Analizador de Intents en Android

Joaquín Erario, Christian Rovera, Francisco Bavera

Departamento de Computación
Universidad Nacional de Rí o Cuarto
Río Cuarto, Argentina
pancho@dc.exa.unrc.edu.ar

Resumen Existen numerosos reportes de vulnerabilidades detectadas en el sistema operativo Android. Si bien muchas de estas vulnerabilidades fueron solucionadas rápidamente, se detectó una, que hasta la fecha, no ha sido solucionada: vulnerabilidades por el uso de "Intent" explícitos. Un "Intent" explícito es un método que puede ser utilizado en aplicaciones Android para invocar desde una aplicación a otra aplicación o servicio determinado explícitamente. Esta invocación puede incluir el paso de algún tipo de información (posiblemente sensible o confidencial). Esta forma de invocar Intents puede ocasionar una vulnerabilidad, ya que, la aplicación o servicio que se invoca puede ser modificada (de forma maliciosa o no) y esta modificación puede permitir manipular de manera indeseada la información recibida. Por ejemplo, una aplicación que envía un intent que agregue un contacto de la agenda o un mensaje para publicar en una red social y la aplicación de destino no es la esperada se puede filtrar información sensible o confidencial sin el consentimiento del usuario. En este trabajo se presenta una herramienta para garantizar que esta vulnerabilidad no pueda ser explotada. El enfoque utilizado segue los lineamientos de la técnica conocida como integridad de control de flujo.

Palabras Clave: Integridad de Control de Flujo, Seguridad, Verificación, Lenguajes, Programas.

1. Introducción

En los últimos años, el sistema operativo Android, inicialmente pensado para teléfonos móviles, fue creciendo cada vez más en cuanto a popularidad debido a una de sus mejores características: la libertad.
Es que Android es un sistema operativo completamente libre. Es decir, no hay que pagar absolutamente nada ni para programar en el ni para poder instalarlo en un teléfono. Lo que lo hace muy popular entre desarrolladores, que deben pagar muy poco para lanzar una aplicación, y fabricantes de celulares, que ahorran a la hora de elegir el sistema operativo para el teléfono que quieren lanzar al mercado.

Otra de las ventajas de ser un sistema operativo libre es que cualquier persona puede descargar el código fuente, inspeccionarlo, modificarlo y finalmente compilarlo. Por lo que, como cualquier software libre, es más fácil detectar errores rápidamente y por ende solucionarlos, esto favorece mucho a la seguridad en el sistema operativo ya que las vulnerabilidades que van surgiendo se van reparando constantemente. Sin embargo, hay ciertos aspectos en cuanto a seguridad, que a pesar de esto, aún no han sido cubiertos. Y esta fue nuestra motivación para llevar a cabo este trabajo: Poder solucionar alguna de estas fallas de seguridad a nivel aplicación que a la fecha aún no han sido solucionadas.

Existen reportes de numerosas cantidades de vulnerabilidades detectadas en el sistema operativo Android, pero como se mencionó anteriormente, al ser un sistema de código libre, estas vulnerabilidades son solucionadas rapidamente luego de ser reportadas. Aunque se encontró una que hasta la fecha, no ha sido solucionada: Vulnerabilidades al usar "Intent" explícitos.

Brevemente, un "Intent" es un método que pueden ser utilizados en aplicaciones Android para invocar desde una aplicación a otra. Esta invocación puede incluir el paso de algún tipo de información y puede ser de manera implícita o explícita. En la forma implícita,el programador no indica que aplicación en concreto quiere invocar sino que simplemente indica el tipo de información que quiere que sea procesada y el sistema operativo se encarga de elegir las aplicaciones correctas para que luego el usuario opte por la que desee. Por otro lado, en la forma explicita, el programador indica específicamente que aplicación invocar.

Esta segunda forma de hacer el Intent puede ocasionar una vulnerabilidad, ya que, la aplicación que se invoca (aplicación destino) puede ser modificada (de forma maliciosa o no) y esta modificación puede permitir manipular de manera indeseada la información recibida. Por ejemplo, si tenemos una aplicación que envía un intent que agregue un contacto de la agenda o un mensaje para publicar en una red social y la aplicación de destino no es la esperada se puede filtrar información confidencial.

