y la popularización de dispositivos móviles es más fácil que los creadores liberen herramientas que favorezcan a cualquier programador para que desarrolle aplicaciones. Esta evolución también se presenta en el área de la seguridad, ya que nuevas tecnologías traen consigo nuevas vulnerabilidades que afectan la información de los usuarios.

Es por ello que los programadores no sólo deben tener en cuenta que las aplicaciones cumplan con los requerimientos del proyecto, sino que al mismo tiempo con los estándares de seguridad que ofrecen distintas instituciones para garantizar la integridad, confidencialidad y disponibilidad de los datos del usuario, ya que una pequeña vulnerabilidad en el código del programa puede no sólo comprometer la información que esa aplicación maneja, sino todos los datos que se encuentran almacenados en el dispositivo.

Cabe destacar que en cuanto a programación segura, la CERT (Computer Emergency Response Team) provee estándares para Java. Sin embargo, dado que en Android se usa una versión reducida de Java (JME, Java Micro Edition), no posee todas las clases y métodos de la versión estándar. Además, la guía está orientada a aplicaciones de computadoras, no para móviles.

1. Seguridad en *software*

Los principales problemas al momento de desarrollar *software* son los defectos en el código, ya que se requiere entrenamiento y experiencia para comprender su funcionamiento y como resolverlos apropiadamente. Inclusive, los errores más básicos en el código pueden ser atacados y explotados de muchas formas. Aunque el código generado es responsabilidad del programador, una aplicación puede ser atacada si posee una plataforma vulnerable. En el caso de Android, las aplicaciones son programadas en Java que corre sobre una máquina virtual llamada Dalvik.

1. 1. Seguridad en Java

Java ha sido un lenguaje para trabajar de forma segura debido a la implementación de librerías y ambientes de ejecución seguros. Es por ello que al hablar de su seguridad, se debe considerar en tres contextos distintos (Sundsted, 2001): *a*) seguridad en la máquina virtual, *b*) seguridad en las aplicaciones y *c*) seguridad en red.

Java es seguro únicamente en uno de estos tres contextos; la máquina virtual, los otros dos dependen meramente del programador. Se ha diseñado para hacer imposible ciertas clases de ataques, entre los que se encuentran (Horstmann y Cornell, 2006): *a*) desbordamiento de la pila de ejecución (*StackOverflow*), *b*) corromper la memoria fuera de su propio espacio de proceso y *c*) leer o escribir archivos sin permiso.

1. 2. Seguridad en Android

Al igual que Java, Android también fue diseñado desde sus inicios para ser un sistema operativo móvil seguro; ahora es una plataforma de Linux, programado con Java con mejoras en sus sistemas de seguridad para correr bajo un ambiente móvil. Entre las características de seguridad que presenta se encuentran: *a*) compartimiento de memoria eficiente, *b*) multi-tareas con prioridad de procesos, *c*) identificadores de usuarios de sistemas Unix (UID, por sus siglas en inglés), *d*) permisos de archivos y *e*) caja de arena o *sandbox* en inglés

Una de las mejoras es visible en la parte de los identificadores de usuario, a diferencia de un sistema operativo de computadora, donde cada proceso corre bajo el UID del usuario. Las aplicaciones de Android son ejecutadas en procesos separados bajo distintos UID, es decir, cada aplicación posee uno con permisos distintos (Burns, 2008). Adicionalmente, cuentan con un archivo de permisos para acceder a los elementos del sistema operativo para proveer mayor seguridad cuando este archivo se encuentra correctamente configurado. Estos permisos son usualmente llamados permisos de Android o permisos de manifestación (*manifest permissions*) con el fin de no confundirlos con los de archivos.

1. 2. 1. Sandbox de Android

La máquina virtual Dalvik es una plataforma basada en registros para la ejecución de aplicaciones del sistema operativo Android. Los archivos ejecutables Dalvik fueron diseñados para optimizar el uso de recursos como la memoria y el procesador.

El objetivo principal de las *sandbox* es mantener el código aislado de tal forma que no pueda causar ningún daño al medio que lo ejecuta. Esto se logra restringiendo el acceso a los recursos y archivos del sistema mediante los permisos de Android.

2. Programación segura

Aproximadamente 50% de todos los errores (o vulnerabilidades) ocurren a nivel del código de programación (Shahriar y Zulkernine 2012). Con el enfoque de programación segura se pretende proporcionar apoyo para la implementación de programas libres de vulnerabilidades, la cual puede ser considerada como la primera línea de defensa para evitar brechas en la seguridad del programa.

