PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 09-06

Autorización de Acceso en MIDP 3.0

Gustavo Mazeikis **Carlos Luna**

2009

Autorización de Acceso en MIDP 3.0
Mazeikis, Gustavo; Luna, Carlos
ISSN 0797-6410
Reporte Técnico RT 09-06
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, abril de 2009

Autorización de Acceso en MIDP 3.0

Gustavo Mazeikis y Carlos Luna

Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay
mazeikis@adinet.com.uy, cluna@fing.edu.uy

Resumen. En la Plataforma Java Micro Edition, el Perfil para Dispositivos de Información Móviles (MIDP) provee el ambiente de ejecución estándar para teléfonos móviles y asistentes de datos personales. La tercera versión del perfil, de reciente publicación, introduce una nueva dimensión en el modelo de seguridad de MIDP: la seguridad a nivel de aplicación. Para la segunda versión de MIDP, Zanella, Betarte y Luna proponen una especificación formal del modelo de seguridad en el Cálculo de Construcciones Inductivas, usando el asistente de pruebas Coq. Este artículo presenta una extensión conservativa de la especificación referida que incorpora la autorización de acceso de aplicaciones a recursos compartidos, introducida por la tercera versión de MIDP. El trabajo también propone un refinamiento de la especificación y un algoritmo funcional para el evento de autorización de acceso, cuyo comportamiento es formalmente verificado. Finalmente, se pone de manifiesto una debilidad del nuevo modelo de seguridad. Mediante un caso concreto se ilustra como una aplicación accede a un recurso compartido falsificando una credencial exigida. Para subsanar esta debilidad, se proponen tres medidas alternativas.

Palabras Clave: Dispositivos Móviles Interactivos, MIDP, Especificación y Verificación Formal, Teoría de Tipos, Coq.

1. Introducción

En los últimos años el uso de dispositivos portátiles, tales como teléfonos celulares y asistentes digitales personales, se ha popularizado a escala mundial. Este tipo de dispositivos tienen fines y características que son en esencia diferentes de los que tienen, por ejemplo, las computadoras portátiles o de escritorio.

En la actualidad, la gran mayoría de las plataformas tecnológicas para los dispositivos portátiles incorporan la tecnología Java para sistemas embebidos – *embedded systems* – denominada *Java Micro Edition* (JME) [1]. La característica fundamental de esta edición, al igual que en toda tecnología Java, es el uso de una máquina virtual, especialmente diseñada y adaptada para aprovechar al máximo los escasos recursos con los que cuentan los dispositivos portátiles. La tecnología JME proporciona mecanismos integrales para garantizar propiedades de seguridad de los dispositivos; en particular, para dispositivos móviles define un modelo de seguridad

que restringe el acceso de las aplicaciones a funciones consideradas potencialmente peligrosas.

JME plantea una arquitectura basada en dos capas por encima de la máquina virtual: la capa configuración y la capa perfil. La configuración *Connected Limited Device Configuration* (CLDC) es un conjunto mínimo de bibliotecas de clases para ser utilizadas en dispositivos con procesadores poco potentes, memoria de trabajo limitada y capacidad para establecer comunicaciones de bajo ancho de banda. Esta configuración se complementa con el perfil *Mobile Information Device Profile* (MIDP) para obtener un entorno de ejecución adaptado a dispositivos portátiles como teléfonos celulares y asistentes digitales personales. El perfil define, entre otras cosas, un modelo de seguridad para aplicaciones, el ciclo de vida de las mismas, los mecanismos de comunicación de red y una biblioteca para el acceso a funciones propias de la clase de dispositivos para la que fue concebido.

En la primera versión de MIDP (MIDP 1.0) [2] se plantea, y en la segunda versión de MIDP (MIDP 2.0) [3] se refina, un modelo de seguridad a nivel de plataforma basado en conjuntos de permisos, para acceder a las funciones del dispositivo. En la tercera versión (MIDP 3.0) [4] se introduce una nueva dimensión al modelo: la seguridad a nivel de aplicación, basada en autorizaciones de acceso, para que una aplicación pueda acceder a los recursos que comparte otra aplicación.

Para MIDP 2.0 Zanella, Betarte y Luna proponen en [5, 6] una especificación formal del modelo de seguridad en el Cálculo de Construcciones Inductivas [7], usando el asistente de pruebas Coq [8]. Esta especificación es una base formal para la verificación del modelo de seguridad y la comprensión de su complejidad.

En este artículo se propone una extensión conservativa de la especificación formal mencionada y un algoritmo certificado para evaluar la autorización de acceso a los recursos compartidos de una aplicación en MIDP 3.0. La extensión de la formalización está también implementada usando el asistente de pruebas Coq. La versión completa del código fuente y la documentación están disponibles en [9].

Otro aporte de este trabajo son observaciones al nuevo modelo de seguridad, las cuales surgen del estudio de la especificación informal. Una debilidad introducida en esta nueva versión se pone de manifiesto mediante un ejemplo de acceso, falsificando una credencial exigida. Se proponen medidas alternativas para subsanar la debilidad referida.

