Universidad Zaragoza

# Proyecto Fin de Carrera

Ingeniería Informática

# Análisis y Diseño de Mecanismos de Seguridad de Agentes Móviles para la Plataforma SPRINGS


Autor

Jorge Sainz Vela

Director

Sergio Ilarri Artigas

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza


Septiembre de 2013

# Análisis y Diseño de Mecanismos de Seguridad de Agentes Móviles para la Plataforma SPRINGS


RESUMEN

Los agentes móviles son programas que pueden viajar de un ordenador a otro y continuar su ejecución en el ordenador destino. Se ha demostrado su utilidad en diversos contextos, especialmente en entornos inalámbricos y distribuidos. Así, ofrecen diversas ventajas, como la posibilidad de desplazar el procesamiento a los datos para realizar un procesamiento local y minimizar las comunicaciones. Sin embargo, dichas ventajas implican ciertos problemas de seguridad, los cuales han impedido el desarrollo y el uso generalizado de la plataformas de agentes móviles. Dichos problemas de seguridad, se suelen categorizar de la siguiente forma:

- Agente contra plataforma. Medidas para evitar que un agente dañe la plataforma sobre la que esta corriendo. Por ejemplo, ataques de denegación de servicio, acceso a recursos no permitidos, etc.
- Agente contra agente. Ataques entre agentes, mediante denegaciones de servicio en el paso de mensajes, robos de datos, etc.
- Plataforma contra agente. Modificación o copia de código y/o datos del agente, modificación de los mensajes que se envían los agentes, enrutamiento incorrecto del agente, etc.

En el proyecto hemos estudiado el estado del arte y hemos elegido e implementado las medidas de seguridad más importantes para proveer a la plataforma de agentes móviles SPRINGS (Scalable PlatfoRm for MovINg Software) de un marco seguro de ejecución que permite su utilización en despliegues sobre redes abiertas y "no confiables". Se ofrece un diseño que permite al administrador de la plataforma hacer uso o no de las funcionalidades de seguridad que considere oportunas según sus necesidades.

Las extensiones de seguridad integradas en SPRINGS permitirán al grupo SID (Research Group of Distributed Information Systems) de la Universidad de Zaragoza continuar el desarrollo de la plataforma con soporte para proteger a los sistemas de agentes móviles de posibles ataques. El grupo SID tiene intención de continuar el estudio y extender la arquitectura diseñando e implementando otras medidas de seguridad que puedan resultar de interés. Posteriormente, se cree que se estará en condiciones de poder preparar un artículo de investigación conjunto que incluya parte del trabajo desarrollado en este proyecto.

## Agradecimientos

A mi familia, por ofrecerme la posibilidad de estudiar esta carrera que tanto me apasiona sin reproches por haber tardado tanto tiempo en llegar hasta aquí.

Gracias también a mis compañeros, ex-compañeros y amigos, por apoyarme y no dejar que me olvidara de que tenía que cerrar etapas para poder abrir otras.

A Sergio, por su paciencia aguantando mis idas y venidas a lo largo de los años siempre dispuesto a ofrecerme su guía y ayuda en el desarrollo del proyecto.

Y, en especial, a María, por estar siempre a mi lado.

## Acrónimos

A continuación se definen varios acrónimos utilizados a lo largo del presente documento:

**ADK** Tryllian Agent Development Kit.

**AES** Advanced Encryption Standard.

**AMS** Agent Management System.

**API** Application Programming Interface.

**ARE** Tryllian Agent Runtime Environment.

**ASN.1** Abstract Syntax Notation One.

**CA** Autoridad de Certificación.

**CORBA** Common Object Request Broker Architecture.

**DER** Distinguished Encoding Rules.

**DNA** Tryllian Definition of New Agent.

**DoS** Denial of Service.

**FIPA** Foundation for Intelligent Physical Agents.

**GPL** General Public License.

**IMTP** Internal Message Transport Protocol.

**JAAS** Java Authentication and Authorization Service.

**JADE** Java Agent DEvelopment Framework.

**JADE-S** JADE Security.

**JAR** Java ARchive.

**JAVA SE** Java Platform Standard Edition.

**JCE** Java Cryptography Extension.

**JDK** Java Development Kit.

**JRMP** Java Remote Method Protocol.

**JSSE** Java Secure Socket Extension.

**JVM** Java Virtual Machine.

**KQML** Knowledge Query and Manipulation Language.

**LGPL** Lesser General Public License.

**MAC** Message Authentication Code.

**MD5** Message-Digest algorithm 5.

**MIDP** Mobile Information Device Profile.

**NTP** Network Time Protocol.

**OTP** One Time Password.

**PAM** Pluggable Authetication Module.

**PKCS** Public-Key Cryptography Standards.

**RMI** Java Remote Method Invocation.

**RNS** Region Name Server.

**SeMoA** Secure Mobile Agents.

**SID** Research Group of Distributed Information Systems.

**SPRINGS** Scalable PlatfoRm for MovINg Software.

**SSL** Secure Sockets Layer.

**TLS** Transport Layer Security.

**UML** Unified Modeling Language.

# Índice

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

En el presente capítulo vamos a realizar una presentación del Proyecto Fin de Carrera desarrollado. En primer lugar ofreceremos una breve descripción de conceptos sobre agentes móviles y su seguridad para familiarizar al lector con el entorno tecnológico del proyecto. Pasaremos a destacar los objetivos establecidos para el proyecto y la planificación estimada para el mismo. Por último, describiremos el contenido del resto de capítulos del documento.

## 1.1 Entorno tecnológico sobre agentes móviles

Un agente móvil es un tipo de agente software que tiene las siguientes capacidades: autonomía y pro-actividad, sociabilidad, adaptación, y, cómo no, movilidad. Los agentes móviles tienen la capacidad de moverse por diferentes nodos de una red, una o más veces, son autónomos para ejecutar tareas correctamente, independientemente del nodo de red desde el que se lanzaron, transportando su estado de un entorno a otro, con sus datos inalterados [1].

La característica diferenciadora de los agentes móviles, frente a paradigmas de programación de código móvil [2] como la evaluación remota o el código bajo demanda, es que los agentes móviles pueden "elegir" cuándo migrar entre nodos de la red en cualquier momento de su ejecución.

Se denomina plataforma de agentes móviles al conjunto de elementos estructurales que ofrecen los servicios necesarios a los agentes móviles para que estos puedan realizar su función correctamente. Una plataforma de agentes móviles suele tener una arquitectura como la mostrada en la Figura 1.1. Normalmente cada plataforma usa una nomenclatura propia pero, en esencia, los elementos mostrados pueden aparecer en la mayoría.

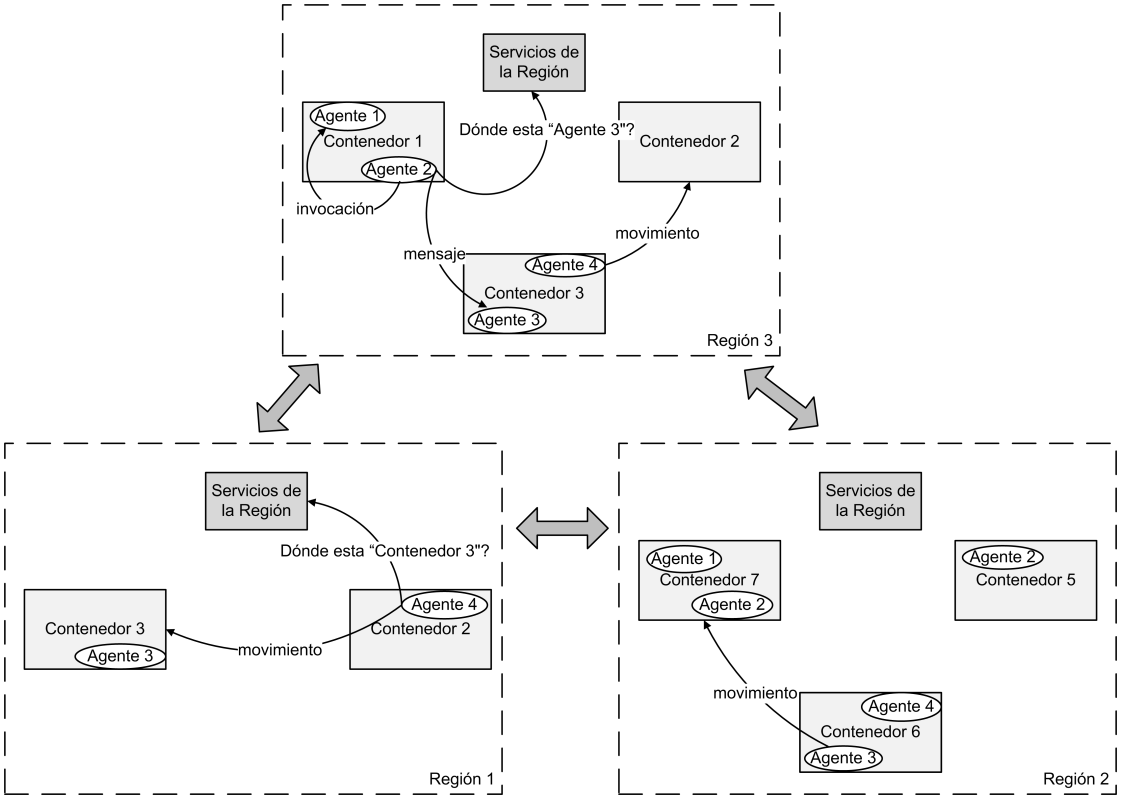



Figura 1.1: Ejemplo de plataforma de agentes móviles

Los elementos de la plataforma de la Figura 1.1 son:

- Región. Suele denominarse así al conjunto de contenedores o servidores que forman parte de una misma organización. Suelen tener varios servicios de control como pueden ser servidores de nombres, de *páginas amarillas*, etc.
- Contenedores. Son los servidores que ofrecen sus servicios computacionales a los agentes. En esencia, es donde se ejecutan los agentes. Los contenedores suelen ser denominados servidores, contextos, agencias, hábitats, dependiendo de las plataformas.
- Agentes. Son los programas que se ejecutan en los contenedores pudiendo moverse entre contenedores y seguir su ejecución en ellos. También suele ser habitual la posibilidad de comunicación entre agentes.

Las principales ventajas de los agentes móviles son [3]:

- reducen la carga de la red,

- ayudan evitar problemas relacionados con la latencia de la red,
- encapsulan protocolos,
- se ejecutan asíncronamente y autónomamente,
- se adaptan dinámicamente,
- permiten un mantenimiento flexible,
- son robustos y tolerantes a fallos.

