I. Agudo, J. A. Onieva, and D. Merida, "Distribución segura de componentes software basada en OpenID", *XI Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2010)*, 2010.
NICS Lab. Publications: `https://www.nics.uma.es/publications`

Distribución segura de componentes software basada en OpenID

Isaac Agudo, Jose A. Onieva, Daniel Merida
Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga
Email: {isaac,onieva,dmerida}@lcc.uma.es

***Resumen*—En la actualidad, cada vez son más frecuentes los ataques software mediante la utilización de malware o sustitución de programas (o componentes) en los repositorios a los cuales los usuarios finales (o máquinas) acceden. Esta situación se ve de alguna manera acentuada con el dinamismo existente en la programación y ejecución de estos componentes, en la que distintos desarrolladores pueden participar para desplegar un determinado servicio o parte de él.**

Por ello, en este artículo se presenta una solución para la distribución de código de forma segura usando OpenID y firmas con certificados de clave pública de corta duración. De esta forma, se consigue un compromiso de seguridad que permite distribuir código firmado sin la necesidad de que los desarrolladores dispongan a priori de un certificado específico. Presentamos además algunos detalles acerca de la implementación realizada para hacer realidad este diseño.

I. INTRODUCCIÓN Y OBJETIVOS

La distribución segura de componentes software es en la actualidad un factor crítico para todos los usuarios, y no sólo para las empresas desarrolladoras. Cada vez son más frecuentes los ataques software mediante la utilización de malware o sustitución de programas (o componentes software) en los repositorios a los cuales los usuarios finales (o máquinas) acceden. Esta situación se ve de alguna manera acentuada por el dinamismo intrínseco en la programación y ejecución de estos componentes software, en la que distintos desarrolladores pueden participar para desplegar un determinado servicio (o partes de él).

Es muy conocido el imparable aumento del malware en todo el mundo. En especial en nuestro país que, según el informe del primer trimestre de este año 2010 de PandaLabs [1], ocupa la primera posición en el ranking de países con respecto al número de ordenadores infectados. Según estas estadísticas, al menos uno de cada tres ordenadores están infectados por algún tipo de malware. Por ello es especialmente importante bloquear en todo momento la capacidad de propagación del mismo.

Dado el aumento de prestaciones de los dispositivos móviles, existe una aplicación disponible para prácticamente cualquier necesidad, y aquí la integridad del software o la confianza en el desarrollador son objetivos fundamentales a perseguir. La proliferación de aplicaciones disponibles para todo tipo de dispositivos y para los dispositivos móviles en particular, ha provocado por añadidura que sean cada vez más el numero de desarrolladores que distribuyen estas aplicaciones. No todos ellos deben tener el mismo nivel de confianza sino queremos que el malware encuentre una puerta de entrada a nuestros sistemas.

Este escenario se vuelve más importante cuando pensamos en el futuro: Multitud de dispositivos, posiblemente autónomos, interconectados entre sí y conectados al "exterior", siendo así actores del *Internet of Things*, y ejecutando servicios cuyo código y composición puede cambiar de forma dinámica, y en los que la instalación y desinstalación de los distintos componentes que construyen el servicio en ejecución se ha de realizar en tiempo real.

Existen soluciones que permiten decir a nuestro sistema que sólo ejecute componentes firmados y que además esta firma sea confiable (nuestro sistema ha de confiar en alguno de los certificados existentes en el camino de verificación de la firma). Pero no escapa a los usuarios finales (ni a los desarrolladores) que estas soluciones transforman a los sistemas en ocasiones en demasiados restrictivos, haciendo que en muchos casos, los administradores de estos dispositivos terminen por cancelar este tipo de comprobaciones.

Por otra parte, los desarrolladores, se ven abocados a soportar una burocracia digital, para en todo momento, contar con certificados digitales no caducados y firmados por una entidad confiable. Este proceso es relativamente sencillo y natural para grandes empresas desarrolladoras de software. Sin embargo, no ocurre lo mismo para desarrolladores particulares que carecen de medios. En los ambientes móviles el uso de código firmado está muy extendido y aunque algunas plataformas como Symbian proporcionan mecanismos para la creación de certificados de desarrollo [2], este proceso requiere de un registro previo del desarrollador y la validez del código firmado está restringida al teléfono de desarrollo registrado.