El artículo está organizado como sigue. En la sección 2 se describe el modelo de seguridad de MIDP a través de sus distintas versiones. En la sección 3 se resume la especificación formal del modelo de seguridad de MIDP 3.0. La sección 4 presenta el algoritmo certificado para autorización de acceso. La sección 5 plantea observaciones a la seguridad a nivel de aplicación de MIDP 3.0 y, finalmente, la sección 6 exhibe las conclusiones y los trabajos futuros.

Una versión preliminar de este trabajo es [10] y una versión extendida es el reporte técnico [11].

2. El Modelo de Seguridad de MIDP

El modelo de seguridad de MIDP ha evolucionado a través de sus distintas versiones. En las primeras dos versiones el principal objetivo fue la protección de funciones sensibles del dispositivo, mientras que en la tercera versión se busca proteger los recursos de una aplicación que pueden compartirse con otras aplicaciones. Estos dos niveles de seguridad se describen a continuación.

2.1 Seguridad a Nivel de Plataforma

En MIDP 1.0 toda aplicación que no haya sido preinstalada en el dispositivo (por ejemplo, por el fabricante), se ejecuta en un ambiente controlado denominado *sandbox* que prohibe el acceso a aquellas funciones del dispositivo que se consideran potencialmente peligrosas.

En MIDP 2.0 se refina el modelo *sandbox* en uno menos restrictivo, basado en el concepto de *dominio de protección*. En los dispositivos compatibles con MIDP 2.0 cada aplicación instalada está ligada a un único dominio de protección. La política de seguridad es una matriz de control de acceso, donde los sujetos son las aplicaciones asociadas a cada dominio de protección y los objetos son las funciones del dispositivo que deben protegerse.

El dominio de protección otorga permisos de dos formas distintas a las aplicaciones que resultan ligadas al mismo. La primera forma es otorgando el permiso de manera incondicional. La segunda forma implica la participación del usuario mediante una modalidad de intervención: *blanket*, el permiso otorgado es válido mientras la aplicación está instalada; *session*, el permiso otorgado es válido mientras la aplicación está ejecutándose; y *oneshot*, el permiso es concedido sólo para ese uso. Por lo tanto, el uso de ciertos recursos sensibles involucra una consulta al usuario y en función de su respuesta se determina la disponibilidad del recurso.

El conjunto de permisos efectivamente otorgados a una aplicación se determina en función de su dominio de protección, de los permisos solicitados por la aplicación y de los permisos otorgados por el usuario. Una aplicación posee un determinado permiso cuando lo ha solicitado, y además su dominio de protección otorga el permiso incondicionalmente, o su dominio especifica un modo de interacción con el usuario para el permiso, y el usuario ha dado una autorización que continúa siendo válida.

El modelo establece dos clases de aplicaciones. Una aplicación no confiable es aquella para la que no se ha podido determinar el origen y la integridad. Toda aplicación no confiable, así como una desarrollada sobre MIDP 1.0, se asocia a un dominio de protección que provee un entorno de ejecución restringido. Una aplicación confiable es aquella para la que se ha podido determinar su origen e integridad. Cuando se verifica el origen de la aplicación, se determina también el dominio de protección al que se asocia.

2.2 Seguridad a Nivel de Aplicación

En MIDP 3.0 se permite que aplicaciones de diferentes dominios de seguridad compartan datos en formato *Record Management System* (RMS) y se comuniquen en tiempo de ejecución mediante eventos de aplicación, o mediante el protocolo *Inter-Midlet Communication* (IMC). La necesidad de autorizar el acceso a los recursos compartidos por una aplicación define una nueva dimensión en el modelo de seguridad de MIDP. La seguridad a nivel de aplicación, basada en el concepto de autorización de acceso, es complementaria de la seguridad a nivel de plataforma, basada en el concepto de permiso.

Los mecanismos para validar la identidad de las aplicaciones involucradas se basan en la Infraestructura de Clave Pública X.509 [3, 12]. Una aplicación se considera firmada si declara la firma del archivo de aplicación (realizada con la clave privada del vendedor) y el certificado de clave pública del vendedor. Los certificados raíz instalados en el dispositivo permiten verificar los certificados y las firmas declarados.

Para que una aplicación comparta recursos debe declarar autorizaciones de acceso, que se diferencian por las credenciales exigidas a las aplicaciones solicitantes. La declaración de autorización puede tomar algunas de las cuatro formas posibles: autorización para las aplicaciones del mismo dominio de seguridad; autorización para las aplicaciones firmadas con un certificado específico; autorización para las aplicaciones de un vendedor determinado, firmadas con un certificado específico; y autorización para las aplicaciones sin firmar, de un vendedor determinado.