A continucación se presenta una breve introducción a Android, seguido de una descripción de algunas vulnerabilidades reportadas. Luego, se describe brevemente la técnica que motivó este trabajo: Integridad de Control de Flujo. Seguido de la presentación de la herramienta. Para finalizar se presentan las conclusiones.

2. Android

Android es un sistema operativo basado en Linux, diseñado principalmente para dispositivos móviles con pantalla táctil como por ejemplo tablets y smarthphones. Fue desarrollado por Android Inc. con el respaldo económico de Google que en el 2005 termina comprando a la empresa. Finalmente, en octubre del 2008 se vendió por primera vez un celular con este sistema operativo.

2.1. Arquitectura de Android

Los componentes principales de Android son cinco:

- Aplicaciones: Este primer componente o nivel, esta compuesto por el conjunto de todas las aplicaciones instaladas en una máquina Android y deben correr en la máquina virtual Dalvik para "garantizar" la seguridad del sistema. Generalmente las aplicaciones Android están escritas en Java aunque también se pueden programar en C++ utilizando el kit de desarrollo Android NDK (Native Development Kit).
- Marco de trabajo de aplicaciones: Los desarrolladores tienen acceso a los mismos APIs del framework que usan las aplicaciones base (sensores, barra de notificaciones, servicios, etc...). Esta capa está diseñada para simplificar la reutilización de componentes. Cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de estas (respetando siempre las reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- Bibliotecas nativas: Android incluye además un conjunto de librerías de C/C++ que son usadas por varios componentes del sistema. Y son expuestas también a los desarrolladores por medio del marco de trabajo de aplicaciones. Se pueden destacar bibliotecas como: System C library, Media Framework, etc..
- Runtime de Android (Máquina Virtual Dalvik): Debido a las limitaciones de memoria y procesador de los dispositivos móviles donde ha de correr Android no fue posible utilizar la máquina virtual estándar de Java para la ejecución de aplicaciones, Google debió crear una nueva máquina virtual Dalvik que funcione mejor ante estas limitaciones. Algunas características de la máquina virtual Dalvik que ayudan a optimizar recursos son:
  - Ejecuta ficheros Dalvik ejecutables (ficheros con extensión .dex). Este formato está optimizador para ahorrar memoria.

- Basado en registros, en lugar de estar basada en una pila.
- Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina.
- Delega al kernel de Linux algunas funciones como threading y el manejo de la memoria a bajo nivel.

- Núcleo Linux: El núcleo de Android está formado por el sistema operativo Linux. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos. Es la única capa dependiente del hardware ya que actúa como capa de abstracción entre el hardware y la pila de protocolos.

2.2. Aplicaciones

Las aplicaciones de Android, como ya mencionamos, se pueden programar tanto en C++ como en Java. Cuando se compilan los programas, el Kit de Desarrollo nos arma un archivo .APK. Los programas compilados son programas en Bytecode para la máquina virtual Dalvik.
Los archivos .APK son los que nos permiten instalar una aplicación en el celular. Contienen una serie de instrucciones a ejecutar y además otros recursos como imágenes, sonidos y archivos .xml cómo por ejemplo el AndroidManifest.xml.
Cada APK se asocia a un proceso único, que proporciona el ambiente de ejecución de los componentes. De los cuales, uno es el componente inicial del programa.
Cuando se ejecuta una aplicación se le asigna un proceso Linux y un único hilo de ejecución (thread), así todos sus componentes corren sobre el mismo proceso y thread.

2.3. Seguridad en Android

La seguridad es un aspecto clave en cualquier sistema. Si nos descargamos una aplicación maliciosa a nuestro celular, esta podría robarnos contactos, enviar sms por su propia cuenta y sin nuestra autorización o hasta incluso saber donde estamos posicionados físicamente utilizando datos del gps del sistema.
Android propone un esquema de seguridad para proteger a los usuarios, sin tener que imponer un sistema centralizado en alguna empresa (como si lo hace el sistema operativo de iPhone por ejemplo). Este esquema esta basado en tres pilares fundamentales: la seguridad de Linux, la firma digital de la aplicación y un modelo de permisos de acceso a partes del

sistema. Este último componente establece que muchas decisiones importantes relativas a la seguridad recaigan en el usuario final.