Por todas estas razones y sus posibles aplicaciones, los agentes móviles han despertado una gran expectación en el mundo de la computación distribuida [4, 5, 6].

Sin embargo, debido al carácter distribuido del paradigma de agentes móviles, uno de los puntos más importantes a tener en cuenta es el referente a la seguridad [7, 8, 9, 10]. Mas específicamente, se suelen tener en cuenta los siguientes aspectos [11]:

- protección del agente contra la plataforma,
- protección del agente frente a otros agentes,
- protección de la plataforma frente al agente.

## 1.2 Objetivos del proyecto

A lo largo del presente Proyecto de Fin de Carrera, se estudiará el estado del arte de la seguridad en plataformas de agentes móviles y se elegirán e implementarán las medidas de seguridad más importantes para la plataforma SPRINGS de la Universidad de Zaragoza [12]. Además, se estudiará el impacto en el rendimiento de la plataforma al introducir dichas medidas de seguridad.

La arquitectura de seguridad de SPRINGS deberá ofrecer un diseño modular, permitiendo al administrador de la plataforma hacer uso o no de las funcionalidades de seguridad que considere oportunas según sus necesidades.

El proyecto comprende las siguientes tareas:

- Estudio del estado del arte sobre los diferentes problemas de seguridad de las plataformas de agentes móviles y de las diferentes propuestas para su solución.
- Definición y diseño de la arquitectura modular de seguridad elegida para SPRINGS.

- Implementación de la arquitectura modular de seguridad diseñada para SPRINGS.

- Pruebas funcionales para comprobar la correcta protección de la plataforma ante posibles ataques.

- Pruebas de carga de la plataforma para comprobar cómo afecta la nueva arquitectura de seguridad al rendimiento de SPRINGS.

Por tanto, el resultado final será una ampliación de la plataforma SPRINGS con protecciones frente a distintos ataques de seguridad, con distintos niveles de protección configurables, así como un estudio del rendimiento de la plataforma en función del nivel de protección escogido. Esta ampliación permitirá la utilización libre de la plataforma por un mayor número de personas que puedan estar interesadas.

## 1.3 Planificación estimada del proyecto

Inicialmente, se ha estimado la duración del proyecto en alrededor de cinco meses. Se debe tener en cuenta la necesidad por parte del autor de investigar en un área nueva para él y el sobre-esfuerzo necesario para realizar una implementación en el lenguaje de desarrollo Java.

El diagrama de Gantt estimado para el proyecto es el mostrado en la Figura 1.2:

| | Nombre de tarea | Duration | Start | Finish | Pred |
|---|---|---|---|---|---|
| 1 | **PFC Seguridad en SPRINGS** | **227 days** | **Mon 09/11/09** | **Tue 21/09/10** | |
| 2 | **Aprobacion del Proyecto** | **20 days** | **Mon 09/11/09** | **Fri 04/12/09** | |
| 3 | Presentacion Propuesta | 0 days | Mon 09/11/09 | Mon 09/11/09 | |
| 4 | Aprobacion Propuesta | 20 days | Mon 09/11/09 | Fri 04/12/09 | 3 |
| 5 | | | | | |
| 6 | **Estudio del Arte** | **20 days** | **Mon 17/05/10** | **Fri 11/06/10** | 2 |
| 7 | **Diseño** | **20 days** | **Mon 14/06/10** | **Fri 09/07/10** | 6 |
| 8 | Definicion de las caracteristicas a implementar | 3 days | Mon 14/06/10 | Wed 16/06/10 | |
| 9 | Diseño de la solucion | 12 days | Thu 17/06/10 | Fri 02/07/10 | 8 |
| 10 | Documentacion | 5 days | Mon 05/07/10 | Fri 09/07/10 | 9 |
| 11 | **Desarrollo** | **30 days** | **Mon 12/07/10** | **Fri 20/08/10** | 7 |
| 12 | **Pruebas** | **15 days** | **Mon 23/08/10** | **Fri 10/09/10** | 11 |
| 13 | Funcionales | 5 days | Mon 23/08/10 | Fri 27/08/10 | |
| 14 | Carga | 5 days | Mon 30/08/10 | Fri 03/09/10 | 13 |
| 15 | Documentacion | 5 days | Mon 06/09/10 | Fri 10/09/10 | 14 |
| 16 | **Documentacion Defensa** | **5 days** | **Mon 13/09/10** | **Fri 17/09/10** | 12 |
| 17 | **Deposito** | 1 day | Mon 20/09/10 | Mon 20/09/10 | 16 |
| 18 | **Defensa** | 1 day | Tue 21/09/10 | Tue 21/09/10 | 17 |
| 19 | | | | | |

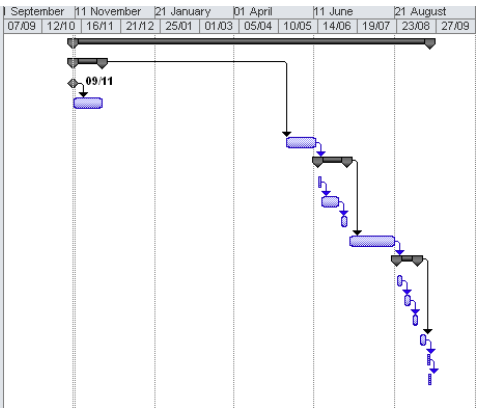



Figura 1.2: Diagrama de Gantt estimado

En dicho diagrama se puede observar la duración estimada de las principales tareas de las que se compone el proyecto.

## 1.4 Contenido de la memoria

En los siguientes capítulos vamos a ir exponiendo cada una de las tareas de las que se compone el presente proyecto:

- En el Capítulo 2 procederemos a definir los diferentes términos relacionados con la seguridad en plataformas de agentes móviles pasando, a continuación, a describir los mecanismos de seguridad que ofrecen las plataformas de agentes móviles más importantes.
- En el Capítulo 3 expondremos la arquitectura de seguridad diseñada para SPRINGS, así como detalles de su implementación en la plataforma.
- El Capítulo 4 detalla las pruebas, tanto funcionales como de carga, realizadas a la plataforma, analizando posibles escenarios de ataque así como el impacto en el rendimiento de la plataforma de las medidas implementadas.
- El Capítulo 5 contiene las conclusiones obtenidas del proyecto, así como guías para trabajos futuros.

También se incluyen varios apéndices a la presente memoria que contienen información detallada sobre el trabajo realizado:

- En el Apéndice A exponemos el entorno tecnológico del proyecto. Más específicamente, se incluye una descripción de conceptos generales de seguridad informática, el uso de Java como lenguaje y entorno de ejecución para muchas plataformas de agentes móviles y sus implicaciones en la seguridad de éstas y la descripción de los entornos sobre los que se ha realizado el desarrollo y las pruebas del proyecto.
- En el Apéndice B se muestra el detalle de todas las pruebas funcionales realizadas a la plataforma, incluyendo opciones de configuración y *logs*.
- El Apéndice C incluye el manual del usuario de la plataforma, tanto a nivel de administración de la misma como a nivel de programación.

# Capítulo 2

# Seguridad en plataformas de agentes móviles

Aunque los *frameworks* de seguridad para plataformas de agentes móviles son relativamente modernos, la mayoría de los principios en los que están basados no lo son. La características fundamentales de una arquitectura de seguridad primitiva son tan relevantes para una plataforma de agentes móviles como para aplicaciones distribuidas tradicionales. En concreto, para el caso de plataformas de agentes móviles podemos indicar los siguientes requisitos:

- Un agente debe poder ser autenticado cuando visite una plataforma de agentes.
- La plataforma debe poder ser autenticada por un agente que la visite.
- Los agentes deben ser capaces de cifrar partes confidenciales de sus datos y de su código.
- Los datos de los agentes deben estar protegidos de una manipulación no autorizada, realizada por otros agentes o plataformas.
- El código de un agente no debe ser alterado por la plataforma.
- Un agente debe ser capaz de firmar un contrato para asegurar su responsabilidad, por ejemplo, realizando una firma digital mediante una clave privada.
- Un agente no debe interferir en la ejecución de otro agente o en la operación de la plataforma.
- Una plataforma no debe evitar la ejecución autorizada y/o la migración de un agente no permitido.

Un resumen de las diferentes capas de seguridad, sus requisitos y su aplicación en plataformas de agentes móviles se muestra en la Tabla 2.1:

| Capa | Requisito | Aplicación |
|---|---|---|
| Externa | Autenticación | Los agentes y las plataformas se deben autenticar. |
| | Confidencialidad | Los agentes deben poder cifrar sus datos y su código. |
| Interna | Integridad | La plataforma debe evitar la modificación de los componentes de la misma. |
| Integridad y Privacidad | Integridad | Los datos de los agentes deben estar protegidos ante manipulaciones no autorizadas. |
| | Responsabilidad | Un agente debe poder disponer de una clave privada para la firma digital de objetos. |
| | Disponibilidad | Un agente no debe interferir en la ejecución de otros agentes ni en la operación de la plataforma y la misma no debe evitar la ejecución de los agentes. |

Tabla 2.1: Capas de seguridad en plataformas de agentes móviles

## 2.1 Clasificación de ataques

En las plataformas de agentes móviles existen, principalmente, tres clases de ataques. Esta clasificación se realiza de acuerdo a quién es el elemento malicioso (el agente y/o la plataforma) y, a su vez, quién es el elemento atacado (de la misma forma, el agente y/o la plataforma) [11, 13, 14].

### 2.1.1 Agente malicioso atacando a la plataforma

A continuación se describen algunos tipos de ataques que puede realizar un agente contra la plataforma:

- Ataques de denegación de servicio: el agente consume excesivos recursos (tales como la memoria, la CPU o el ancho de banda de red) del contenedor, haciendo que ésta no pueda ofrecer sus servicios a otros agentes.
- Acceso no autorizado a datos del contenedor: El agente intenta acceder a datos confidenciales, ficheros del sistema, almacenes de claves o ficheros de políticas, o intenta manipular los mecanismos de gestión del contenedor o propiedades del entorno de ejecución.

- *Masquerading*: el agente se hace pasar por otro agente con más permisos para conseguir acceso a datos o servicios sensibles.

- Ataques complejos mediante la colaboración con otros agentes.

### 2.1.2 Agente malicioso atacando a otros agentes

A continuación se citan algunos ataques de agentes contra otros agentes.

- Cambiar el estado o tareas de otro agente.

- Leer o manipular los datos de otro agente.