Cuando otra aplicación pretende acceder a recursos compartidos, el dispositivo aplica un conjunto de reglas para determinar si está autorizada. Las reglas evalúan la declaración de la aplicación que comparte y las credenciales de la aplicación que solicita, como ser: el nombre de su dominio de protección y, el nombre, la firma y los certificados de clave pública de su vendedor. Si las credenciales de la aplicación solicitante se ajustan a las autorizaciones declaradas entonces se concede autorización de acceso a los recursos compartidos.

3. Especificación Formal del Modelo de Seguridad de MIDP 3.0

Esta sección describe una extensión de la formalización del modelo de seguridad de MIDP 2.0 [5, 6] con los conceptos relacionados a la seguridad a nivel de aplicación introducidos en MIDP 3.0. Se presenta la notación utilizada y los principales elementos de la especificación formal: la política de seguridad del dispositivo, las aplicaciones instaladas, el conjunto de estados válidos del dispositivo y los principales eventos vinculados con la seguridad. La conducta de los eventos se especifica mediante pre y pos condiciones. En la extensión se conservan las propiedades del modelo de seguridad demostradas en la formalización original. También se plantean y demuestran nuevas propiedades relacionadas con la autorización de acceso.

3.1 Notación

En la formalización se usa la notación estándar para la igualdad y los operadores lógicos ($\wedge, \vee, \neg, \rightarrow, \forall, \exists$). La definición de un tipo registro R con un único constructor mkR y funciones de proyección $campo_i$: $R \rightarrow A_i$, tiene la forma:

$$R =_{def} \{ campo_1: A_1, \ldots, campo_n: A_n \} \quad (1)$$

La aplicación de las proyecciones sobre un registro r se abrevia con la notación punto ($campo_i\ r = r.\ campo_i$). Por cada $campo_i$ de un registro r se define una relación binaria $\equiv campo_i$ entre objetos del tipo como sigue,

$$r_1 \equiv campo_i\ r_2 =_{def} \forall j, j \neq i \rightarrow r_1.campo_j = r_2.campo_j \quad (2)$$

Se asume como predefinido el tipo inductivo paramétrico *option T*, con los constructores *None: option T* y *Some: T→ option T*.

3.2 Política de Seguridad y Aplicaciones

Se denota al conjunto de los dominios de protección del dispositivo con *Domain* y al conjunto de los permisos para cada función protegida del dispositivo con *Permission*. La política de seguridad define, para cada dominio de protección, los permisos que se otorgan a las aplicaciones ligadas al dominio, así como los permisos que pueden ser otorgados por el usuario en una de tres modalidades. *Mode* denota el conjunto enumerado de estas modalidades {*oneshot, session, blanket*}. La política de seguridad es una constante de tipo *Policy*.

$$\begin{aligned} &Policy =_{def} \{ \\ &\quad allow: Domain \rightarrow Permission \rightarrow Prop, \\ &\quad user: Domain \rightarrow Permission \rightarrow Mode \rightarrow Prop \ \} \end{aligned} \quad (3)$$

Las aplicaciones MIDP, comúnmente llamadas MIDlets, suelen agruparse para su distribución en paquetes denominados suites. Se denota el conjunto de los identificadores válidos para MIDlet suites con *SuiteID*. Un tipo registro *Suite* representa una suite instalada, con campos para su identificador, el dominio de protección asociado y su descriptor.

$$Suite =_{def} \{ id : SuiteID, domain : Domain, descriptor : Descriptor \} \quad (4)$$

Una MIDlet suite expone, mediante un archivo descriptor, información relevante para el dispositivo. Entre otros atributos declara: el nombre de su vendedor, la firma y el certificado de la clave pública en caso de ser firmada, los permisos sobre los recursos sensibles que utiliza, y las autorizaciones de acceso sobre los recursos de aplicación que comparte.

El tipo registro *Descriptor* representa el archivo homónimo. *Vendor* denota los nombres de vendedores de suites, *Signature* representa la firma del archivo de la aplicación, y *Certificate* el conjunto de los certificados de clave pública. Los predicados *required* y *optional* denotan respectivamente la declaración de los permisos necesarios y opcionales. El predicado *domainAuthorization* denota la declaración de autorización de acceso por dominio de protección. La autorización de

acceso para un conjunto de suites firmadas con un certificado particular se establece con el predicado *signerAuthorization*. El predicado *vendorUnsignedAuthorization* hace lo propio para el conjunto de suites sin firmar de un vendedor, mientras que *vendorSignedAuthorization* denota la autorización de acceso al conjunto de suites, firmadas con un certificado específico de un vendedor.

$$
\begin{aligned}
&Descriptor =_{def} \{ \\
&\quad vendor: Vendor, \\
&\quad jarSignature: option\ Signature, \\
&\quad signerCertificate: option\ Certificate, \\
&\quad required, optional: Permission \rightarrow Prop, \\
&\quad domainAuthorization: Domain \rightarrow Prop, \\
&\quad signerAuthorization: Certificate \rightarrow Prop, \\
&\quad vendorUnsignedAuthorization: Vendor \rightarrow Prop, \\
&\quad vendorSignedAuthorization: Vendor \rightarrow Certificate \rightarrow Prop \}
\end{aligned}
\tag{5}
$$