Seguridad Linux Como se comentó en la sección anterior, Android está basado en Linux, por lo tanto, se aprovecha la seguridad que incorpora este sistema operativo. De esta manera Android puede impedir que las aplicaciones tengan acceso directo al hardware o interfieran con recursos de otras aplicaciones.

Firma digital de las aplicaciones Las aplicaciones deben ser firmadas con un certificado digital que identifique al autor. Esto nos permite ver que aplicaciones se supone que sean más confiables que otras. Además, la firma digital nos garantiza que los archivos de una aplicación no han sido modificados. Si se opta por modificar la aplicación, está deberá ser firmada nuevamente. Generalmente este certificado digital no es firmado por alguna autoridad de certificación.

Modelo de permisos: Android Manifest Si queremos que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad del sistema necesitamos utilizar un modelo de permisos, de forma el usuario puede conocer los riesgos antes de instalar la aplicación. Para lograr esto, se utiliza el archivo Android Manifest.
Cada aplicación de Android debe contener un archivo AndroidManifest.xml (con exactamente este nombre) en su directorio raíz. Este archivo le presenta información esencial sobre la aplicación al sistema operativo, información que el sistema debe tener antes de poder correr el código de la misma.Entre otras cosas, el AndroidManifest.xml tiene:

- El nombre del paquete Java que sirve como identificador único para las aplicaciones del sistema operativo.
- Componentes de la aplicación.
- Permisos que la aplicación va a solicitar al momento de su instalación. Pueden ser tanto como para acceder a la API o interactuar con otras aplicaciones.
- Bibliotecas con las que debe linkear.
- Nivel mínimo de la API de Android requerido para su funcionamiento.

2.4. Vulnerabilidad por Intent Explícitos

La interacción entre componentes en un sistema Android están dados por pasaje de mensajes llamados Intents. Estos mensajes están formados

por una dirección o nombre de destino e información relacionada con la acción a ejecutar.
Cuando un componente envía un Intent, el receptor del mismo iniciará una actividad, mensaje broadcast o servicio que ejecutarán una acción.
Hay dos tipos de intents posibles, por un lado tenemos los intents implícitos donde no se indica el destino del mensaje, sino que solo se indica el tipo de mensaje (texto plano por ejemplo) y entonces todas las aplicaciones que declaren en su AndroidManifest que pueden recibir ese tipo de mensajes, aparecerán en una lista para que el usuario decida que aplicación es la que recibirá el mensaje. Un claro ejemplo de esto, es cuando tenemos más de un navegador web instalado en el celular y hacemos click en algún link que nos lleve a una dirección web. En ese momento, nos aparecerá una lista con todos los navegadores instalados de la cual debemos elegir cuál queremos usar.
Por otro lado, tenemos los intents explícitos, aquellos a los que si queremos indicarle específicamente a quién va dirigido el mensaje, es decir, el nombre del paquete de la aplicación receptora.
La posible vulnerabilidad de este tipo de intents, radica en que al indicar solo el nombre del paquete de la aplicación, es posible cambiar la aplicación receptora por otra con el mismo nombre y cuyo comportamiento no es el que nosotros deseamos.

Para entender mejor esta vulnerabilidad veamos un simple ejemplo: dada una agenda de contactos que posee la funcionalidad de enviarle los contactos favoritos a otra aplicación de mensajería (Similar a aplicaciones como Viber o Whatsapp). Esta aplicación de mensajería, recibe los contactos de esta nueva agenda y los utiliza para poder enviarle y recibir mensajes de ellos.El problema esta en qué si se cambia esta aplicación de mensajería, puede ocurrir que en lugar de recibir los contactos y guardarlos para luego poder comunicarse con ellos, los envié a una dirección de correo y así nos robe los contactos de la agenda.