De acuerdo con las vulnerabilidades investigadas (The MITRE corporation, 2011), se consideraron aquellas que pueden afectar a las aplicaciones desarrolladas para Android, entre las que destacan SQL Injection, OS Command Injection, falta de autenticación y autorización para datos o funciones críticas o asignación de permisos incorrectos.

2. 1. Validación de entrada

Validar las entradas de datos del usuario significa verificar que lo que haya introducido contenga únicamente caracteres válidos que son esperados como una respuesta legítima.

Mediante el uso de esta técnica se puede mitigar ataques como transversamiento de rutas, inyección de comandos SQL (Structured Query Language) y de sistema operativo, por mencionar algunos.

La forma de realizar la validación de entrada es crear métodos que eliminen todos los caracteres que son inválidos o que verifiquen la existencia de ellos antes de ser procesados.

2. 1. 1. Verificación de caracteres inválidos

Mediante el uso de la verificación de datos de entrada, una aplicación puede saber si un usuario ha ingresado cadenas con caracteres correspondientes a los elementos usados en comandos de SQL o del sistema operativo que podrían llevarla a realizar comportamientos inadecuados.

El método de verificación de datos de entrada únicamente va a revisar que la información ingresada por el usuario sea válida, mas no realizará correcciones sobre las cadenas.

La primera forma para verificar una cadena es mediante el uso de gramáticas. En este caso, se debe considerar a todos los

caracteres que se juzguen inválidos. Para lograr este objetivo se usará el método estático pattern.matches, el cual recibe como parámetro la expresión regular y la cadena a validar, y va a regresar un booleano informando si la cadena cumple con la gramática.

2. 1. 2. Corrección de cadenas inválidas

A diferencia de la verificación de cadenas, la corrección de cadenas tiene el objetivo de procesar la cadena que haya introducido el usuario, pero para tener una secuencia de comandos o caracteres inválidos es necesario eliminar cada uno de esos caracteres.

a) *Eliminación de caracteres*

Mediante este método se eliminan todos los caracteres que sean considerados como peligrosos o que son parte de los elementos de los comandos como SQL o del sistema operativo. El resultado es una cadena, la cual contendrá puros caracteres válidos.

El método para lograr el objetivo es el *replaceAll* de la clase *String*, el cual recibe como parámetros una expresión regular y una cadena con los que va a remplazar cada coincidencia, que en este caso es una cadena vacía: recibe una y va a regresar otra eliminando todos los caracteres detectados por la expresión regular, que en este caso hace que coincidan todos los signos de puntuación.

b) *Escape de caracteres*

Los métodos de escape de caracteres son comúnmente usados en ambientes web y consiste en sustituirlos por elementos que los representen de otra forma; por ejemplo, sus valores en numéricos ASCII (American Standard Codefor Information Interchange) en hexadecimal anteponiendo un signo de porcentaje para su identificación posterior.

En este formato la cadena con comandos inyectados ya puede ser procesada por el sistema sin que cause comportamientos inadecuados en la base de datos o en la aplicación.

Cabe mencionar que al utilizar este método para escapar la información, también debe existir un método que la desescape para que pueda ser interpretada por el usuario.

2. 2. Hardcodeo de credenciales

La información sensible que se encuentre directamente almacenada en el código fuente del programa como direcciones, nombres de usuario, contraseñas o llaves criptográficas representa un agujero de seguridad, ya que un atacante puede obtener esta información mediante el uso de herramientas de ingeniería inversa.

Para solucionar este problema es necesario tener toda la información sensible almacenada en archivos externos que al momento de ser requerida debe leerse, y una vez que ya no es necesaria deberá destruirse para evitar que puedan ser extraídas de la memoria RAM mediante un análisis forense o con herramientas de debug.

Aunque este mecanismo ayuda a evitar la fuga de información, en Android existe un método más eficiente para evitar el *hardcodeado* de información sensible mediante el uso del archivo strings.xml (contiene todos los recursos por defecto de la aplicación como su nombre, título de la actividad, etc.). El programador también puede agregar valores constantes que se utilicen en la aplicación.

2. 3. Autenticación

Mediante la autenticación se puede saber si el usuario que va a utilizar los recursos o datos almacenados del dispositivo, es legítimo. El objetivo es lograr la confidencialidad.

Para realizar una correcta práctica de programación segura es necesario manejar una contraseña maestra para proteger la información sensible al acceso de usuarios indeseados.

Se tendrán métodos y variables que aseguren una autenticación adecuada del usuario, así como el cierre correcto cuando el usuario abandone la aplicación.