Esta situación se agrava en entornos en los que los servicios (programas en ejecución) se componen de forma dinámica y en tiempo real de distintos módulos o componentes, tal y como ocurre en la plataforma OSGi [4]. La plataforma OSGi es un sistema de módulos y servicios para el lenguaje de programación Java que implementa un modelo completo y dinámico de componentes. Las aplicaciones o componentes (en forma de *bundles*) pueden ser instaladas de forma remota, iniciadas, detenidas, actualizadas y desinstaladas sin necesidad de reiniciar. La gestión del ciclo de vida se realiza a través de APIs que permiten la descarga remota de políticas de gestión. El Registro de Servicios permite a los bundles detectar la

adición o la supresión de los servicios, y adaptarse en consecuencia. Las especificaciones OSGi se han movido más allá del enfoque original, y ahora se utilizan en aplicaciones que van desde teléfonos móviles al IDE de Eclipse [5], pasando por un amplio espectro de áreas de aplicación tan heterogéneas como pueden ser los automóviles, la automatización industrial y la automatización de edificios, dispositivos móviles como las PDA's, el ocio y el entretenimiento, la gestión de flotas o los servidores de aplicaciones.

Así, el escenario que se presenta (ver Figura 1 para más detalles) es el de un dispositivo que soporta OSGi que con el objeto de lograr la ejecución dinámica de componentes cuenta con conexión constante a distintos *repositorios* OSGi de los que cargará los distintos bundles de acuerdo a las dependencias y necesidades de los bundles actuales en ejecución. Nuestra solución ha de tener en cuenta los puntos de vista y necesidades de este dispositivo y de los desarrolladores para proporcionar integridad y confianza a la hora de ejecutar estos componentes a la par que facilidad de uso y sencillez a la hora de subir estos componentes a los repositorios.

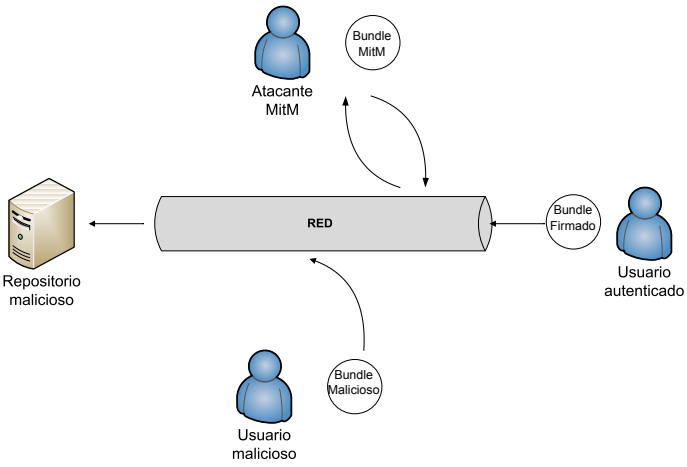

Figura 1. Amenazas en el proceso de desarrollo de bundles

II. REQUISITOS

Las amenazas a la seguridad para el despliegue de los bundles [6] pueden ser de tres tipos:

1. La presencia de repositorios maliciosos para la publicación de bundles.
2. Ataques "Man in the Middle", de forma que un atacante pueda modificar un bundle o sustituirlo completamente por otro durante la carga o la descarga.
3. La posibilidad de que un atacante acceda al repositorio de bundles o a la plataforma cliente para modificar los componentes almacenados en ellos.

Tras el análisis del escenario y sus principales amenazas, los principales requisitos que pueden extraerse del mismo son los siguientes:

1. Sencillez.
2. Movilidad / portabilidad.
3. Autenticación del desarrollador.
4. Integridad de los bundles.
5. Generación de firmas seguras.

Se pretende conseguir una plataforma **sencilla** y robusta para la firma de componentes software. El impacto para el desarrollador (en términos de tiempo y complejidad) que sube un bundle a un repositorio OSGi ha de ser mínimo, sin que ello dificulte el proceso de verificación por parte de los usuarios[1] o dispositivos.

Al no requerirse el uso de certificados personales para la distribución de los bundles se permite cierta **movilidad** de los desarrolladores, ya que pueden subir los bundles a los repositorios OSGi simplemente accediendo al repositorio con su usuario y contraseña habitual.