3. Integridad del control de flujo

Las computadoras con frecuencia son objeto de ataques externos que apuntan a controlar el comportamiento de algún software. Generalmente estos ataques llegan como datos a través de un canal de comunicación normal y, una vez que residen en la memoria del programa, explotan defectos preexistentes en el software. Explotando dichos defectos, el atacante desestabiliza y toma control del comportamiento del software. Por ejemplo, un desbordamiento del buffer en una aplicación puede resultar en una

llamada a una función sensible del software, posiblemente una función que la aplicación no fue diseñada para su uso.
Las políticas de la integridad de control de flujo [9,10] dictan que la ejecución del software deben seguir un camino de su grafo de control de flujo creado de antemano. El grafo en cuestión puede ser definido por análisis de código, análisis de los ejecutables o mediante perfiles de ejecución.
Hay varias formas de llevar dicho control, pero en el presente trabajo nos centraremos en la instrumentación de código debido a que es el método utilizado en nuestra herramienta.

3.1. Instrumentación de código

La instrumentación de código es la inserción de código con el fin de asegurar la perfomance de un sistema, diagnosticar errores y escribir información del flujo del programa. La instrumentación le otorga a un programa la de incorporar:

- Seguimiento de código: recibiendo mensajes informativos acerca de la ejecución de una aplicación en tiempo de ejecución.
- Depuración y un estructurado manejo de excepciones: búsqueda y corrección de errores de programación en una aplicación bajo desarrollo.
- Profiling: un medio por el cual los comportamientos dinámicos de los programas pueden ser medidos durante un ejecución de prueba con una entrada representativa. Esto es útil para probar propiedades de un programa que no puede ser analizadas estáticamente con suficiente precisión, como por ejemplo análisis de aliasing.
- Contadores de rendimiento: componentes que permiten el seguimiento de la performance de una aplicación.
- Registro de datos: componentes que permiten el registro y seguimiento de la mayoría de los eventos en la ejecución de la aplicación.

4. La Herramienta

Teniendo en cuenta la vulnerabilidad explicada en la sección anterior sobre los intents explícitos, se decidió crear una herramienta que pudiera de alguna forma solucionar este posible problema en las aplicaciones. La herramienta tiene la funcionalidad de controlar que el flujo de la información mediante intents sea el requerido por el usuario. Para eso se introduce en la aplicación que envía la información, antes de cada invocación de intent, una verificación para garantizar que efectivamente el mensaje sea atrapado por la aplicación deseada.

Para lograr esto se trabajó sobre el control de flujo de la aplicación, más precisamente se instrumentó código a la aplicación que envía el intent. A grandes rasgos, este código es una función que toma como parámetro el nombre de una aplicación y un identificador único (id) de la misma. Este id es obtenida calculando un código hash para luego checkear en una lista confiable presente en el celular si el id de esa aplicación es el mismo que el id actual de la aplicación con ese nombre. De esta forma se detecta si el receptor del mensaje fue modificado o no.

La herramienta y su documentación esta disponible en `https://sourceforge.net/projects/`

4.1. El Proceso

A continuación se detalla el proceso realizado por la herramienta para generar las verificaciones. Este proceso se realiza cuando la aplicación es instalada en el dispositivo Android.

Conversión a código Jasmin En primer lugar, cuando el usuario elige el programa que desea verificar que sus intents sean correctos, la herramienta transforma el archivo .APK (el programa a instalar en el dispositivo) a código Jasmin [6] (una representación intermedia de código bytecode), para eso invoca una serie de scripts (basados en la herramienta Dex2Jar [3]:

- **d2j-dex2jar.sh**: Extrae el classes.dex del apk y lo convierte en un archivo .jar.
- **d2j-asm-verify.sh**: Verifica que el .jar creado sea correcto.
- **d2j-jar2jasmin.sh**: Transforma el código .jar en código Jasmin.

Obtención de la Id de la aplicación Para poder controlar que la aplicación receptora del mensaje o intent no sea modificada es necesario asignarle una id única. Para esto se creó un script en Python que obtiene el código hash de un archivo, de esta forma se le obtiene el hash al archivo instalador (.apk) de la aplicación receptora que si recibe un mínimo cambio entonces su id cambiará. Para obtener el código hash del .apk se utiliza la librería hashlib de Python.