- Enmascarar su identidad para engañar a otros agentes y conseguir acceso a información sensible de otros agentes o usar servicios suplantando la identidad de otros agentes.

- Retardar o evitar que otro agente pueda realizar las tareas que tuviera que ejecutar, por ejemplo tardando intencionadamente más de lo previsto en devolver la respuesta a una invocación.

- Ataques de denegación de servicio contra otros agentes, enviándoles mensajes basura.

### 2.1.3 Plataforma atacando agentes

A continuación se listan una serie de ataques que pueden realizar los contenedores u otras partes de la plataforma contra los agentes.

- Acceder o modificar a los datos de los agentes: leyendo datos confidenciales (claves privadas) o manipulando los datos recogidos.

- Acceder o modificar el código de un agente y/o su *workflow*: leyendo el listado de migraciones que un agente tiene previsto seguir o sus algoritmos; cambiando el comportamiento del agente permanentemente o temporalmente para el beneficio del contenedor malicioso o para dañar a otros contenedores. Si se pudiera modificar el código de un agente, se podrían "reprogramar" sus acciones, con lo que se podría cambiar su comportamiento.

- Retrasar la ejecución de un agente innecesariamente o rechazar su ejecución incluso estando autorizado sin informar al contenedor origen.

- Ataque de “copiar y pegar”: Un contenedor “copia” porciones de los datos de un agente y los “pega” en otro agente. Si dichos datos estuvieran cifrados, el agente podría migrar a otro contenedor donde el descifrado fuera posible y volver al contenedor original con los datos en claro.

Una plataforma idealmente debería disponer de medidas de seguridad para prevenir todos los tipos de ataques anteriormente citados.

## 2.2 Estudio de medidas de seguridad en plataformas de agentes móviles existentes

Con el objetivo de estudiar los aspectos de seguridad en plataformas de agentes móviles, se han examinado con más detalle plataformas de agentes que se están utilizando en la actualidad.

Históricamente han existido numerosas plataformas de agentes móviles, tanto de código abierto como comerciales. Sin embargo, debido a intereses comerciales de la empresas que las mantenían parece que muchas de las plataformas de agentes móviles comerciales ya no están disponibles para su utilización.

Existen también muchas plataformas que fueron desarrolladas con motivos académicos en proyectos de investigación y no están mantenidas, o que fueron implementadas para configuraciones específicas y no se pueden utilizar universalmente. Otras plataformas, simplemente, no ofrecen servicios de migración.

Debido a lo anteriormente descrito, el estudio propuesto se ha centrado principalmente dos plataformas de agentes móviles desarrolladas bajo licencia GPL (General Public License). La primera es SeMoA (Secure Mobile Agents), que es una plataforma de agentes móviles de referencia en todos los aspectos de la seguridad de agentes móviles, incluyendo la protección de los agentes frente a nodos maliciosos. La segunda es JADE (Java Agent DEvelopment Framework), uno de los marcos de desarrollo de agentes móviles más utilizados y mejor mantenidos de la actualidad.

De todas formas, tanto por razones históricas, como por la importancia de las plataformas también han sido objeto de estudio las plataformas Aglets [15], Voyager [16] y Tryllian [17].

### 2.2.1 Características de seguridad de SeMoA

SeMoA es una plataforma de código abierto de agentes móviles enfocada a la seguridad. Fue desarrollada por el *Fraunhofer-Institut für Graphische Datenverarbeitung*,

y se ha utilizado en varios proyectos esponsorizados por el Ministerio Federal Alemán de Economía y Tecnología. La plataforma ofrece las funcionalidades básicas para la migración, comunicación y seguimiento/monitorización (*tracking*) de agentes. Su última versión data de agosto de 2007. Además, SeMoA ofrece una arquitectura de seguridad muy elaborada con una estructura modular que permite la inclusión de más mecanismos de seguridad para evitar ataques contra tanto agentes como servidores.

Los elementos que conforman la plataforma SeMoA son servidores (que ofrecen servicios), agentes y lo que se denomina el registro [18]. Los servidores se encargan principalmente de cargar e instalar agentes en la plataforma, de enviar la señales de parada a los agentes, de eliminar agentes, de mantener la separación y el anonimato entre los agentes y de ofrecer servicios a los agentes. El registro tiene como función publicar los nombres de los objetos, obtener los objetos por su nombre, buscar los objetos por su nombre y listar los nombres de los objetos.

SeMoA utiliza transparencia en la localización de agentes mediante un sistema propio denominado ATLAS. Puede utilizar diferentes protocolos de comunicación entre agentes, entre los que se encuentran FIPA (Foundation for Intelligent Physical Agents) [19] y KQML (Knowledge Query and Manipulation Language) [20].

A continuación se describe la arquitectura de seguridad de la plataforma SeMoA, como se detalla en [18].

#### Arquitectura de seguridad

SeMoA ha sido desarrollado sobre JAVA SE (Java Platform Standard Edition) y con la intención de proveer de una seguridad adecuada a sistemas de agentes móviles, tanto a servidores como a agentes. La arquitectura de seguridad de SeMoA se compara a una cebolla: los agentes tienen que pasar por varias capas de protección antes de que sean admitidos por el sistema de ejecución (ver Figura 2.1) y de que la primera clase de un agente sea cargada en la JVM (Java Virtual Machine) del servidor.

La primera (la más externa) capa de seguridad es un protocolo de seguridad en el transporte, como TLS (Transport Layer Security) o SSL (Secure Sockets Layer). En este momento, la implementación de SSL utilizada es la que viene con el *framework* JSSE (Java Secure Socket Extension) de Oracle. Esta capa ofrece una autenticación mutua entre servidores de agentes, cifrado transparente y protección de integridad. Las peticiones de conexión entre objetos autenticados pueden ser aceptadas o rechazadas según una política de seguridad configurable.

La segunda capa consiste en un *pipeline* de filtros de seguridad. Se soportan *pipelines* separados para agentes entrantes y salientes. Cada filtro inspecciona y procesa los agentes entrantes y salientes, y o los acepta o los rechaza. Este procedimiento de filtrado se denomina "inspección de contenidos", en analogía a conceptos conocidos de *firewalls*.

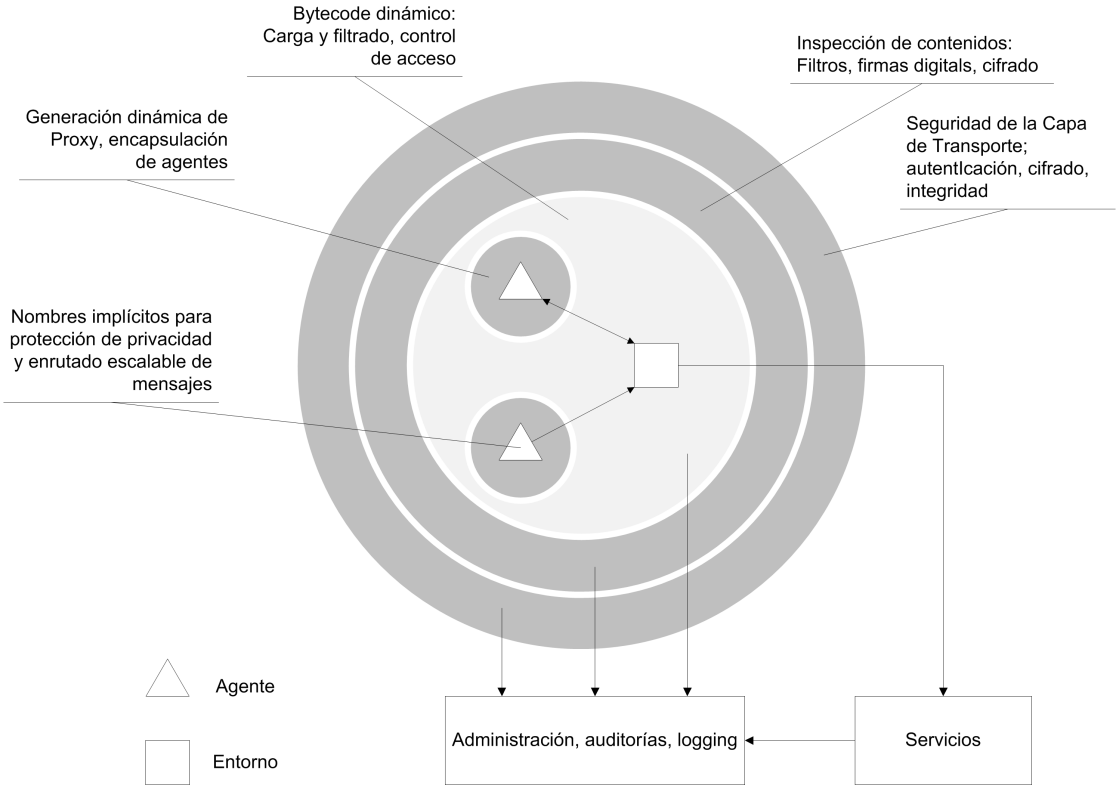



Figura 2.1: Capas de seguridad de SeMoA (adaptado de http://semoa.sourceforge.net)

SeMoA incluye dos pares de filtros complementarios que se encargan de las firmas digitales y del cifrado selectivo de agentes (verificación de firmas y descifrado de agentes, cifrado y firma de agentes). Un filtro adicional al final del *pipeline* de entrada asigna un conjunto configurable de permisos a los agentes entrantes, basados en la información obtenida y verificada por los filtros anteriores. Se pueden verificar permisos basándose en las identidades autenticadas del propietario del agente, del responsable del último cambio de estado, y de su emisor mas reciente. Los filtros se pueden activar y desactivar tanto dinámicamente como en el momento de inicialización del servidor SeMoA.

Después de pasar todos los filtros de seguridad, SeMoA crea una *sandbox* para el agente aceptado. A cada agente se le asigna un grupo de *threads* y un *class loader*. Utilizando el *framework* de serialización de Java SeMoA consigue que el *thread* se bloquee hasta que no queden mas *threads* dentro del grupo de *threads* del agente. Solo entonces, SeMoA se encarga de cualquier petición de migración de ese agente. Este mecanismo evita que los agentes puedan sobrecargar una red de servidores de agentes al intentar migrar y evitar terminar su ejecución al mismo tiempo.

Las clases de un agente son cargadas por su *class loader* dedicado. Este *class loader* permite la carga de clases empaquetadas con el agente, así como clases de

fuentes externas de código. Todas las clases cargadas son verificadas contra un conjunto seguro de funciones *hash*. Las huellas de las clases verificadas deben coincidir con las huellas firmadas por el propietario del agente. Por lo tanto, solo clases autorizadas por propietario del agente para ser usadas por su agente son cargadas dentro del *namespace* del agente.