3.3 Estado del Dispositivo y Eventos

El estado del dispositivo se modela con el conjunto de suites instaladas, la información de la suite activa, en caso de que haya una, los permisos otorgados o revocados en modo permanente, y las autorizaciones de acceso concedidas o denegadas a las suites instaladas.

$$
\begin{aligned}
&State =_{def} \{ \\
&\quad suite : SuiteID \rightarrow Prop, \\
&\quad session: option\ SessionInfo, \\
&\quad granted, revoked: SuiteID \rightarrow Permission \rightarrow Prop, \\
&\quad authorized, unauthorized: SuiteID \rightarrow SuiteID \rightarrow Prop \}
\end{aligned}
\tag{6}
$$

La información de la suite activa, denotada con *SessionInfo*, comprende el identificador de la suite, junto con los permisos otorgados o revocados a ella en la sesión.

$$
SessionInfo =_{def} \{ id : SuiteID, granted, revoked : Permission \rightarrow Prop \} \tag{7}
$$

La validez del estado del dispositivo se establece con el predicado *Valid*, como una conjunción de condiciones que deben verificarse. La compatibilidad entre los permisos ofrecidos por el dominio de protección y los permisos solicitados por la suite, se establece con el predicado *SuiteCompatible*. La unicidad de los identificadores de suites instaladas está dada con el predicado *UniqueSuiteID*. Por otra parte, la compatibilidad entre los permisos otorgados, en una sesión o en forma permanente, y los permisos solicitados se establece con *ValidSessionGranted* y *ValidGranted,* respectivamente. Finalmente, el predicado *ValidAuthorization* establece que una suite no puede estar autorizada y desautorizada por otra suite al mismo tiempo.

$$Valid = SuiteCompatible \wedge UniqueSuiteID \wedge CurrentInstalled \wedge ValidSessionGranted \wedge ValidGranted \wedge ValidAuthorization \quad (8)$$

Los eventos relacionados con la seguridad se modelan como constructores del tipo *Event* y se detallan en la tabla 1.

Tabla 1. Eventos relacionados con la seguridad

Nombre	***Descripción***	***Tipo***
install	*instala una suite*	$SuiteID \rightarrow Descriptor \rightarrow Domain \rightarrow Event$
remove	*remueve una suite*	$SuiteID \rightarrow Event$
start	*inicia una sesión*	$SuiteID \rightarrow Event$
terminate	*fin de una sesión*	*Event*
request	*solicita permiso*	$Permission \rightarrow option\ UserAnswer \rightarrow Event$
authorization	*solicita autorización*	$SuiteID \rightarrow Event$

El comportamiento de los eventos se especifica mediante pre y pos condiciones. Las pre condiciones están definidas en términos del estado del dispositivo, mientras que las pos condiciones se definen en términos del estado anterior, el estado posterior y una respuesta opcional del dispositivo. La respuesta del dispositivo se representa con el tipo *Response*, donde el valor *allowed* denota la aceptación y *denied* el rechazo.

El evento *authorization* modela la solicitud de autorización de acceso por parte de una suite a los recursos compartidos de la suite activa. Tiene como precondición que en el estado *s* exista una suite activa y que el identificador de la suite solicitante *idReq* corresponda al identificador de una suite instalada en el dispositivo.

$$Pre\ s\ (authorization\ idReq) =_{def} \forall ses: SessionInfo, \quad (9)$$
$$s.session = Some\ ses \rightarrow \exists msReq: Suite,$$
$$s.suite\ msReq \wedge msReq.id = idReq$$

La poscondición establece que se autoriza el acceso a la suite solicitante, si no estaba previamente desautorizada y sus credenciales coinciden con alguna declaración de autorización de la suite activa. En este caso, el estado resulta modificado ya que la suite solicitante pasa a integrar el conjunto de suites autorizadas por la suite activa.

$$Pos\ s\ s'\ (Some\ allowed)\ (authorization\ idReq) =_{def} \forall ses: SessionInfo, \quad (10)$$
$$s.session = Some\ ses \rightarrow \neg\, s.unauthorized\ ses.id\ idReq \wedge$$
$$AccessAuthorization\ s\ ses.id\ idReq \wedge s \equiv_{authorized} s' \wedge$$
$$s'.authorized\ ses.id\ idReq$$

El predicado *AccessAuthorization* representa la evaluación de las reglas de autorización de acceso en un estado del dispositivo. Se concede la autorización cuando se verifica al menos una de las siguientes condiciones.

La primera condición verifica que el dominio de protección de la suite solicitante *msReq* satisface el predicado *domainAuthorization* de la suite *msGrn* que comparte el recurso.