Inserción de las Verificaciones Dinámicas Se introducen las verificaciones dinámicas que chequean que el id de cada aplicación en tiempo de instalación se correspondan con el id de la aplicación en tiempo de ejecución. Estas verificaciones se introducen en el código Jasmin.

Para esto se implementó un método Check en Jasmin que toma como parámetros el nombre de la aplicación que se va a checkear y el id o código hash del mismo que espera. Luego busca en un archivo auxiliar instalado en el dispositivo donde se encuentran los nombres de las aplicación instaladas con nuestra herramienta y su código hash actual, que si el nombre de la aplicación pasada como parámetro coincide con alguno del archivo, sus ids también deben coincidir.

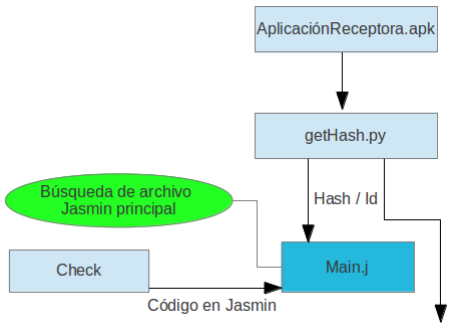

Figura 1. Instrumentación de código en el archivo principal

Finalmente se introduce antes de cada invocación a un intent que se desea controlar la llamada a esta función. En pseudocódigo, es una simple llamada a la función check: `check(nombrePaquete,IdEsperado)`.

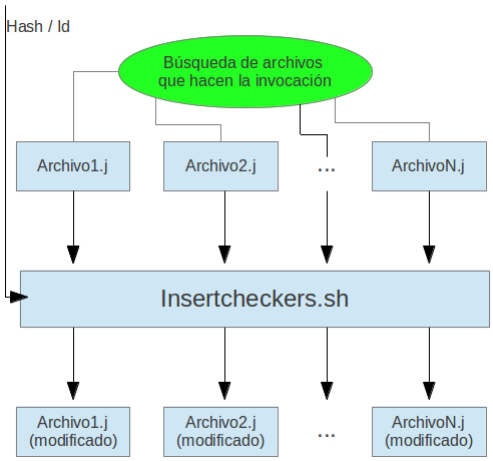

Figura 2. Script insertcheckers.sh: Instrumentación de código en el resto de los archivos que invocan a la aplicación receptora

Reconstrucción del .APK Una vez finalizada la inserción de código solo resta reconstruir el .APK para ello se utilizan algunos scripts de la herramienta Dex2Jar:

1. **d2j-jasmin2jar.sh**: Reconstruye el .jar.
2. **d2j-asm-verify.sh**: Nuevamente se verifica que el .jar sea correcto.
3. **d2j-jar2dex.sh**: Se reconvierte a código dex.

Posteriormente se crea una copia de seguridad del .apk y se le reemplaza el dex original con el modificado.
Por último, como Android necesita que los .apk estén firmados para que te los permita instalar, se vuelven a firmar.

La Verificación El proceso de verificación en tiempo de ejecución, es decir, la verificación cuando se ejecuta la aplicación es sencilla. El código instrumentado ejecuta un método qie verifica si el id actual del intent implícito que se ejecutará a continuación es igual al id en tiempo de instalación. En caso de no coincidir no se permite la ejecución del intent.

4.2. Ejemplos

En primer lugar se implementó una agenda de contactos donde se almacenan números telefónicos que pueden ser usadas por otra aplicaciones

para el envío de mensajes.
Si seleccionamos un contacto de la agenda, automáticamente este se lo envía a la aplicación de mensajería quién almacena el número y luego de que se escriba el mensaje, envía su contenido al destinatario.

Como segundo ejemplo se implementó una aplicación de logueo a homebanking de distintos bancos. La aplicación permite ingresar usuario y contraseña y elegir a que banco se desea acceder para luego enviar al modulo del banco seleccionado los datos de sesión y consultar saldo o movimientos. Este módulo podría ser modificado para que los datos del cliente se envíen ocultamente por email, o directamente que se hagan transferencias no deseadas.