Los agentes no pueden compartir clases, por lo que un agente no puede cargar una clase de otro agente que contenga un *Caballo de Troya* [21] en su *namespace*. Sin embargo, para permitir invocar métodos entre agentes, pueden compartir interfaces. Se entiende que los interfaces son iguales si sus firmas coinciden. En este caso, el *class loader* de un agente devuelve un interfaz cargado previamente en vez de cargar el interfaz del otro agente otra vez, por lo que los interfaces utilizados por los agentes son compatibles siempre que sea posible. Por supuesto, este mecanismo solo funciona si los interfaces de las clases referenciados por dos agentes son idénticas a nivel de bit. Otros esquemas mejorados, pueden comparar implementaciones del interfaz a nivel de API (Application Programming Interface). Sin embargo, esto añade una sobrecarga al proceso de carga de clases. Los objetos de interfaz son gestionados en un *Map* que es global a todos los class loaders de los agentes. Ataques de "contaminación" contra este *Map* requieren que se rompan a la vez todas las funciones de *hash*.

Antes de que una clase sea definida en la JVM, el *bytecode* de esa clase debe pasar unos niveles de seguridad similares a los de los agentes entrantes. Cada filtro de clases puede inspeccionar, rechazar e incluso modificar el *bytecode*.

Los agentes están separados del resto de agentes del sistema, no se publican referencias a instancias de agentes por defecto. El único método de compartir instancias entre agentes es publicarlos en un entorno global. Cada agente obtiene su propia vista de su entorno, que sigue la pista a los objetos registrados por el agente. Todos los objetos publicados se empaquetan en *proxies* que son creados dinámicamente. Si un agente termina o reclama un objeto publicado, entonces el entorno del agente realiza un petición de invalidar su enlace al objeto original al correspondiente *proxy*. Esto hace que el objeto original deje de estar disponible, incluso para otros agentes que busquen su referencia en el entorno global. Esto también hace que el objeto original esté disponible para el *Garbage Collector*.

### Estructura del agente

En SeMoA, los agentes móviles son transportados como archivos JAR (Java ARchive). La especificación de los ficheros JAR de Oracle extiende la de los archivos ZIP, incluyendo soporte para firmas digitales, mediante la inclusión al contenido del archivo ZIP de los ficheros de firmas apropiados. El formato de la firma es PKCS (Public-Key Cryptography Standards)#7 [22], un estándar de sintaxis de mensajes cifrados, que está construido a partir de estándares como ASN.1 (Abstract Syntax Notation One) [23],

X.501 [24] y X.509 [25]. A su vez, utilizando el estándar PKCS#7, SeMoA extiende el formato JAR con soporte para cifrado selectivo del contenido del JAR para múltiples receptores. El cifrado y descifrado se realiza de forma transparente a los agentes mediante los filtros descritos previamente. Para evitar que partes cifradas de un agente sean copiadas y usadas en conjunto con otros agentes (ataques de copiar y pegar), estos filtros pueden obtener, de forma no interactiva, las claves de descifrado necesarias.

Cada agente contiene dos firmas digitales. La entidad que firma la parte estática del agente (la parte que se mantiene inalterada a lo largo del ciclo de vida del agente) se entiende como el propietario de pleno derecho de dicho agente (la entidad por la que el agente está actuando en su nombre). Cada servidor emisor también firma el agente completo (tanto la parte estática como la que contiene datos variables), vinculando el nuevo estado del agente con su parte estática.

Los agentes pueden acceder a los datos almacenados en sus partes estáticas y dinámicas (se puede apreciar en la Figura 2.2). Cuando un agente migra, su estructura es procesada por los filtros de seguridad de salida, y se vuelve a comprimir en un fichero JAR para su transporte hacia el servidor de destino. El grafo de los objetos serializados de la instancia de un agente (es decir, todo el conjunto de datos dinámicos del agente) también se almacena en la estructura del agente. Las estructuras del agente pueden almacenarse tanto en almacenamiento persistente como en no persistente, dependiendo de la configuración del servidor. La estructura del agente también contiene las propiedades del agente (pares clave/valor), tales como el nombre del agente en formato legible, y las fuentes de código de donde cargar las clases. Las propiedades deben ser firmadas por el propietario del agente, para ser protegidas del *tampering*.

Además, SeMoA calcula nombres implícitos para los agentes, aplicando una función SHA1 a la firma del propietario. Esto hace que los nombres sean globalmente únicos, a la par que anónimos. Los nombres implícitos se usan en SeMoA para proveer al agente de trazabilidad a la vez que de una capacidad de enrutado escalable independiente de la ubicación de los mensajes entre agentes.

#### Seguridad en la ejecución

SeMoA utiliza el modelo de seguridad de Java 2 para controlar el acceso a objetos críticos del servidor. Los permisos otorgados a los agentes son definidos por el filtro de políticas del conjunto de filtros de entrada. Estos permisos son asignados a todas las clases cargadas por el agente (tanto a las clases propias del agente, como a las clases remotas cargadas por el agente). SeMoA permite a código autorizado revocar los permisos en tiempo de ejecución. Las consecuencias de la revocación son inmediatas y se aplican a todas las clases del agente. Esto permite limitar el impacto de agentes "fuera de control" en un servidor, ya que la plataforma podría actuar limitando los permisos de las clases de un agente.

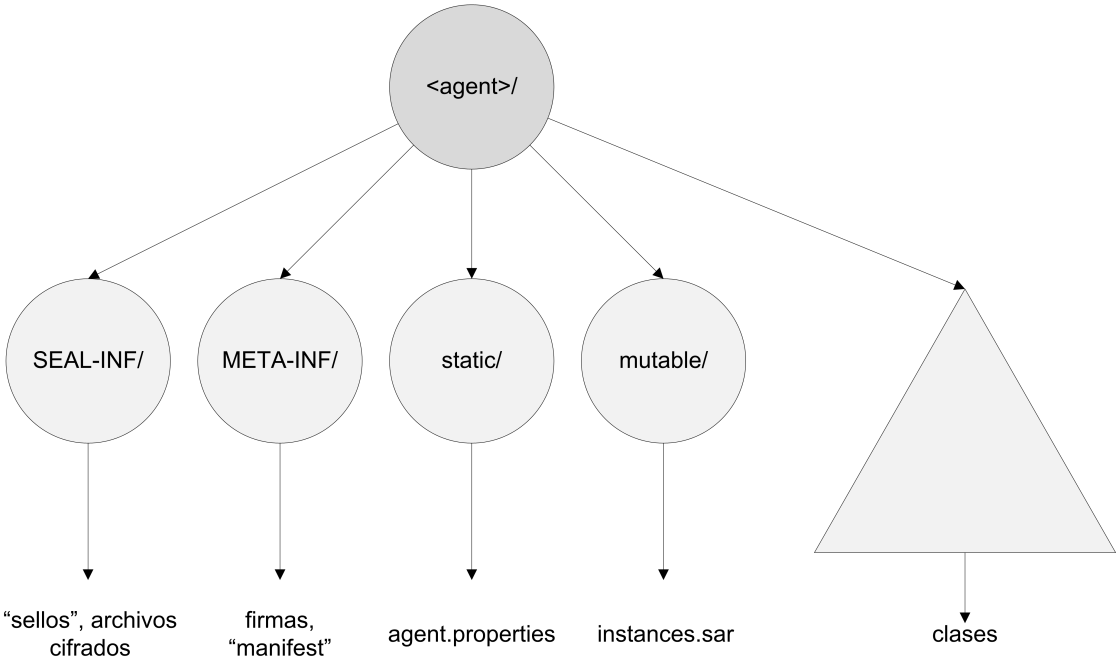



Figura 2.2: Estructura de un agente SeMoA (adaptado de http://semoa.sourceforge.net)

### Limitaciones

SeMoA está construido sobre una (JVM) estándar, donde no se han modificado los paquetes del núcleo de Java. Esto hace que ciertas medidas de seguridad, que se podían esperar en una plataforma de agentes móviles, no se puedan implementar. Una de las más importantes es la falta de un control de recursos apropiado en la JVM. Como consecuencia, SeMoA no es robusto ante una serie de ataques de denegación de servicio (DoS (Denial of Service)) como los que provocan una agotamiento de la memoria.

Otro problema es la terminación forzada de agentes o, más precisamente, la terminación de *threads*. Todos los métodos que permiten parar threads están marcados como "obsoletos" en Java 2 debido a que utilizarlos provoca estados inconsistentes en los objetos. Incluso si se llama a `stop()` en el *thread* de un agente, el agente puede capturar la excepción `ThreadDeath` (o cualquier otra `Throwable`) que se propague por la pila del *thread* y continuar la ejecución. SeMoA ignora este problema. El servidor SeMoA marca los agentes que deberían terminar y "confía" en que estos terminen su ejecución.

La última limitación, pero no menos importante es que hay una serie de clases en el núcleo de Java que se sincronizan (método *synchronize*) con clase de un objeto determinado. Como las clases locales son compartidas y su visibilidad es global a cualquier agente que adquiera un bloqueo en dicha clase, bloqueará cualquier *thread* que intente acceder a la clase.

Alguno de estos problemas se puede intentar evitar mediante la reescritura dinámica del código de ejecución de los agentes, restricciones en la visibilidad de las clases principales y la minimización de la compartición de clases. SeMoA no implementa estas técnicas, ya que los problemas subyacentes pueden ser mitigados utilizando filtros de fácil integración en su arquitectura.

Por lo tanto, una serie de limitaciones de Java pueden ser utilizadas por agentes maliciosos para lanzar varios ataques DoS. Sin embargo, para hacer eso, en primer lugar los agentes deben ejecutarse primero. Para ejecutarse (e incluso antes de que las clases del agente hayan sido cargadas), los servidores SeMoA verifican la identidad del propietario del agente basándose en firmas y certificados digitales, con lo que se establece una relación de confianza.

Desde el punto de vista de la administración, la plataforma SeMoA requiere un buen nivel de conocimientos de la misma, con lo que no es sencilla de configurar ni de administrar. A esto se le une la falta de documentación tanto a nivel de administración como a nivel de programación (se proveen ejemplos y una API documentada, pero no una guía de programación que facilite el uso de la plataforma a un usuario novato). Por lo tanto, no consideramos que la plataforma sea "sencilla", probablemente debido al nivel de funcionalidad que abarca y sus propuestas de seguridad, que intentan ser lo más "paranoicas" posibles [26].