IsDomainAuthorized := (**11**)
msGrn.descriptor.domainAuthorization (msReq.domain)

La segunda condición establece que el vendedor y el certificado de la suite solicitante *msReq* satisfacen el predicado *vendorSignedAuthorization* de la suite *msGrn* que comparte el recurso.

IsVendorSignedAuthorized := (**12**)
msGrn.descriptor.vendorSignedAuthorization
(msReq.vendor msReq.signerCertificate)

La tercera condición verifica que el certificado de la suite solicitante *msReq* satisface el predicado *signerAuthorization* de la suite *msGrn* que comparte el recurso.

IsSignerAuthorized := (**13**)
msGrn.descriptor.signerAuthorization (msReq.signerCertificate)

Finalmente, la cuarta condición verifica que el nombre de vendedor de la suite solicitante *msReq* satisface el predicado *vendorUnsignedAuthorization* de la suite *msGrn* que comparte el recurso.

IsVendorUnsignedAuthorized := (**14**)
msGrn.descriptor.vendorUnsignedAuthorization (msReq.vendor)

El caso donde se niega el acceso a la suite solicitante, se omite por razones de espacio. Por la misma causa se omite el comportamiento de los restantes eventos. El lector interesado puede consultar [9, 11] por más información.

Un resultado interesante, demostrado en la formalización, es la invariancia de la validez del estado del dispositivo con respecto a la ejecución de cualquier evento. También se demuestra que cuando una suite es desautorizada, cualquier solicitud de acceso posterior de la misma suite es rechazada. Estas propiedades y sus demostraciones están disponibles en [9].

4. Prototipo Certificado para Autorización de Acceso

La representación del modelo de seguridad, descripta en la sección anterior, posee un alto nivel de abstracción. Esta característica no condiciona posibles implementaciones y resulta adecuada para razonar sobre las propiedades del modelo. Sin embargo, para extraer un prototipo del algoritmo para autorización de acceso y certificar que el mismo satisface la conducta especificada, se debe buscar una representación concreta que sea representable en un lenguaje funcional.

4.1 Representación concreta

Sin pérdida de generalidad ni inconsistencia se asume que todos los predicados utilizados para describir el estado del dispositivo y los eventos son decidibles y tienen un dominio finito. El criterio adoptado para la transformación segura de un predicado *P* decidible y definido sobre un conjunto finito *S* es el siguiente. Cuando el predicado *P* caracteriza una colección de elementos, se representa como una lista exhaustiva *l* de elementos de tipo *S* que satisfacen el predicado. Por otra parte, cuando el predicado *P* caracteriza una propiedad sobre un conjunto de elementos, se representa como una función *F* de *S* en un tipo isomorfo a *bool*, tal que: $\forall x \in S,\ P\ x \Leftrightarrow F\ x = true$.

A modo de ejemplo, la representación concreta del estado del dispositivo se presenta a continuación. El conjunto de suites instaladas se representa como una lista, mientras que las autorizaciones y desautorizaciones de acceso se representan con las correspondientes funciones sobre booleanos.

$$
\begin{aligned}
&CState =_{def} \{ \\
&\quad suites : list\ CSuite, \\
&\quad session : option\ CSessionInfo, \\
&\quad granted : SuiteID \rightarrow Permission \rightarrow bool, \\
&\quad revoked : SuiteID \rightarrow Permission \rightarrow bool, \\
&\quad authorized : SuiteID \rightarrow SuiteID \rightarrow bool, \\
&\quad unauthorized : SuiteID \rightarrow SuiteID \rightarrow bool\ \}
\end{aligned}
\tag{15}
$$

4.2 Algoritmo de Autorización de Acceso

En la representación concreta se propone una implementación del algoritmo para autorización de acceso. El algoritmo *AlgAccessAuthorization* recibe como parámetros el identificador de la suite solicitante y el estado concreto actual, y retorna una pareja formada por un estado, eventualmente modificado, y la respuesta a la solicitud de autorización. A partir del estado actual el algoritmo obtiene la suite activa, que es la que dispone del recurso compartido; y a partir del identificador obtiene la suite solicitante. Luego, el algoritmo procede a comparar las credenciales de la suite solicitante con las declaraciones de autorización de la suite activa.