2. 4. Autorización

Suponiendo que un usuario posee una identidad en la aplicación, la autorización es el proceso de determinar si puede acceder a un recurso basado en sus privilegios, permisos u otras especificaciones de control de acceso que se aplican al recurso.

En caso de que la aplicación sea usada por múltiples usuarios, es necesario tener un control de los recursos a los que tenga permitido manejar.

Una forma de autorización es mediante el uso de bases de datos. Android provee soporte para SQ Lite, que se puede crear directamente y la aplicación puede acceder a ella para determinar cuáles son los recursos que posee cada usuario.

La autorización puede trabajar de forma más fuerte y efectiva si se combina con la autenticación, dado que con este mecanismo se puede saber si un usuario es quien dice ser, una vez que se ha firmado, con la autorización se sabe cuáles son los recursos que puede usar.

2. 5. Asignación de permisos

Con la finalidad de proteger a los usuarios de Android, el acceso a los recursos del teléfono y datos de usuario están restringidos a las aplicaciones mediante el uso de permisos. A una aplicación se le deben asignar éstos para el uso de los recursos como la cámara, el micrófono o el Log de llamadas, y que además se muestran al usuario a la hora de instalar la

aplicación y de esa forma informar qué recursos va a usar. Si el usuario no está de acuerdo con la solicitud de recursos, puede cancelar la instalación.

Al momento de la instalación en los permisos, se proporciona a los usuarios el control sobre su privacidad para reducir el impacto de errores y vulnerabilidades en la aplicación. Sin embargo, el sistema de permisos no es efectivo si los desarrolladores solicitan más permisos de los que la aplicación necesita. Excederse expone al usuario al manejo de privilegios innecesarios que incrementa las vulnerabilidades.

Para complementar las prácticas de programación segura con la asignación de permisos, es necesario que los programadores no usen más permisos de los que necesitan.

Figura 1. Prueba de SQL Injection.

Fuente: elaboración propia.

Figura 2. SQL Injection mitigado validando la entrada.

Entrada	Acceso no autorizado	Detectado por los métodos
‘ OR ‘1’=’1	Sí	Sí
admin’--	Sí	Sí
‘ OR 1--	Sí	Sí
‘ OR id=1--	Sí	Sí

Fuente: elaboración propia.

3. Resultados

3. 1. Validando la entrada de los usuarios

Para demostrar estos métodos de programación segura se desarrolló una aplicación con dos entradas de datos, una para el usuario y otra para la contraseña y cuatro botones. Cada botón realiza una práctica segura exceptuando uno, que realiza una verificación normal.

El primer botón verificará los dados ingresados con la base de datos sin ningún mecanismo de validación. El segundo validará si la cadena tiene caracteres inválidos. El tercer botón realiza el escape de la cadena, mientras que el cuarto la limpia.

A continuación se coloca en los campos de texto el comando de inyección de SQL (‘OR ‘1’=’1) (OR: comando de SQL que muestra registros cuando la primera, segunda o ambas condiciones se cumplen) y se realiza la verificación en la base de datos. El comando pudo acceder a la base de datos obteniendo la información del primer registro encontrado en la base de datos (figura 1).

Una vez demostrada la vulnerabilidad de inyección de SQL en Android, se comprueban las prácticas de programación segura, las cuales lograron que este comando no tuviera efecto, por lo que la aplicación no regresó ningún registro de la base de datos evitando el acceso no autorizado (figura 2).

3. 2. Validando al usuario y sus permisos

En la aplicación desarrollada se combinaron los factores de autenticación y autorización para crear un modelo de seguridad más robusto. Para iniciar se cuenta con una pantalla de inicio, la cual pide al usuario que se autentique. Es importante destacar que los nombres de usuario y contraseñas ya se registraron previamente. Si no está registrado, manda un mensaje de error; en caso contrario, si se encuentra en la base de datos, pasa a la siguiente actividad (pantalla) donde se encuentran los botones con las funciones que puede realizar.

Una vez que la persona se ha autenticado, la actividad manda los permisos de dicho usuario a la segunda actividad mediante un intento de Android, y de esa forma la segunda actividad asigna los recursos para ese usuario (figura 3).

3. 3. Protegiendo datos sensibles

Por último se muestran los resultados de las pruebas realizadas al uso de credenciales *hardcodeadas*. Para esta práctica,

se realizó una aplicación que presenta tres credenciales en distintos sitios: *hardcodeadas* en el código fuente, almacenadas en un archivo externo y almacenadas en el archivo strings.xml de Android.