### 2.2.2 Características de seguridad de JADE

JADE es una de las plataformas de agentes móviles más extendidas, utilizadas y mejor mantenidas del momento (la última versión es la 4.3.0 de marzo de 2013). A diferencia de otras plataformas de agentes móviles, no se ha considerado primordial la implantación de mecanismos de seguridad en la plataforma, primando otros aspectos, como la simplicidad en la implantación de un sistema de agentes móviles y la utilización de estándares abiertos (como FIPA). Es un *framework* basado en Java que, además, dispone de varias herramientas gráficas que facilitan tanto la gestión de la plataforma como el desarrollo y depuración de la misma.

JADE es software libre y es distribuido por Telecom Italia (poseedor de su copyright) en forma de código abierto bajo los términos LGPL (Lesser General Public License).

La plataforma JADE incluye tanto las librerías necesarias para desarrollar agentes como el entorno de ejecución que provee los servicios básicos que deben estar activos en un dispositivo antes de que se puedan ejecutar los agentes. Cada instancia del entorno de ejecución de JADE se denomina *contenedor* (debido a que "contiene" agentes). El conjunto de todos los contenedores se denomina *plataforma* y provee una capa homogénea que abstrae de la complejidad y la diversidad de la tecnología necesaria para ejecutar la plataforma (*hardware*, sistemas operativos, tipos de red, JVM) a los agentes (y, también,

a los desarrolladores de aplicaciones) [27].

Como mecanismo de comunicación entre agentes, JADE utiliza el estándar FIPA. Los *proxies* no existen en JADE, por lo que las búsquedas de agentes se realizan consultando a un elemento de la plataforma denominado AMS (Agent Management System), según se indica en el estándar FIPA.

### Capa de Seguridad JADE-S

La seguridad dentro de JADE se ha desarrollado como un módulo adicional opcional, no formando parte del núcleo de la plataforma. Este módulo se denomina JADE-S (JADE Security). Las características de seguridad que ofrece dicho módulo son: autenticación de usuarios, autorización de acciones de los agentes por parte de un sistema de permisos, y firma y cifrado de mensajes [28].

### Autenticación

La autenticación ofrece la garantía de que el usuario iniciando una plataforma JADE y, por lo tanto, los contenedores y los agentes que forman dicha plataforma, sea considerado legítimo dentro del ámbito de seguridad del sistema que integra el contenedor principal de la plataforma.

En general, una sistema de autenticación se compone de dos elementos principales: un *Callback Handler*, que permite al usuario proveer su nombre de usuario y su clave, y un *Login Module*, que comprueba si dicho nombre de usuario y clave son válidos.

El mecanismo de autenticación de JADE está basado en el API JAAS (Java Authentication and Authorization Service), que permite forzar el control de acceso diferenciado de usuarios del sistema. El mecanismo JAAS provee un conjunto de *Login Modules*, como son los módulos Unix, Windows NT y Kerberos. Los módulos Unix y Windows NT son dependientes del sistema operativo y están diseñados para utilizar la identidad de los usuarios extraída de la sesión actual del sistema operativo. La operación del módulo Kerberos es independiente del sistema operativo, pero requiere una configuración específica de la plataforma previa a su uso

Junto a todos estos mecanismos, se ofrece también un módulo especial denominado `Simple`. Este módulo permite una autenticación básica contra un fichero de claves en texto plano y se recomienda su uso simplemente para pruebas.

Actualmente también se proveen los siguientes mecanismos de *callback*:

- cmdline - el nombre de usuario y la clave se incluyen como parámetros de

configuración de JADE, tanto a nivel de parámetros de línea de comando (-owner user:pass) como a nivel de fichero de configuración (owner=user:pass).

- text - el contenedor inicial pide el nombre de usuario y la clave por consola.
- dialog - el contenedor inicial muestra un diálogo para introducir el usuario y la clave.

Si ocurre cualquier problema durante la autenticación, o si el usuario no puede ser correctamente autenticado, el sistema saldrá y generará el mensaje de error apropiado.

### Permisos

Gracias al mecanismo de autenticación previamente descrito, una plataforma basada en JADE-S se convierte en un sistema multi-usuario donde todos los componentes (contenedores y agentes) pertenecen a un usuario autenticado. Todas las acciones que realizan los agentes en la plataforma pueden ser autorizadas o denegadas, de acuerdo con un conjunto de reglas. De este modo es posible permitir el acceso a los servicios de la plataforma o a los recursos de las aplicaciones de una forma selectiva. Este conjunto de reglas suelen estar descritos en un fichero llamado `policy.txt`, que sigue la sintaxis estándar de Java/JAAS, pero usa un modelo de políticas extendido que permite una mayor flexibilidad en un escenario basado en agentes distribuidos.

Reflejando la arquitectura de JADE que incluye un contenedor principal y varios contenedores periféricos, se pueden utilizar para otorgar permisos a los agentes dos tipos de ficheros de políticas:

- El fichero de políticas del contenedor principal, que especifica los permisos a nivel de plataforma, tales como "los agentes del usuario A pueden matar a los agentes del usuario B".
- Los ficheros de políticas de los contenedores periféricos (uno por cada contenedor), que especifican permisos específicos del contenedor, tales como "Los agentes del usuario A pueden matar a los agentes del usuario B en el contenedor local".

Los ficheros de políticas de los contenedores también regulan el acceso a recursos locales (JVM, sistema de ficheros, red, etc...).

### Integridad y confidencialidad de la mensajería

La firma y el cifrado garantizan un cierto nivel de seguridad cuando se envían mensajes, tanto a un agente corriendo en el mismo contenedor como o a uno corriendo en otro contenedor. Las firmas permiten asegurar la integridad de un mensaje (confianza en que los datos no se han modificado durante la transmisión) y la identidad del emisor del mensaje. Por otro lado, el cifrado asegura la confidencialidad del mensaje (confianza en que solamente el receptor original será capaz de leer el mensaje en claro). Un mensaje se compone de dos partes: el *envelope*, que contiene la información relacionada con el transporte, y el *payload*, que contiene la información sensible.

En JADE-S la "firma" y el "cifrado" siempre se aplican a todo el *payload* con el objetivo de proteger toda la información importante contenida en el mensaje (contenido, protocolo, etc). La información relacionada con la seguridad (como la firma, el algoritmo o la clave) se almacena en el *envelope*. Los usuarios no se encargan de los mecanismos de firma o de cifrado, sino que simplemente tienen que requerir que sus mensajes se firmen o comprobar que un mensaje recibido ha sido firmado. Si surge algún problema durante la firma, el cifrado, la verificación o el descifrado de los mensajes, estos serán descartados y se devolverá un mensaje de error al emisor del mensaje.

### IMTPoverSSL

Aparte del *plugin* JADE-S, existe otro mecanismo de seguridad para la plataforma JADE. Este mecanismo asegura confidencialidad, integridad de datos y autenticación mutua de conexiones entre contenedores JADE mediante la ejecución de IMTP (Internal Message Transport Protocol) sobre TLS/SSL, es decir RMI (Java Remote Method Invocation) [29] sobre SSL [30].

Esta estructura de comunicación entre contenedores es diferente de la estructura de comunicación entre agentes securizada mediante JADE-S. Cada contenedor posee un certificado que incluye, entre otras cosas, su clave pública. Además, todos los certificados de todos los contenedores de la plataforma se almacenan en el *trust store* de cada contenedor. La autenticación mutua, cifrado y firmas digitales se realizan mediante el protocolo TLS/SSL: cada contenedor presenta su propio certificado a su interlocutor en la comunicación y comprueba si el certificado de la otra parte se encuentra en su propio *trust store*. Si se aprueba con éxito la comunicación, ésta continúa con el cifrado y la firma digital de toda la información intercambiada por los contenedores.

Una aplicación práctica del uso de JADE-S y de ITMPoverSSL se encuentra en [31].

### Limitaciones

Como pone de manifiesto la Guía de Seguridad de JADE [32], existen una serie de limitaciones en la plataforma actual. Estas limitaciones son las siguientes:

- No existen permisos relacionados con la movilidad. Como consecuencia de esto, para conseguir un entorno seguro, una plataforma basada en JADE-S no debería ofrecer movilidad a los agentes.

- Parte de la comunicación intercambiada por los contenedores (los llamados comandos horizontales) se transfieren sobre canales seguros (SSL), pero no se firman. Un agente malicioso o un *hacker* podría enviar un comando horizontal falso con el objetivo de hacer que un contenedor remoto se comporte de una forma que difiera de la que debería comportarse.

- La mayoría de características de seguridad no están disponibles para agentes corriendo en dispositivos de tipo MIDP (Mobile Information Device Profile).

En un principio, se tenía previsto trabajar en solucionar estas limitaciones, pero en los últimos años los expertos en seguridad de la plataforma han abandonado el proyecto, con lo que algunas características no se han mejorado o refactorizado adecuadamente, ni se han podido realizar pruebas exhaustivas para asegurar su correcto funcionamiento. Con todo esto, los propios desarrolladores de JADE recomiendan no utilizar esta extensión de seguridad en entornos en producción. Dicho anuncio fue publicado por Giovanni Caire (el desarrollador principal de la plataforma JADE) en la lista de distribución de desarrolladores de JADE <jade-develop>[1]:

```
The latest version of JADE-S is 3.7 (the one you can download from the
    add-ons area of the JADE web site).
It works with JADE 4.0.
The main reasons why we do not encourage its usage in a real world
    deployment scenario are:
- Some features should be improved/refactored to make them easier to be
    used and more performing.
- Its level of testing is not sufficient
- Since some years, our group lost its security experts --> at present we
     cannot guarantee its
maintenance/evolution. No need to say that if someone is willing/able to
    make it evolve to meet his
requirements, we would be glad to help him taking over the ownership of
    the add-on.
```

A nivel administración y configuración de la plataforma, los procedimientos necesarios son sencillos y están bien explicados en la Guía de Seguridad de JADE [32]. Estos procedimientos son sencillos principalmente debido a que la funcionalidad ofrecida también lo es. No obstante, a nivel programación, se requiere invocar a servicios específicos seguros para el envío y la recepción de mensajes, lo cual supone tanto una complejidad añadida para el programador como un riesgo para el administrador de la plataforma, ya que por defecto estos métodos no son los utilizados.

[1] http://comments.gmane.org/gmane.comp.java.jade.devel/9215

### 2.2.3 Características de seguridad de Aglets

Aglets es una plataforma de agentes móviles de propósito general. Aglets [15], es un sistema de Agentes Móviles basado en Java. Aglets fue desarrollado por IBM Tokyo en 1996 y es mantenida por la comunidad *Open Source* desde 2001, si bien, el número de nuevas versiones desde entonces ha disminuido considerablemente. Uno de los puntos fuertes de la plataforma es que sigue el estándar *MASIF* [33].