Es importante considerar que en el modelo de desarrollo de software libre, el conjunto de desarrolladores es dinámico y que la implicación de estos en el proyecto es variable. Por tanto se tiene que dar cabida no solo a los desarrolladores estables sino también a los eventuales que solo quieren aportar su granito de arena al proyecto. Por este motivo se define en mecanismo de autenticación para el repositorio basado en federación de identidades, de forma que el repositorio no tenga que **autenticar** directamente al desarrollador sino que sea su proveedor de identidad el que proporcione la información necesaria.

En la solución propuesta se utilizan certificados temporales (de vida corta) generados a partir del conjunto de atributos que definen la identidad de desarrollador proporcionado por el proveedor de identidad para firma el código y por tanto proporcionar **integridad** de éste. Además, al ser certificados temporales, pueden ser utilizados desde cualquier dispositivo independiente de su situación geográfica y evitando la implicación en seguridad que tendría llevar consigo siempre un par de claves pública y privada para la firma de estos componentes.

El uso de una federación de identidad permite que el desarrollador solo tenga que autenticarse de la misma manera (y con los mismos credenciales) que lo hace en otros servicios para poder almacenar un bundle firmado. Por supuesto, el repositorio OSGi cuenta con políticas de autorización que determinarán la visibilidad de estos bundles para otros usuarios, aunque consideramos este proceso fuera del ámbito de este trabajo.

Para la **generación de firmas de forma segura** por parte del usuario, esta plataforma permitirá al usuario generar el par de claves de forma local y será el repositorio OSGi quien haga la función de tercera parte confiable que certifique esas claves, y que certifique que su usuario legítimo es quien dice ser. Este esquema difiere del sistema de distribución de paquetes de Debian [3], donde los repositorios pueden firmar sus paquetes permitiendo su posterior verificación, ya que en nuestro esquema para la verificación de un bundle el filtro de confianza se hace a nivel de desarrollador y no de repositorio.

[1]Estos usuarios, potencialmente, pueden ser a su vez desarrolladores que trabajen de manera colaborativa en la implementación de bundles.

III. COMPONENTES DE LA PLATAFORMA

III-A. Bundles

Un bundle no es más que un fichero JAR (Java Archive) que contiene además un conjunto de metadatos que especifican las características del componente. Estos metadatos están incluidos dentro del archivo `MANIFEST.MF`. Por tanto:

Bundle = archivo `JAR` + `MANIFEST.MF` con metadatos

Es necesario proporcionar algún tipo de soporte para el ciclo de vida de los bundles, desde la instalación hasta la ejecución y el borrado, por lo que la seguridad deberá contemplarse a lo largo de todo el ciclo de vida.

III-B. Repositorio OSGi

Se trata del repositorio en el que se almacenan los distintos bundles al que las máquinas que cuenten con plataformas de ejecución OSGI acceden para descargar componentes. De la misma manera, los desarrolladores suben estos bundles una vez testeados o con el objetivo de colaborar en el desarrollo de los mismos.

El administrador del repositorio ha de autenticar a todos los desarrolladores que pretendan almacenar bundles y a su vez facilitar la integridad de estos componentes, de forma que se enlace de manera inequívoca cada bundle con cada uno de sus autores. Para ello, la plataforma proporciona un servicio o autoridad de certificación (CA) cuyo único objetivo es permitir la firma de los mismos.

Ciertamente, este servicio de certificación podría ser externo, y reconocido, pero dado que los certificados han de ser temporales y de una funcionalidad muy reducida, el repositorio puede ejercer esta función, reduciendo así el proceso previo a la firma por parte de los desarrolladores.

III-C. Servidor OpenID

OpenID [7] es un sistema de autenticación digital descentralizado, con el que un usuario puede identificarse en una página Web a través de una URL (o un XRI en la versión actual) y puede ser verificado por cualquier servidor que soporte el protocolo. En los sitios que soporten OpenID, los usuarios no tienen que crearse una nueva cuenta de usuario para obtener acceso. En su lugar, solo necesitan disponer de un identificador creado en un servidor OpenID (OpenID provider), es decir, un proveedor de identidad OpenID (IdP).

Este proveedor de identidad puede confirmar la identificación OpenID del usuario a un sitio que soporte este sistema. A diferencia de arquitecturas SSO (Single Sign-On), OpenID no especifica el mecanismo de autenticación. Por lo tanto, la seguridad de una conexión OpenID depende de la confianza que tenga el cliente OpenID en el proveedor de identidad. Si no existe confianza en el proveedor, la autenticación no será adecuada para servicios bancarios o transacciones de comercio electrónico. Sin embargo, el proveedor de identidad puede usar autenticación fuerte si el proveedor de servicio (Service Provider) así lo requiere.