Posteriormente se realizó un ataque de ingeniería inversa en la aplicación para de-compilar el código y encontrar estas credenciales.

3. 3. 1. Harcodeadas en el código fuente

Tras aplicar ingeniería inversa a la aplicación, se obtienen los archivos .class, por lo que sólo falta realizar el de-compilado de las clases mediante el comando javap –c a la clase principal y como resultado las primeras credenciales *hardcodeadas* (figura 4).

3. 3. 2. Archivo externo llamado credenciales.data

Para esta prueba se debe navegar por los archivos a través de la ADB (Android Debug Bridge) para localizar el archivo credenciales.data.

Acceder a estas carpetas no está permitido por un usuario común, ya que si se intenta listar los documentos y carpetas se marcará el error permiso denegado, al igual de cualquier instrucción que intente mostrar un archivo.

Sin embargo, en un dispositivo Android que se encuentre *rooteado* es posible visualizar el contenido de las carpetas al igual que visualizar el contenido de los archivos.

Por lo que también es posible obtener las credenciales en dispositivos que se encuentren *rooteados* (figura 5).

3. 3. 3. Almacenadas en el archivo strings.xml de Android

Tras realizar el análisis a los archivos de-compilados, lo único que se pudo encontrar del archivo strings.xml son los nombres de las variables, mas no se pudo obtener ninguna credencial o información sensible (figura 6).

Conclusiones

Como se demostró, el uso de prácticas seguras evita que las aplicaciones que se desarrollen tengan vulnerabilidades que permitan a los atacantes obtener información confidencial de los datos del usuario.

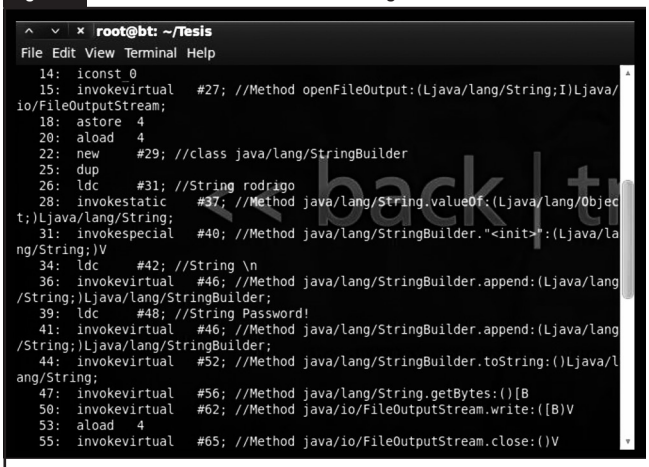
Las prácticas de programación segura son métodos que deben ser aplicados no sólo a aplicaciones desarrolladas en Android, sino en general para asegurar que los usuarios tengan la confianza de que la información almacenada en sus computadoras y dispositivos móviles no pueda ser robada, alterada o destruida por gente maliciosa.

Cabe mencionar que el uso de prácticas de programación segura puede ser tan simple como agregar un método en la clase que puede ser llamado en cualquier momento por cualquier otro, y que el tiempo en el que se procesa puede ser despreciable, dependiendo de la complejidad del algoritmo. Los métodos expuestos muestran una complejidad igual al número de caracteres contenidos en la cadena, por lo que se puede considerar que el uso de recursos es mínimo a diferencia del uso de otros dispositivos de seguridad como *firewalls*, IPS (Intrusion Prevention System), por mencionar algunos, en donde la cantidad de recursos utilizados es mucho mayor y de forma constante.

Figura 3. **Pantallas de autenticación y autorización.**

Fuente: elaboración propia.

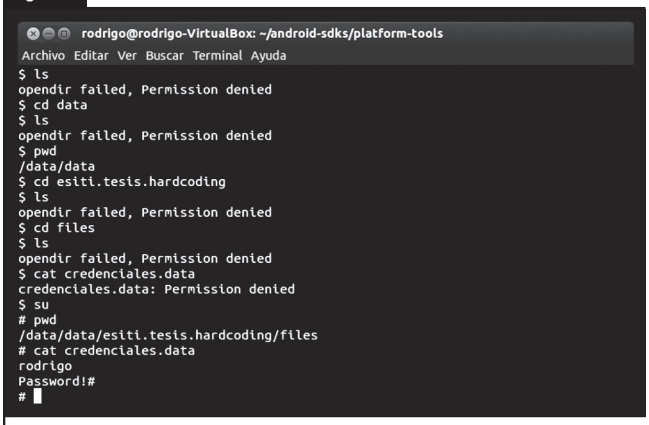
Figura 4. **Credenciales *hardcodeadas* en el código fuente.**



Fuente: elaboración propia.