Este análisis se descompone en varios casos, según las condiciones evaluadas. Si hay correspondencia entre la declaración de la suite activa y las credenciales de la suite solicitante, la respuesta es solicitud concedida, sino es solicitud denegada. El estado del dispositivo es modificado en la medida que la suite solicitante pasa a formar parte de las suites autorizadas o desautorizadas por la suite activa [11].

```
AlgAccessAuthorization (sidReq: SuiteID)(cst: CState): CState * option Response :=

    let msGrn:= getMIDletSuiteInSession cst in          (* suite activa en la sesión *)
    let msReq:= getMIDletSuite sidReq cst.suites in     (* suite solicitante     *)

    match (isAuthorized cst msReq) with
    | true          => (cst, Some allowed)      (* autorización existente *)
    | false         =>

    match (isUnauthorized cst msReq) with
    | true          => (cst, Some denied)       (* desautorización existente *)
    | false         =>

    match (isDomainAuthorized cst msReq) with
    | true          => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            addMIDletSuiteAuthorized cst sidReq, cst.unauthorized )
                            , Some allowed)     (* autorización por Dominio *)
    | false         =>

    match (isSigned msReq) with
    | true          =>                          (* suite solicitante firmada *)
          match (isVendorSignedAuthorized cst msReq) with
          | true    => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            addMIDletSuiteAuthorized cst sidReq, cst.unauthorized )
                            , Some allowed)     (* autorización por Vend. y Certificado *)
          | false   =>

          match (isSignerAuthorized cst msReq) with
          | true    => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            addMIDletSuiteAuthorized cst sidReq, cst.unauthorized )
                            , Some allowed)     (* autorización por Certificado *)

          | false   => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            cst.authorized, addMIDletSuiteUnauthorized cst sidReq )
                            , Some denied)      (* desautorización de suite firmada *)
          end end
    | false         =>                          (* suite solicitante no firmada *)
          match (isVendorUnsignedAuthorized cst msReq) with
          | true    => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            addMIDletSuiteAuthorized cst sidReq, cst.unauthorized )
                            , Some allowed)     (* autorización por Vendedor *)

          | false   => (mkCState (cst.suites, cst.session, cst.granted, cst.revoked,
                            cst.authorized, addMIDletSuiteUnauthorized cst sidReq )
                            , Some denied)      (* desautorización de suite no firmada *)
          end
    end end end end
```

4.3 Certificación del Algoritmo

Una propiedad importante que podemos verificar en la formalización concreta es la certificación del algoritmo de autorización de acceso. El algoritmo está certificado si a partir de un estado inicial válido, en el cual se cumple la precondición del evento *authorization*, se obtiene por la ejecución del algoritmo, un estado resultante que satisface la poscondición de dicho evento. Esta propiedad se formaliza a continuación.

$$\forall (cst\ cst' : CState)\ (msReq : CSuite)\ (r : option\ Response), \quad (16)$$
$$CPre\_authorization\ cst\ msReq.id \rightarrow$$
$$AlgAccessAuthorization\ msReq.id\ cst = (cst', r) \rightarrow$$
$$CPos\_authorization\ cst\ cst'\ msReq.id\ r$$

La demostración se sigue por la estructura del algoritmo y está basada en el análisis de casos de las condiciones evaluadas. El lector interesado puede encontrar la demostración completa en [9].

5. Debilidad del Sistema de Autorización de MIDP 3.0

En esta sección se revisan las diferencias entre los niveles de seguridad de MIDP 3.0. De la comparación surge una debilidad en la seguridad a nivel de aplicación. Esta situación se ilustra mediante un ejemplo concreto y se aportan sugerencias para su corrección.

5.1 Comparación entre Niveles

En la especificación informal de MIDP pueden apreciarse diferencias en la definición de los niveles de seguridad. Las diferencias están en la granularidad de la protección sobre los recursos, en la modalidad en que los permisos son otorgados y en los mecanismos para establecer la confianza en las aplicaciones.

A nivel de plataforma, el dominio de protección define permisos para cada uno de los recursos sensibles del dispositivo. Por el contrario, a nivel de aplicación la declaración de autorización de acceso no diferencia entre los múltiples recursos que una aplicación puede compartir. Esto implica que una vez que se otorga autorización de acceso, la misma permite acceder a todos los recursos compartidos por la aplicación.

En segundo lugar, aunque un permiso a nivel de plataforma puede tener distintas modalidades de otorgamiento (por única vez, por sesión o permanente), la autorización de acceso a nivel de aplicación sólo es otorgada en forma permanente.

Finalmente, el grado de confianza a nivel de plataforma está basado exclusivamente en la Infraestructura de Clave Pública X.509 [12], a través de firmas y certificados digitales. Sin embargo, a nivel de aplicación no todas las declaraciones de confianza se basan en dicho mecanismo. Existe una declaración de autorización de acceso que exige como credencial a la aplicación solicitante el nombre del vendedor de la aplicación. Este atributo de la aplicación no está protegido por firma o

certificado alguno y su autenticidad no es evaluada en ningún momento. Esto deja una brecha que puede ser explotada por una aplicación maliciosa para acceder a un recurso compartido, como se muestra en la siguiente sub sección.

5.2 Un Ejemplo de Acceso Indeseado

Este ejemplo ilustra un posible escenario donde una aplicación no confiable accede a un recurso compartido por otra aplicación, presentando una credencial falsa.

El *Record Management System* (RMS) es un recurso sensible del dispositivo que permite crear y acceder a información persistida en un almacén de registros. Independientemente del dominio de protección al que resulten ligadas, todas las aplicaciones tienen permisos sobre los métodos del RMS.