De entre los atributos que proporciona OpenID (ver [7] para más detalles), se han seleccionado los siguientes para la generación de los certificados temporales:

1. openid.ext1.value.firstname (en adelante, firstname)
2. openid.ext1.value.lastname (en adelante, lastname)
3. openid.ext1.value.email (en adelante, e-mail)
4. openid.op_endpoint (en adelante, ID provider)

Con la elección de OpenID, se pretende evitar la centralización que supone un sistema jerárquico de PKI, que restaría dinamismo a la plataforma e introducirían una dependencia sobre la disponibilidad de la jerarquía de CA para la generación de certificados. En otras palabras, logramos con ello evitar la necesidad de acudir presencialmente a una Autoridad de Registro.

III-D. Applet de firma

Para la firma y creación de los certificados de firma digital se requiere la descarga y ejecución por parte del desarrollador de del applet de firma. El applet de firma solo toma como entrada del usuario el repositorio de destino, así como el bundle que se quiere firmar.

La estructura de clases [9] que presenta el código fuente del applet es la que se muestra en la Figura 2

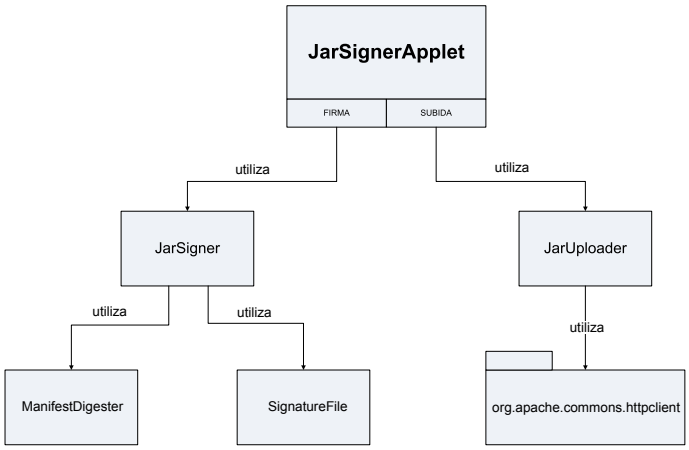

Figura 2. Estructura de clases que utiliza el applet de firma

Como apoyo auxiliar para la firma de bundles se han utilizado dos herramientas proporcionadas por Java: Jarsigner y Keytool.

La herramienta Jarsigner [10] tiene dos objetivos principales: firmar archivos JAR y verificar la firma y la integridad de archivos JAR firmados. Para ello, utiliza información de certificados y claves que obtiene a partir de un almacén de claves (keystore). Un keystore contiene una de base de datos con claves privadas y las cadenas de certificados X.509 que autentican sus correspondientes claves públicas.

La herramienta Keytool [11] se encarga de la gestión de claves y certificados, de forma que permite a los usuarios administrar sus propias claves públicas y privadas y los certificados asociados a éstas para autenticarse a sí mismo frente a otros usuarios o servicios o para realizar verificaciones de integridad mediante las firmas digitales.

Un usuario se puede encontrar con dos tipos diferentes de entradas que formarán parte de un almacén de claves:

1. Claves: Una clave almacenada suele ser una clave privada acompañada de la correspondiente cadena de certificación de su clave pública
2. Certificados de confianza: Los certificados de confianza son aquellos certificados de clave pública de otro usuario o entidad de los que se conoce con certeza la identidad de su propietario

Existen otras opciones de Keytool para importar en el keystore certificados ya existentes, para exportar certificados, para generar certificados autofirmados o para generar solicitudes CSR que serán firmadas posteriormente por una CA, opción ésta de la hemos hecho uso en la implementación.

IV. PROCESOS DE LA PLATAFORMA

El objetivo final de esta plataforma es la distribución segura, mediante un repositorio, de componentes software. Para ello, se utilizará la información de identidad proporcionada por un servidor OpenID[2] para crear unos certificados temporales, que serán firmados por un servicio de CA online del propio repositorio y serán utilizados para la firma del bundle, como paso previo a la carga en el repositorio.