Un *aglet* es un agente móvil escrito en Java. Se derivan de una clase abstracta llamada `Aglet`. Son objetos Java que se pueden mover de un servidor a otro a través de la red. La plataforma Aglets sigue un paradigma orientado a eventos análogo al de la clase `Applet` de la librería de Java. Cada *aglet* implementa una serie de manejadores de eventos que definen su comportamiento. Los agentes en Aglets son *single-threaded* y su modelo de comunicaciones se basa en el paso de mensajes, tanto de forma síncrona como asíncrona. Los agentes en Aglets utilizan *proxies* (similares a los *stubs* de RMI) como abstracción para referirse a agentes remotos (por ejemplo, para enviarles mensajes).

Si bien Aglets fue una de las primeras plataformas de agentes móviles, ya desde sus comienzos se tuvo en cuenta la seguridad en la plataforma, utilizando los mecanismos que se podían aplicar en su época.

Aglets usa una aproximación de organización donde todos los agentes en cierto dominio se consideran "confiables", y evalúa la autenticidad de un agente dependiendo del dominio por el que ha estado viajando [34]. Por lo tanto es importante ser capaces de autenticar al usuario de un agente y a su desarrollador. Es razonablemente sencillo identificar al desarrollador de un agente mediante la firma digital del código.

Dentro de Aglets se ha intentado encontrar un balance entre seguridad, complejidad y usabilidad que permite su uso en entornos de producción. Los siguientes mecanismos de seguridad están soportados por Aglets:

- Autenticación de usuarios y dominios.
- Comprobación de integridad en las comunicaciones entre los servidores de un dominio.
- Autorización similar al modelo de seguridad de JDK (Java Development Kit) 1.2.

Pasamos a describir cada uno de los mecanismos.

#### Autenticación de Dominios

Los servidores de Aglets son capaces de autenticar si otro servidor pertenece a cierto dominio. Todos los servidores que pertenecen a un domino específico comparten una

clave secreta, y se pueden autenticar entre los servidores que pertenecen al dominio mediante dicha clave usando una función MAC (Message Authentication Code). La ventaja de este método es que este MAC no tiene que ser firmado mediante algoritmos de cifrado y se puede implementar directamente usando el JDK.

Tras la autenticación entre los servidores, las credenciales del *aglet* se envían junto con el agente. El receptor deberá decidir cuánto confía en las credenciales enviadas por el servidor en base a la información obtenida en la autenticación. En Aglets, el servidor simplemente confía en las credenciales si éstas fueron enviadas por un servidor en el mismo dominio.

Para utilizar la autenticación de dominios, cada usuario (o administrador del servidor) debe obtener la clave secreta del dominio de la autoridad de seguridad. La autoridad es responsable de generar las claves para cada servidor específico. La clave secreta compartida es firmada con la clave del usuario, por lo que el usuario debe introducir su clave correctamente para poderla utilizar. El fichero de la clave debe mantenerse en secreto, ya que no está cifrado.

Una desventaja es que no se pueden identificar y verificar los servidores que inician una comunicación. Si la clave compartida es robada de un servidor del dominio, no hay forma de diferenciar a los servidores válidos de aquellos que se han apropiado de la clave. Por lo tanto, todos los servidores del dominio se exponen a peligros.

#### Comprobación de integridad en las comunicaciones

Como ya se ha explicado, todos los servidores en un mismo dominio de Aglets comparten una clave secreta. La comprobación de integridad se puede realizar de la misma forma que se realiza la autenticación del dominio. El emisor calcula un valor MIC (Message Integrity Code, análogo al MAC) del contenido de la clave compartida, y lo envía junto con el contenido del mensaje. El receptor verifica el MIC usando la clave compartida, el contenido y el valor. Como solo un servidor que conoce la clave compartida puede generar el mismo MIC, el receptor valida que el contenido del mensaje fue enviado por un servidor del mismo dominio y que no ha sido modificado.

#### Identificación de Código

En la actual especificación de la plataforma Aglets, solo se utiliza el *codebase* para la identificación del agente. En dicha especificación, no se soporta la firma digital de código.

### Autorización de Aglets

Cuando un agente aglet accede a información sensible o a recursos como las propiedades de Java, *threads*, y/o a cualquier otro recurso externo como ficheros, se deben controlar los permisos otorgados al *aglet*. Los permisos se pueden especificar mediante un interfaz gráfico o directamente editando la base de datos de políticas. El formato de la base de datos de políticas usado por Aglets está diseñado para cumplir con la especificación de JDK 1.2. Un usuario puede especificar los siguientes permisos en la base de datos de políticas:

```
java.io.FilePermission      : File read/write/execite
java.net.SocketPermission   : Socket resolve/connect/listen/accept
java.awt.AWTPermission      : showWindowWithoutWarningBanner,
    accessClipboard
java.util.PropertyPermission    : Java Property
java.lang.RuntimePermission     : queuePrintJob, load library
java.security.SecurityPermission: getPolicy, setSystemScope
java.security.AllPermission     : all other permissions
com.ibm.aglets.security.ContextPermission   : contextproperty,start,
    shutdown
com.ibm.aglets.security.AgletPermission     : dispatch, deactivate, etc.
com.ibm.aglets.security.MessagePermission   : messaging
```

Donde:

- com.ibm.aglets.security.ContextPermission: `ContextPermission` especifica si un *aglet* puede acceder a las propiedades del contexto, apagar el contexto, etc. El nombre para un `ContextPermission` puede ser uno de los siguientes: “start”, “retract”, “create.<codebase@classname>”, “listener.add”, “listener.remove”, “property.<key>”.

- com.ibm.aglets.security.AgletPermission: esta clase representa el acceso a un *aglet*. Un `AgletPermission` consiste en el nombre de un *aglet* y un conjunto de operaciones para dicho *aglet*.

  El nombre del *aglet* es el nombre de usuario del *aglet* (por ejemplo, “jorge”, “sergio”, “anonimo”), y la operación es el nombre de un método de la clase `Aglet` (por ejemplo “dispatch”, “dispose”).

- com.ibm.aglets.security.MessagePermission: esta clase representa el permiso para enviar un mensaje a un *aglet*. Un `MessagePermission` consiste en el nombre del *aglet* y un tipo de mensaje.

  El nombre del *aglet* es el nombre de usuario del *aglet* (por ejemplo, “jorge”, “sergio”, “anonimo”), y el tipo de mensaje es el tipo de mensaje a enviar. El tipo de mensaje debe tener como prefijo “message” (por ejemplo, “message.show”, “message.getResult”).

Los Aglets se identifican por su *codebase* y por su propietario. El desarrollador de un *aglet* es, según la especificación actual, anónimo, ya que no se soporta la firma digital de código en la plataforma.

Mas detalles sobre la arquitectura de seguridad de Aglets, incluyendo una especificación mas extensa sobre el lenguaje utilizado para la gestión de permisos se pueden encontrar en [35].

### 2.2.4 Características de seguridad de Tryllian

Tryllian [36] se desarrolló por la compañía homónima en 2001 (la última versión 3.2.0, fue distribuida como *Open Source* en 2005). Sin embargo, a día de hoy, la página comercial del producto parece que no tiene actividad, y la página del proyecto *Open Source* (http://www.tryllian.org) ya no existe.

Está basado en un mecanismo de acción-reacción. Permite a los programadores definir un comportamiento reactivo (basado en un mensaje de entrada) y un comportamiento proactivo de los agentes. Tryllian propone un modelo de programación basado en tareas y la comunicación entre agentes se realiza mediante paso de mensajes de acuerdo con el estándar FIPA. También provee de un servicio de persistencia. La mayor desventaja de Tryllian es que no ofrece transparencia de localización, es decir, es necesario conocer la dirección exacta de un agente para poder comunicarse con el (si éste ha viajado, lógicamente hay que conocer su nueva dirección). Tampoco ofrece mecanismos para la comunicación síncrona ni para la invocación de métodos.

La plataforma Tryllian se conforma de [37]:

- Hábitat. Un hábitat es una colección de una o más habitaciones que comparten una misma JVM. Provee los servicios de gestión del ciclo de vida de los agentes, comunicación movimientos inter-hábitat, persistencia de agentes y habitaciones y un modelo de seguridad básico.
- Habitaciones. Los agentes solo pueden existir en habitaciones. Las habitaciones son los entornos en los que los agentes interactúan. También son el sitio donde los agentes anuncias sus capacidades para que otros agentes las encuentren.
- Agentes. Un agente Tryllian es un componente software que consiste en dos partes: cuerpo y comportamiento. El cuerpo es la parte que ejecuta el código del agente mediante el envío de mensajes y el movimiento del código del agente por la red. El comportamiento consiste en las acciones y en el estado del agente.
- Agentes del Sistema. Son agentes especiales que ofrecen los servicios de los hábitats. Los mas importantes son el agente hábitat (permite las comunicaciones en un hábitat), el agente habitación (ofrece y gestiona información sobre los agentes

que hay en una habitación determinada) y el agente transportador (es el punto de interacción para que los agentes puedan moverse).

El ADK (Tryllian Agent Development Kit) de Tryllian implementa el modelo completo de seguridad de Java usando certificados y permisos [37]. Para el desarrollador, la seguridad se reduce a firmar el agente con la herramienta de firmado de ficheros JAR provista por el JDK.

#### Permisos en el Hábitat

Al más bajo nivel, el dueño de un hábitat tiene que decidir qué acciones un hábitat, o más específicamente, un ARE (Tryllian Agent Runtime Environment), puede realizar en la plataforma que se está ejecutando: a qué recursos puede acceder, etc. El ARE es el *software* de servidor que implementa un hábitat.

Los permisos del ARE no se encuentran directamente relacionados con los permisos de los agentes. Esto se hace para dotar al ARE de más permisos de los que podrían disfrutar los agentes sin crear un agujero de seguridad y para separar completamente las clases de un agente en el ARE de las clases de otros agentes.

#### Permisos de los Agentes

Para permitir agentes externos en un hábitat de una forma segura se necesita controlar sus permisos. Con el objetivo de decidir qué agente disfrutará de ciertos permisos y qué agente no, se tiene que poder determinar dónde se ha originado el agente. Esto se consigue mediante la inclusión de un certificado dentro de los ficheros del DNA (Tryllian Definition of New Agent) del agente.