En MIDP 3.0 la aplicación propietaria de un almacén de registros puede compartirlo con otras aplicaciones y restringir el acceso al mismo usando el mecanismo de autorización. En la API del RMS, el método *openRecordStore* permite crear y acceder a un almacén de registros. Dependiendo de su propósito, cada aplicación emplea una signatura distinta del mismo método.

La aplicación propietaria del almacén de registros emplea la siguiente signatura para crearlo y compartirlo en forma restringida. Usando el valor AUTHMODE_ APPLEVEL en el tercer parámetro del método, se activa el mecanismo de control de acceso basado en autorizaciones.

public static javax.microedition.rms.RecordStore openRecordStore ((**17**)
String recordStoreName, boolean createIfNecessary, ***int authmode****,*
boolean writable)

En el ejemplo, se crea un almacén de cuentas bancarias llamado *BankAccountsDb* con seguridad a nivel de aplicación.

rsPrv = RecordStore.openRecordStore ((**18**)
"BankAccountsDb", true, ***AUTHMODE_APPLEVEL****, false);*

Para restringir el acceso, la aplicación debe incluir autorizaciones de acceso en su descriptor. En el ejemplo, la *declaración de autorización de acceso por vendedor* autoriza a todas las aplicaciones realizadas por el mismo vendedor *TrustyVendor* que provee la aplicación.

MIDlet-Name: TrustyMIDlet (**19**)
MIDlet-Version: 1.0
MIDlet-Vendor: TrustyVendor
MIDlet-Access-Authorization-1: vendor;TrustyVendor

Por otra parte, una aplicación que pretende acceder al recurso compartido debe emplear la signatura indicada abajo del método *openRecordStore*. Requiere además conocer el nombre del recurso compartido y algunos datos de la aplicación proveedora: su vendedor y su nombre público.

public static javax.microedition.rms.RecordStore openRecordStore ((**20**)
String recordStoreName, ***String vendorName****, String suiteName)*

Supongamos que un tercero no autorizado pretende obtener la información de las cuentas bancarias. Para ello desarrolla una aplicación sin firmas ni certificados digitales, empleando como nombre de vendedor el mismo nombre del propietario del almacén de registros. Un fragmento de su descriptor se detalla a continuación.

```
MIDlet-Name: MaliciousMIDlet                                    (21)
MIDlet-Version: 1.0
MIDlet-Vendor: TrustyVendor
```

Al no estar firmada, la aplicación maliciosa es instalada y ligada al *Dominio de Protección para Terceros No Identificados*. Esto protege a la plataforma de que la aplicación acceda a los recursos sensibles del dispositivo. Sin embargo no evita que tenga acceso al almacén de cuentas bancarias compartidas.

El siguiente código Java le permite a la aplicación no confiable abrir el almacén de registros y obtener la información.

```
// abre el almacén compartido                                   (22)
rsCns =RecordStore.openRecordStore(
"BankAccountsDb","TrustyVendor","TrustyMIDlet");

// procede a leer los registros del almacén
int nextID = rs.getNextRecordID();
byte[] data = null;
for( int id = 0; id < nextID; ++id ){
    try {
     int size = rs.getRecordSize( id );    // obtiene el tamaño del registro
    if( data == null || data.length < size )
              data = new byte[ size ];
               rs.getRecord( id, data, 0 );          // obtiene el registro
              processRecord( rs, id, data, size );  // procesa el registro
    } catch( InvalidRecordIDException e ){
    } catch( RecordStoreException e ){
              handleError( rs, id, e ); }
}
```

5.3 Observaciones y Alternativas

El ejemplo anterior ilustra un escenario donde una aplicación explota una debilidad del modelo de seguridad para acceder a un almacén de registros compartidos. La debilidad se explica por dos factores: el tipo de credencial exigida y el aseguramiento de la compatibilidad hacia atrás.

En primer lugar, el tipo de declaración de autorización empleada exige como única credencial el nombre del vendedor de la aplicación. Sobre este atributo no hay un mecanismo que permita determinar su autenticidad, por lo cual es fácilmente modificable por una aplicación maliciosa. Sin embargo, los demás tipos de declaración exigen como credenciales el dominio de protección o la firma y el certificado de clave pública. A diferencia del primer caso, la autenticidad de estos atributos está asegurada.

En segundo lugar, MIDP 2.0 considera a la API del RMS como un recurso sensible a nivel de plataforma, aunque no tan crítico, ya que otorga permisos de uso a una aplicación no confiable. Para asegurar la compatibilidad hacia atrás, este permiso también es otorgado en MIDP 3.0 a una aplicación no confiable.

Como alternativa a esta situación se plantean dos sugerencias. La primera es eliminar la declaración de autorización de acceso por nombre de vendedor, que exige una credencial fácilmente alterable. La segunda es revisar los permisos otorgados a las aplicaciones no confiables, como es el caso de los permisos sobre el RMS. Lo que hasta MIDP 2.0 se consideraba un recurso local a nivel de plataforma, pasa a ser un recurso de aplicación que puede ser accedido por otras aplicaciones, al ser compartido.