IV-A. Proceso de firma de un bundle

La solución planteada para proporcionar seguridad en los bundles es la firma digital, que por las peculiares características de los bundles, se debe almacenar dentro del bundle, junto al resto de componentes. Los archivos relacionados con la firma de un bundle son los siguientes:

1. Un archivo de manifiesto (Manifest File), que contiene un listado con el valor hash de cada uno de los recursos del bundle.
2. Para soportar varias firmas, la firma digital no se aplica directamente sobre el archivo de manifiesto, sino sobre un archivo para la firma denominado (Signature File) que contiene un valor hash del archivo de manifiesto. Hay un archivo de firmas por cada firmante.
3. La firma digital del archivo de firmas se almacena en un archivo CMS (Content Management System) denominado Block Signature File. La extensión de este archivo es el nombre del algoritmo de firma (.dsa, .rsa, entre otros). Además, este archivo contiene los datos necesarios para la verificación de la firma, como es la cadena de certificados que verifican al usuario firmante, que en nuestro caso será la cadena formada por el certificado temporal creado con los atributos OpenID del usuario y el certificado de clave pública de la CA online del repositorio OSGi.

En nuestra implementación, el orden de estos recursos dentro del bundle es el que aparece descrito anteriormente. Todos estos ficheros se colocarán delante del resto de recursos del bundle. Este orden será uno de los aspectos que se tendrán en cuenta durante el proceso de verificación de un bundle firmado.

[2]En nuestras pruebas se han utilizado con éxito, los servidores OpenID de Google, RedIRIS y Yahoo.

En la Figura 3 se muestra la interacción entre los componentes de la plataforma. El flujo de información en el proceso de firma de un bundle paso por paso sería el siguiente:

- El desarrollador se conecta al servicio de subida de bundles del repositorio (Bundle Upload Service).
- El primer paso será acceder a la URL de descarga del applet. Al acceder a esta URL se comprueba si el usuario accede con atributos OpenID y, en caso de no llevarlos, es redirigido a un servicio WAYF (Where Are You From).
- El servicio WAYF permitirá al usuario seleccionar su proveedor de identidad OpenID, hacia el que será redirigido.
- El usuario introducirá su usuario y contraseña para autenticarse en el servidor OpenID, lo que permitirá que éste servidor envíe los atributos OpenID del usuario hacia la página de redirección, que será la URL de descarga del applet.
- En un segundo acceso a la URL de descarga del applet, si la comprobación de atributos OpenID se realiza con éxito se permite que se inicie el proceso de descarga. En el navegador del usuario, se pedirá la autorización del usuario para la ejecución del applet a partir de la confianza en el certificado del desarrollador del mismo.
- A partir de los atributos OpenID con los que se inicializará el applet, se generan un par de claves pública y privada y la correspondiente solicitud CSR (Certificate Signing Request). Los campos necesarios para las claves se cumplimentarán de la siguiente manera:
  - Common Name: firstname + lastname
  - Organization Unit: e-mail
  - Organization: ID provider
  - Country: ES
- El CSR se envía al servicio de la CA online del repositorio.
- El servicio de la CA online generará un certificado de clave pública a partir del CSR y la clave privada de la CA.
- La CA online envía el certificado de clave pública, junto al certificado de clave pública de la CA si fuera necesario. En el applet, se unen el certificado de clave pública y la clave privada y el usuario ya tiene la capacidad para firmar componentes software con un certificado temporal completo.
- Se realiza el proceso de firma del bundle a partir del certificado temporal obtenido y se envía el bundle al repositorio.

La principal razón para la elección de un applet como elemento central de la plataforma en el proceso de firma es que el par de claves pública y privada se deben generar de forma local en el equipo del usuario (para que el desarrollador realmente confíe en las claves generadas), pero a partir de un software existente en la plataforma.

De esta forma, el usuario se descarga el applet en el navegador y autoriza su ejecución (confiando en su autor) durante el proceso automático de descarga del mismo. La se-