Con este certificado, se puede averiguar quién creo el agente y quién confía en él. El hábitat determina quién puede entrar en él mediante la asignación de permisos a los certificados. El mecanismo usado por el ARE es similar al mecanismo utilizado por los navegadores para los *applets* [38].

#### Otros mecanismos de seguridad

El ARE protege a los agentes de otros agentes: los agentes no pueden leer o o modificar el código o los datos o interceptar los mensajes para otros agentes. Todas las transmisiones entre hábitats se pueden cifrar usando SSL. Esto protege tanto a los agentes móviles como a sus mensajes. El ARE también asegura que solo los hábitats en los que confía pueden comunicarse.

Las comunicaciones entre el hábitat y las diferentes herramientas del ADK también se pueden cifrar.

### 2.2.5 Características de seguridad de Voyager

Voyager (http://www.recursionsw.com), fue desarrollado inicialmente por *ObjectSpace* en 1997 y actualmente pertenece a *Recursion Software* (última versión: Voyager 8.0 de 2011). Es un *middleware* de computación distribuida enfocado a simplificar la gestión de comunicaciones remotas de protocolos tradicionales [39] como CORBA (Common Object Request Broker Architecture) [40] o RMI.

Voyager provee capacidades de envío de mensajes y también permite el movimiento de agentes a través de la red. Es una plataforma con ciertas funcionalidades que facilitan el desarrollo de sistemas distribuidos. Ofrece servicios como la generación dinámica de *proxies* CORBA, de código móvil y de agentes móviles. Los agentes se comunican vía invocaciones remotas de métodos usando *proxies*. Voyager ofrece a los agentes tiempos de vida flexibles, pudiéndose configurar cuándo se quiere que termine un agente de varias formas (cuando no tenga mas referencias remotas, cuando haya expirado su tiempo de vida máximo, etc.).

Como se cita habitualmente en la literatura [41, 42], uno de los problemas de la plataforma Voyager es que se trata de un producto comercial que no se encuentra disponible gratuitamente, lo que puede hacer que muchos investigadores y usuarios eviten su uso, en favor de otras alternativas más accesibles. De todas formas, la documentación ofrecida por *Recursion Software* es abundante y de gran calidad y, además, es posible acceder a un mes de prueba de la plataforma gratuitamente, así como disponer de un mes de soporte gratis por parte de *Recursion Software*, para poder evaluar la plataforma.

En Voyager, una política de seguridad se aplica a las entidades que se reconocen dentro de un mismo dominio de seguridad, y otorga permisos y privilegios a dichas entidades dentro del dominio [43]. Una entidad se autentica contra una política y el código de la aplicación verifica que un permiso o un privilegio fue otorgado a dicha entidad (asociada al *thread* actual).

La seguridad en Voyager provee un marco y una implementación para crear una aplicación distribuida segura mediante mecanismos basados en políticas.

La autenticación y la verificación de permisos se realiza mediante un interfaz estándar provisto por el *Voyager Security API* utilizando un proveedor de políticas de seguridad extensible. Mediante la extensión de dicho proveedor de seguridad, Voyager puede utilizar desde mecanismos sencillos de autenticación mediante usuario y contraseña hasta certificados con claves públicas/privadas.

#### Confianza entre código y contexto

Voyager sigue el modelo de *sandbox* de la JVM, utilizado típicamente en los navegadores para los *applets*. Para que se pueda confiar y, por lo tanto, ejecutar una clase Java, es necesario que el código esté firmado por una autoridad válida. La *sandbox* de la JVM provee barreras a través de las cuales el código "no confiable" no se puede ejecutar. Estas barreras se activan mediante la creación de una instancia de una subclase de `java.net.SecurityManager`. Solo puede haber una instancia de `java.net.SecurityManager` por JVM.

Desde el momento en que una entidad se autentica en el domino de seguridad de la plataforma (utilizando el método que haya sido definido), ésta es capaz de comprobar si tiene los privilegios suficientes para ejecutar una operación en particular, tanto si está ejecutando en local como en una JVM remota.

#### Principales mecanismos de seguridad

El lenguaje Java y la JVM evitan el acceso a los datos en memoria por parte de otros procesos y ofrecen verificación del *bytecode* durante la carga de las clases.

Para proteger los datos mientras se mueven por una red pública, Voyager permite la utilización del protocolo SSL para proveer tanto privacidad en los datos como integridad de los mismos durante la transmisión. La comprobación de integridad de los datos garantiza que estos no pueden ser modificados durante su transferencia entre servidores. La privacidad de los datos (o cifrado) hace que no puedan ser leídos en la comunicación entre diferentes *hosts* por un sistema ajeno.

Con el objetivo de evitar el repudio del uso de recursos por parte de una entidad, se utiliza un mecanismo sencillo de *logging* que identifica quién está ejecutando instrucciones en cada uno de los servidores.

También se permite reemplazar temporalmente los permisos necesarios para la ejecución de ciertas partes del código. Es decir, si una entidad no tiene permisos suficientes para hacer una consulta a base de datos, por ejemplo, ésta tiene mecanismos para pedir permisos temporales para realizar dicho acceso.

### 2.2.6 Resumen de medidas de seguridad en diferentes plataformas

A modo de resumen, nos gustaría exponer en la Tabla 2.2 cómo se comportan las plataformas estudiadas según las principales medidas de seguridad esgrimidas en esta sección, y más detalladamente, según la Tabla 2.1.

En dicha tabla podemos observar cómo todas las plataformas estudiadas disponen

| | SeMoA | Jade | Aglets | Tryllian | Voyager |
|---|---|---|---|---|---|
| **Autenticación** | Sí | Jade-S | Sí | A nivel de JAR | Sí, programable |
| **Confidencialidad** | | | | | |
| **- Agentes** | Sí | No | No | No | No |
| **- Mensajes** | Sí | Jade-S | A nivel de dominio | No | No |
| **- Comunicaciones** | Sí | IMTPoverSSL | No | Sí | Sí |
| **Integridad** | | | | | |
| **- Plataforma** | Sí | Sí | Sí | Sí | Sí |
| **- Agente** | Sí | Sí | Sí | Sí | Sí |
| **- Comunicaciones** | Sí | IMTPoverSSL | No | Sí | Sí |
| **Responsabilidad** | Sí | No | No | No | No |
| **Disponibilidad** | JVM | JVM | JVM | JVM | JVM |

Tabla 2.2: Comparativa de medidas de seguridad en las plataformas estudiadas

de algún mecanismo de autenticación. Desde mecanismos completamente programables como ofrece Voyager a un simple mecanismo basado en firma del JAR del código como utiliza Tryllian.

Respecto al cifrado de los elementos, tan solo SeMoA es capaz de ofrecer un mecanismo de cifrado de los agentes. Tanto SeMoA como JADE-S permiten utilizar métodos para intercambiar información cifrada utilizando cifrado de clave asimétrica entre dos elementos de la plataforma. Aglets también es capaz de cifrar información, pero únicamente mediante un cifrado con clave simétrica que permite que todos los miembros de un mismo dominio sean capaces de descifrar un dato, ya que todos los miembros del dominio tienen conocimiento de la clave de cifrado. Todas las plataformas exceptuando Aglets son capaces de cifrar el canal de comunicaciones mediante SSL.

Todas las plataformas disponen de sistemas basados en JAAS para controlar en alguna medida las acciones que los agentes pueden llevar a cabo dentro de los contenedores. Los controles pueden ser estrictos y completamente extensibles como ocurre en SeMoA o más básicos como los ofrecidos por Tryllian o Aglets. La JVM protege el acceso a zonas de memoria no autorizadas entre los procesos así que, en principio, ningún proceso debería poder acceder y modificar datos de un agente o de un contenedor en ejecución. Solo Aglets no ofrece un mecanismo de comunicaciones basado en SSL, con lo que es la única plataforma que no puede asegurar la integridad de las mismas.

SeMoA es la única plataforma que dispone de mecanismos de responsabilidad sobre las acciones efectuadas ya que permite la firma mediante clave privada por parte de un agente. El resto de plataformas utilizan mecanismos básicos de *logging* para analizar quién está ejecutando acciones.

Todas las plataformas estudiadas corren sobre un entorno de ejecución basado en una JVM sin modificar. Esto hace que los problemas expuestos en la Sección A.2 respecto al uso de la JVM les afecten. Para poder mitigar este tipo de ataques de denegación de servicio y de abuso de recursos de los contenedores, Java debería ofrecer mecanismos

avanzados de control de recursos.

A nivel de seguridad queda claro que SeMoA es la plataforma de agentes móviles más avanzada de las estudiadas. Si bien su desarrollo se encuentra paralizado desde que el director del proyecto dejó el *Fraunhofer-Institut für Graphische Datenverarbeitung*, aunque su código se encuentra disponible como *Open Source*. Es posible que aun siendo tan avanzada a nivel de seguridad, la falta de mantenimiento de la plataforma, unida a la complejidad de uso y de administración de la misma haga que otras plataformas menos seguras en teoría (pero quizá lo suficiente a nivel práctico) pero mucho mejor documentadas y mantenidas como JADE o Voyager sean el referente del mercado de plataformas de agentes móviles.

# Capítulo 3

# Arquitectura de Seguridad para SPRINGS

SPRINGS es una plataforma de agentes móviles, desarrollada por la Universidad de Zaragoza, cuyo principal foco es la escalabilidad y la fiabilidad en escenarios con un gran número de agentes móviles. Al igual que otras plataformas, como Grashopper y Jade, ofrece una arquitectura basada en regiones. También ofrece total transparencia en la localización 1) para movimientos (el programador no necesita especificar la dirección de red, sino simplemente el nombre del contexto al que desea viajar) y 2) para invocación a métodos y envío de mensajes a otros agentes a través de *proxies* dinámicos [41]. SPRINGS implementa un mecanismo de movilidad "débil" [44] basado en llamadas a métodos [12].

La plataforma consta de agentes, contextos y regiones. Los agentes se crean en contextos, los cuales son entornos de proceso donde los agentes se pueden ejecutar. Los contextos, además, proveen los servicios de comunicación y movimiento a los agentes. Un conjunto de contextos definen una región. Una región ofrece a los agentes que se están ejecutando y a los contextos de un servicio que garantiza un espacio de nombres único a través de un elemento llamado RNS (Region Name Server) cuya función es gestionar los elementos que forman parte de la región.

SPRINGS está implementada en Java usando una JVM estándar y utiliza RMI como mecanismo de comunicación entre todos sus elementos. Por estar basada en Java, SPRINGS se beneficia de las medidas de seguridad expuestas en el Apéndice A.2 como, por ejemplo, la imposibilidad de que se pueda acceder a los datos de un objeto desde otro *thread*, aunque puede tener problemas, por ejemplo, ante ataques de denegación de servicio.