Con estas dos aplicaciones se pudo apreciar que el enfoque es factible para reforzar la integridad de control de flujo en aplicaciones Android con la herramienta presentada. En abos casos se detectó cuando las aplicaciones fueron cambiadas y/o modificadas.

5. Conclusiones

El desarrollo de esta herramienta se puede fundamentar fácilmente revisando las vulnerabilidades que están presentes actualmente en el sistema operativo Android y en la presunción (fundamentada en la experiencia hasta el momento) de que se detectarán nuevas vulnerabilidades en el futuro. Por ejemplo, es posible escalar privilegios y modificar una aplicación, que se encuentre presente en el celular, receptora de algún intent o mensaje explícito. También puede ocurrir que la aplicación sea modificada por una actualización de la aplicación (una actualización que implique un cambio defuncionalidad no deseado).

La herramienta presentada permite controlar por fuera del sistema operativo y darle más garantía a los programadores de Android, de manera autómatica, verificando que los intent tengan el destinatario esperado. En otras palabras, la herramienta garantiza la integridad del control de flujo relacionada con los intents implicitos.

La solución presentada en el presente trabajo tiene la desventaja que no es una solución a nivel sistema operativo por lo que se necesita sí o sí de la herramienta realizada. Esta desventaja no es tan negativa por el hecho de que son escasas las veces que se requieren realizar este tipo de checkeos por lo que no es tan molesto tener que recurrir a la herramienta para lograrlo. Además necesita también de otra base confiable, en este caso de un instalador de aplicaciones alternativos que vaya actualizando y agregando los ids de las aplicaciones instaladas en un archivo

confiable. Por otro lado, la herramienta tiene la ventaja que se puede modificar fácilmente o incluso utilizarla para realizar otro tipo de chequeos o modificaciones de las aplicaciones.

5.1. Trabajos futuros

Los trabajos más relevantes que se deben realizar para mejorar la herramienta y extender su funcionalidad son:

- extender el método de verificación para incluir los intent implícitos. En Android cuando se hace un intent implícito se crea una lista de aplicaciones que pueden manejar el tipo de dato enviado (por ejemplo, texto plano) para que el usuario elija la aplicación que reciba el intent. Para garantizar la aplicación/es autorizadas a recibir determinado intent implícito es necesario poder generar la lista dinámicamente (y no tomar la lista generada autómaticamente por Android).
- La herramienta esta programada en bash de Linux y en Python ambos estan disponibles para Android, por lo que es factible hacerla nativa para Android. Sin embargo habría que modificar o sustituir el instalador de aplicaciones de Android por un instalador "seguro" para darle mayor grado de confiabilidad.
- Es necesario estudiar la posibilidad de implementar este enfoque para garantizar la integridad del control de flujo desde el receptor de los intent. Es decir, que el receptor sea el que verifique si el intent fue disparado por una aplicación autorizada.

Referencias

1. Web oficial de desarrollo para Android, *http://developer.android.com/index.html*
2. Android Argentina, *http://androidargentina.com.ar/*
3. Web oficial de Dex2jar, *http://code.google.com/p/dex2jar/*
4. ¿Qué es Android?, *http://www.xatakandroid.com/sistema-operativo/que-es-android.*
5. Wikipedia: Android, *http://es.wikipedia.org/wiki/Android*
6. Web de Jasmin, *http://jasmin.sourceforge.net/about.html*
7. Android Security, *https://source.android.com/tech/security/*
8. Introduction to Instrumentation and Tracing *http://msdn.microsoft.com/en-us/library/aa983649%28VS.71%29.aspx*
9. Martín Abadi; Mihai Budiu; Úlfar Erlingsson y Jay Ligatti. *Control-Flow Integrity Principles, Implementations, and Applications.* ACM Journal Vol V, Febrero 2007.
10. Chao Zhang; Tao Wei; Zhaofeng Chen; Lei Duan; László Szekeres; Stephen McCamant; Dawn Song y Wei Zou. *Practical Control Flow Integrity & Randomization for Binary Executables.*