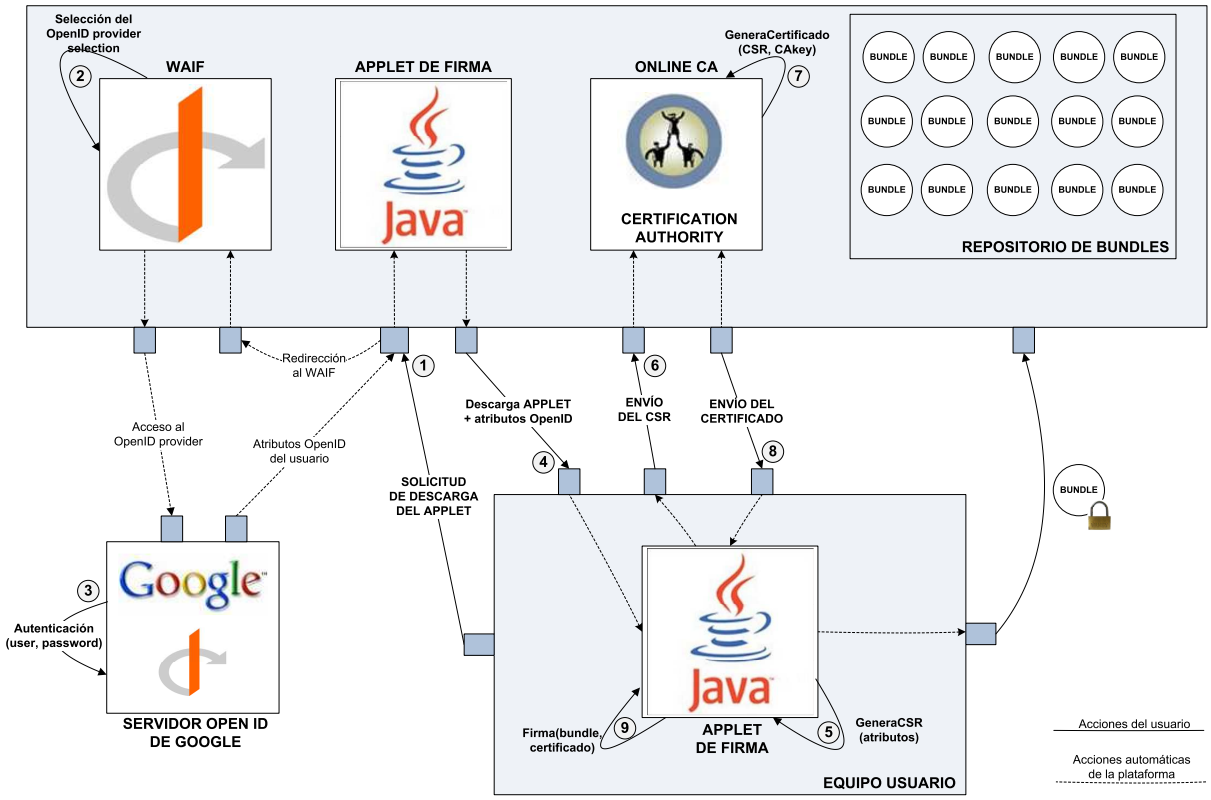

Figura 3. Arquitectura de la plataforma

guridad de este applet y como el usuario deposita su confianza autorizando su ejecución es un problema resuelto que por lo tanto queda fuera del ámbito de este trabajo.

IV-B. Proceso de verificación de un bundle firmado

El proceso de verificación se realiza de forma simétrica al proceso de firma consta de los siguientes pasos:

1. El primer paso será validar la identidad del firmante, para lo que habrá que verificar la cadena de confianza del certificado, que sólo estará formada por el certificado de clave pública de la CA online del repositorio, verificando también que dicha CA se encuentra en el almacén de certificados de confianza (truststore). Además, se deberán mostrar los atributos del certificado temporal, para que el cliente decida si confía en el proveedor de identidad OpenID que proporcionó los atributos con que se generó el certificado temporal.
2. El segundo paso de la verificación consiste en comprobar si los recursos dentro del bundle siguen el orden descrito en las especificaciones: archivo de manifiesto, archivo de hash (Signature File), archivo de firma (Block Signature File) y resto de archivos del bundle
3. El tercer y último paso será verificar la coherencia de los archivos de metadatos [8]: el archivo de firma debe contener una firma válida del archivo de hash, el archivo de hash debe contener un valor hash válido para el archivo de manifiesto y el archivo de manifiesto debe contener el nombre y valor hash de todos y cada uno de los recursos del bundle

En nuestra implementación hemos decidido que la verificación de la firma se haga en el mismo repositorio, de manera que se incorpora el resultado de dicha verificación en la vista que se muestra al usuario de forma que pueda decidir que en bundles deposita su confianza de forma que no supongan una sobrecarga para el dispositivo que va a ejecutar los bundles. Obviamente, esto implica que deba haber una relación de confianza con el servlet del repositorio que se encarga de esta función de verificación de firma si bien en nuestra esquema siempre suponemos que el repositorio es confiable. En cualquier caso, el usuario podrá descargar la firma junto con el bundle y verificarla localmente para un mayor nivel de seguridad.