Más allá de este ejemplo concreto, y considerando la comparación entre los niveles del modelo de seguridad, se plantea una tercera sugerencia. La misma tiene que ver con eliminar la diferencia de granularidad entre los permisos a nivel de plataforma y las autorizaciones a nivel de aplicación. Con una declaración de autorización de acceso donde se incluya el identificador del recurso compartido se establece un mejor control. Actualmente la declaración de autorización no discrimina sobre el recurso al que se brinda acceso.

6. Conclusiones y Trabajos Futuros

El perfil MIDP define el ambiente de ejecución estándar para aplicaciones Java que se ejecutan en teléfonos celulares. Su viabilidad y éxito como plataforma móvil para aplicaciones depende en gran medida de la madurez de su modelo de seguridad.

El primer abordaje formal del modelo de seguridad de MIDP 2.0 demuestra ser una poderosa herramienta para razonar sobre las propiedades que debe cumplir cualquier implementación del estándar. Los cambios introducidos con la reciente publicación de MIDP 3.0 motivaron el desarrollo de una extensión de la especificación formal citada.

La extensión propuesta incorpora la autorización de acceso a los recursos compartidos entre aplicaciones, conservando las propiedades verificadas en la formalización anterior. A partir de un refinamiento seguro de la formalización se desarrolló un algoritmo para evaluar la autorización de acceso a los recursos de una aplicación y se certificó con respecto a la especificación del evento de seguridad involucrado. El algoritmo puede utilizarse en una técnica complementaria a la verificación formal, como es el testing de caja negra, donde oficie de oráculo frente a casos de pruebas derivados de la especificación. Este constituye un aporte trascendente para la comunidad vinculada al desarrollo de tecnologías seguras para dispositivos móviles de última generación. La formalización completa consta de una decena de módulos (archivos) Coq con unas 6000 líneas, que incluyen 130 definiciones, y 148 lemas y teoremas.

Del análisis de la especificación informal de MIDP 3.0 surgen notorias diferencias entre los niveles del modelo de seguridad. Tanto en la granularidad de la definición de la protección, como en la modalidad de otorgamiento de los permisos y, especialmente, en los mecanismos para establecer confianza en las aplicaciones, estas diferencias contribuyen a una debilidad del modelo. Se establece, con un caso

concreto, como una aplicación obtiene acceso a un recurso compartido falsificando una credencial. En este trabajo se plantearon tres alternativas para contrarrestar la debilidad analizada.

Como trabajo futuro se plantea la obtención de un prototipo ejecutable de la especificación completa del modelo de seguridad de MIDP 3.0. Haciendo uso de las ventajas que un asistente de pruebas basado en teoría de tipos, como Coq, brinda tanto para verificar sistemas como para derivar implementaciones correctas por construcción, este prototipo certificado puede ser construido.

Referencias

1. Java Platform Micro Edition, http://java.sun.com/javame/index.jsp, último acceso: Abril de 2009.
2. JSR 37 Expert Group: Mobile Information Device Profile for Java Micro Edition. Version 1.0. Sun Microsystems Inc. (2000).
3. JSR 118 Expert Group. Mobile Information Device Profile for Java Micro Edition. Version 2.0. Sun Microsystems Inc. and Motorola Inc. (2002).
4. JSR 271 Expert Group. Mobile Information Device Profile for Java Micro Edition.Version 3.0, Public Review Specification, Motorola Inc. (2007).
5. S. Zanella. Especificación formal del modelo de seguridad de MIDP 2.0 en el Cálculo de Construcciones Inductivas. Master's thesis, UNR, Argentina (2006).
6. S. Zanella, G. Betarte, y C. Luna. A Formal Specification of the MIDP 2.0 Security Model. T. Dimitrakos et al. (Eds.): FAST 2006, LNCS 4691, pp. 220–234, 2007. Springer-Verlag Berlin Heidelberg (2007).
7. Y. Bertot y P. Castéran. Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer-Verlag, (2004).
8. The Coq Development Team, The Coq Proof Assistant Reference Manual VersionV8.1 (2006).
9. G. Mazeikis. MIDP Security Model, Código Coq de la Especificación Formal, 2008, http://midpsecurity.wetpaint.com/, último acceso: Abril de 2009.
10. G. Mazeikis, C. Luna, G. Betarte. Formalización y Análisis del Modelo de Seguridad de MIDP 3.0. Seguridad a Nivel de Aplicación, First Chilean Workshop on Formal Methods, ChWFM 2008, págs 34–43, ISBN 978-956-319-507-1, Punta Arenas, Chile, Noviembre de 2008.
11. G. Mazeikis. Formalización y Análisis del Modelo de Seguridad de MIDP 3.0. Reporte Técnico (2009). Disponible en http://midpsecurity.wetpaint.com/, último acceso: Abril de 2009.
12. Internet X.509 Public Key Infrastructure, http://www.ietf.org/rfc/rfc2459, último acceso: Abril de 2009.