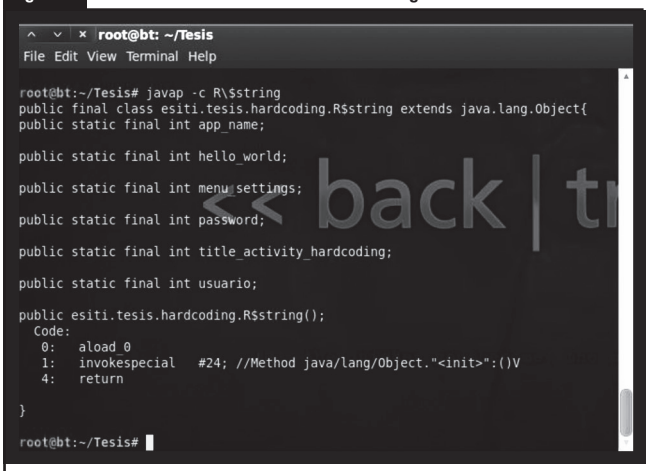
Los métodos mostrados no corresponden a una metodología, esto se debe a que no se sigue ningún proceso sistemático ni secuencial para lograr los objetivos, además de que existen muchas formas de mitigar errores con programación segura.

Figura 5. Credenciales *hardcodeadas* en un archivo externo.



Fuente: elaboración propia.

Figura 6. Información localizada en el archivo strings.xml.



Método	Se pudo obtener datos
Código fuente	Sí
Archivo externo	Sí
Archivo strings.xml	No

Fuente: elaboración propia.

Adicionalmente, también es necesario hacer conciencia en los programadores mediante capacitaciones y certificaciones para que estén informados sobre dichas prácticas con la finalidad de que el porcentaje de aplicaciones vulnerables sea cada vez menor. Por estas razones, lo que queda por hacer como trabajo futuro es adaptar estos métodos dentro de la metodología de programación para dispositivos móviles.

Prospectiva

Mediante el uso de los métodos de programación segura se pueden mitigar vulnerabilidades que permitan el acceso no autorizado a los datos almacenados en el dispositivo, y en el peor de los casos, el control absoluto. Los métodos propuestos pueden ser utilizados para el desarrollo de aplicaciones sin que afecten el rendimiento. Es importante mencionarlo, ya que los recursos de los dispositivos móviles son mucho más limitados que el de las computadoras.

Los métodos propuestos van dirigidos no sólo a las personas que trabajan o estudian en el área de la seguridad informática, también para el desarrollador de *software* que desee proveer de seguridad básica a las aplicaciones que desarrolle con el fin de evitar que aplicaciones vulnerables se conviertan en el blanco de atacantes con el fin de robar la información del usuario.

Con los resultados obtenidos, es posible mitigar las vulnerabilidades con tan sólo pocas líneas de códigos, por lo que no se necesita ser experto en programación o en seguridad para el uso de estos métodos.

El código propuesto puede llegar a generar librerías de Java para su uso constante, de esta forma, al necesitar de estos métodos, lo único que bastaría hacer es importar una de ellas y mandar a llamar el método deseado.

Es significativo considerar estos métodos como parte de una metodología de programación, integrarlos con las ya existentes de tal forma que al programar alguna aplicación ya se considere la parte de la seguridad.

Bibliografía

Burns, J. (2008). *Developing secure mobile applications for Android*. Nueva York: ISEC Partners.

Go-Globe. (2012). *Smartphone users in the world*. Disponible en http://www.go-globe.com

Horstmann, C. y Cornell, G. (2006). *Core Java 2. Volumen 1: fundamentos*. (7a ed.). Madrid: Pearson.

Nielsen, W. (2012). *More US consumers choosing smartphones as Apple closes the gap on Android*. Disponible en http://blog.nielsen.com/nielsenwire/consumer/more-us-consumers-choosing-smartphones-as-apple-closes-the-gap-on-android/

Shahriar, H. y Zulkernine, M. (2012). Mitigating program security vulnerabilities: approaches and challenges. *ACM Computing Surveys*, *44*(3).

Sundsted, T. (2001). *Secure your Java apps from end to end. Part 1. Java World*. Disponible: http://www.javaworld.com/jw-06-2001/jw-0615-howto.html.

The MITRE Corporation. (2011). *CWE/SANSTop 25 Most dangerous software errors*. Disponible en https://cwe.mitre.org/top25/.