Por tanto podemos describir el proceso de la autenticidad de los bundles desde el punto de vista de la siguiente manera:

- El cliente se conecta al servicio que muestra el listado de bundles disponibles en el repositorio
- Por cada bundle, el repositorio deberá mostrar:
  - Los atributos propios del bundle, como pueden ser el nombre, el tamaño o la fecha de creación
  - La identidad OpenID del desarrollador/firmante del bundle
  - El proveedor de identidad OpenID (ya que el cliente deberá decidir si es de confianza o no)
  - La CA que certifica dichas identidades,

- Algún tipo de mensaje de error en caso de que se produzca algún tipo de error durante el proceso de verificación.

- Viendo esta información, será misión del cliente decidir si se descarga el bundle y lo ejecuta o no.[3]

V. CONCLUSIONES Y TRABAJO FUTURO

En la actualidad es una máxima que la distribución de software debe de tener en cuenta durante todo su ciclo la seguridad. Entre otros aspectos se deben proporcionar mecanismos para evitar ataques de sustitución de software (o componentes) con el objetivo crítico de impedir la propagación de malware (y, por supuesto, todas las consecuencias nefastas que éste lleva asociado).

La plataforma OSGi tiene en la distribución de componentes software, así como en la actualización, instalación y desinstalación de manera dinámica, autónoma y en tiempo real, su mejor exponente. Nos sirve pues este escenario para desarrollar una plataforma de distribución de bundles (componentes software en OSGi) segura.

Sin embargo, el diseño de la plataforma es generalizable a cualquier escenario de distribución de ficheros (aplicaciones, componentes software, imágenes, etc.) donde el la identidad del propietario sea un factor crítico. La ventaja de nuestro esquema sobre los sistemas tradicionales de firma de código es su flexibilidad y facilidad de uso ya que solo requiere que el desarrollador se autentique frente al repositorio usando un proveedor OpenID favorito. Si bien no se elimina la necesidad de confiar en el repositorio, tal y como ocurre con la distribución de paquetes en sistemas tipo Debian, se añade un grado adicional de seguridad al certificarse la identidad del propietario mediante la firma del bundle con el certificado temporal creado a partir dos atributos proporcionados por el proveedor de identidad.

En la actualidad estamos trabajando en una migración de esta plataforma hacia otros escenarios, y considerando las implicaciones, sobre los terminales en particular y la plataforma diseñada en general, de aumentar los niveles de seguridad de este diseño (por ejemplo mediante el uso de CAs externas para la generación de certificados).

Otro aspecto que estamos considerando es la definición de extensiones para los certificados X.509v3 generados de forma que la cookie de autenticación OpenID o parte de ella se pueda incluir en el certificado temporal de firma.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado a través de los siguientes proyectos: OSAmI (TSI-020400-2009-92) y SPRINT (TIN2009-09237).

REFERENCIAS

[1] Panda Security, Informe Trimestral PandaLabs (Enero-Marzo 2010)
[2] Symbian Developer Certificates (http://wiki.forum.nokia.com/index.php/Developer_certificate)
[3] Secure APT en Debian (http://wiki.debian.org/SecureApt)
[4] OSGi Alliance (http://www.osgi.org)
[5] Eclipse (http://www.eclipse.org)
[6] Pierre Parrend, Protecting code archives with digital signatures, Proyecto OWASP, Enero 2008
[7] OpenID, OpenID Authentication 2.0 – Final, Specification, December, 2007.
[8] How a provider can do self-integrity checking, How to implement a provider for the Java Cryptography Extensions, Java Tutorials, SDK 1.5.
[9] Raffi Krikorian, Programatically Signing JAR Files, O'Reilly on Java, Abril 2001,
[10] JAR Signing and Verification Tool, Java Tutorials (http://java.sun.com/j2se/1.3/docs/tooldocs/win32/jarsigner.html)
[11] Key and Certificate Management Tool, Java Tutorials (http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html)

[3] El usuario podrá verificar de forma local la firma del bundle para aumentar el nivel de seguridad.