Cabe resaltar que la arquitectura de seguridad implementada para SPRINGS se ha diseñado como una extensión a la plataforma. SPRINGS no se diseñó originalmente

basándose en principios de seguridad, sino que se hizo énfasis en la creación de una plataforma capaz de gestionar una gran cantidad de agentes móviles de una manera eficiente [12]. Por esta razón, un requisito básico para la arquitectura de seguridad de SPRINGS es que debe poderse activar y desactivar a voluntad por parte del usuario o del administrador de la plataforma. En caso de que la extensión de seguridad de SPRINGS se encuentre desactivada, el rendimiento de la plataforma no debe verse penalizado en gran medida. Dicha desactivación puede ser deseable en ciertos entornos controlados, en los que prime la necesidad de rendimiento y no se adviertan riesgos de seguridad a priori. De todas formas, se recomienda la utilización de la extensión de seguridad como práctica habitual en el diseño de un sistema de agentes móviles basado en SPRINGS.

En la Figura 3.1 se ilustran las diferentes capas de seguridad que se han diseñado para dotar a la plataforma SPRINGS de los mecanismos de seguridad más importantes para cumplir con los principios generales de seguridad de sistemas descritos en el Capítulo 2.

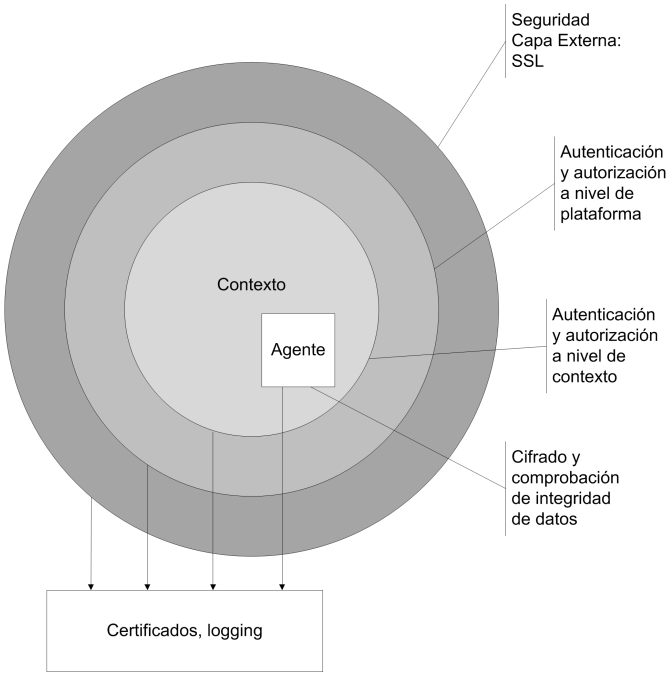



Figura 3.1: Capas de seguridad de SPRINGS

En dicha figura se muestran los siguientes mecanismos:

- Certificados. Todos los elementos que forman parte de la plataforma disponen de un certificado o de una estructura creada a partir de un certificado para su identificación por el resto de elementos de la plataforma
- Seguridad en la Capa Externa. Todas las comunicaciones entre los elementos de la plataforma pueden hacer uso del protocolo SSL para evitar que las mismas puedan

ser interceptadas y/o alteradas.

- Autenticación y autorización a nivel de plataforma. Se denominan así a los mecanismos que se utilizan entre los contextos y el RNS para autenticarse y permitir el acceso a ciertas funcionalidades como la creación de agentes y el movimiento de los mismos.
- Autenticación y autorización a nivel de contexto. Este es el mecanismo a través del cual un contexto puede aceptar y/o limitar el acceso a sus recursos por parte de un agente.
- Cifrado y comprobación de integridad de datos. Son los mecanismos disponibles para un agente para poder cifrar y comprobar la integridad de datos que él posee con el objetivo de evitar que se pueda acceder a estos o que puedan ser modificados.
- *Logging*. Se ha dotado a la plataforma de un mecanismo unificado de *logging* para que toda la gestión de *logs* de lo elementos se realice de una forma homogénea.

A continuación pasamos a detallar cada una de las medidas de seguridad anunciadas.

## 3.1 Identificación - Uso de certificados digitales

Un certificado digital es un documento digital mediante el cual un tercero confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y una clave pública [45, 46].

Todos los elementos estructurales de la plataforma SPRINGS (RNS y contextos) se encuentran identificados unívocamente por un certificado digital. Los certificados digitales utilizados en una región pueden estar expedidos por una misma CA (Autoridad de Certificación) o por varias (o incluso puede tratarse de certificados auto firmados [46]). Sin embargo es necesario que las JVMs sobre las que corren los elemento de la plataforma sean capaces de confiar en todos los certificados de las CAs (o, directamente, en los certificados auto firmados) utilizadas. La gestión de la expedición de los certificados se considera una tarea paralela pero ajena al funcionamiento de la plataforma en sí. En la Figura 3.2 se muestra el diagrama de gestión de los certificados en una región.

En el que:

- El administrador de la plataforma debe incluir la clave pública de la CA con la que se haya firmado el certificado del RNS en el fichero `cacerts` de la JVM en la que se vayan a ejecutar tanto el RNS como los contextos.
- El administrador de la plataforma solicita un certificado digital para el RNS a la CA, que contiene una clave privada que solo será conocida por él.

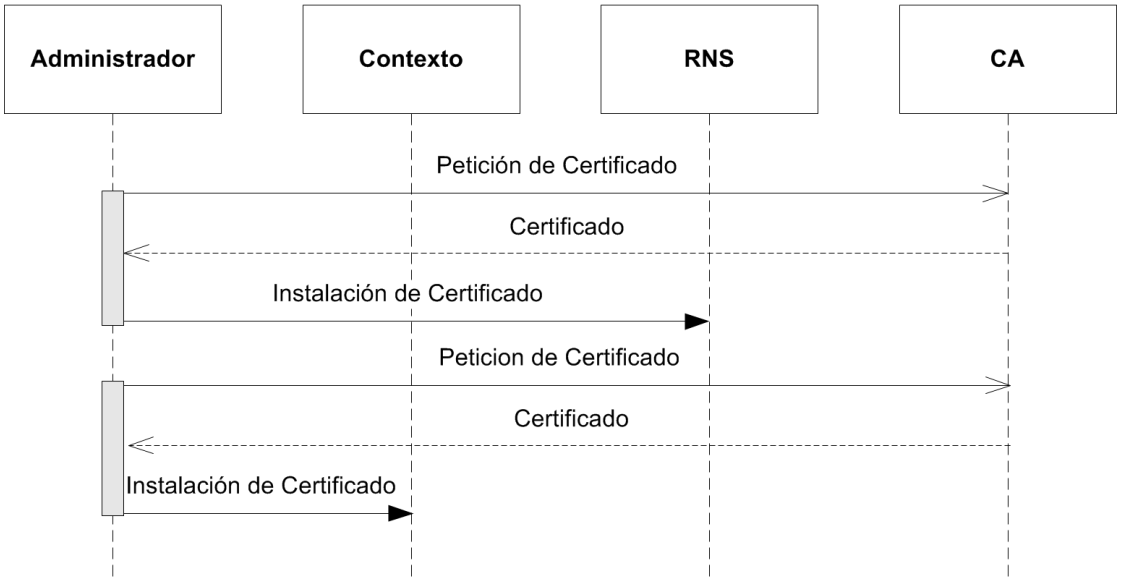



Figura 3.2: Identidad, uso de certificados digitales

- El administrador de la plataforma (o el de un contexto, dependiendo de cómo se esté administrando la plataforma) solicita un nuevo certificado digital para el contenedor a la misma CA que firmó el certificado digital del RNS o a otra CA. Si se utiliza un certificado digital firmado por otra CA diferente a la del RNS es necesario que se incluya la clave pública de dicha CA en el fichero `cacerts` de las JVMs en la que se vayan a ejecutar tanto el RNS como los contextos.

  Por lo tanto, para simplificar la administración de la plataforma, es recomendable que todos los certificados utilizados en una región estén firmados por la misma CA.

- El certificado digital del contexto está firmado con una clave que solo debe ser conocida por el administrador del contexto y se encuentra expedido con el nombre único del contexto, por lo que solamente ese contexto podrá utilizarlo.

Cuando un contexto crea un agente, éste hereda sus credenciales de identificación del contexto que lo ha creado. Por lo tanto, como veremos más adelante en las Secciones 3.3 y 3.4, todas las tareas de autenticación y autorización se realizan a nivel de contexto. La principal razón para esta decisión de arquitectura es facilitar la gestión de la seguridad en la plataforma. La identificación por agente supondría tener que crear certificados, distribuirlos e instalarlos en los contextos y en el RNS, así como gestionar los diferentes niveles de autorización para cada agente en la plataforma. Estas tareas administrativas harían la gestión de una plataforma orientada a soportar gran número de agentes como es SPRINGS casi imposible. Esta decisión tiene un efecto lateral en el funcionamiento de la plataforma, no es posible crear un agente sin haber creado un contexto primero. Todo agente se debe crear dentro de un contexto, del que heredará sus credenciales de

identificación.

Las reglas generales en las que se basa la identificación en la plataforma son:

- El RNS se comporta como la autoridad principal de comprobación de identidades. Esto es consecuente con la arquitectura centralizada que ofrece SPRINGS, aunque puede suponer una limitación en entornos móviles como se expone en [47].
- Cada contexto que se cree y se registre en el RNS debe ofrecer un certificado firmado por una CA en la que confíen las JVMs del resto de elementos de la plataforma.
- La autenticación de los contextos ante el RNS está basada en el uso de certificados válidos y de la existencia de dichos certificados en el *keystore* del RNS, como se detalla en la Sección 3.3.
- Los agentes heredan las credenciales de identificación de los contextos en los que fueron creados.

En siguientes revisiones de la plataforma sería recomendable abordar temas como la gestión de la revocación de certificados o, incluso, la utilización de *frameworks* de gestión de identidades como OpenID [48] que quedan fuera del alcance del presente proyecto ya que, por sí mismos, no aportan más seguridad a la plataforma, sino que facilitan la gestión de la seguridad de la misma, haciendo posible la delegación de gestión de las identidades y su autenticación por parte de entidades externas.

## 3.2 Seguridad en la capa externa

Como se ha explicado anteriormente, el mecanismo utilizado para las comunicaciones entre todos los elementos de la plataforma SPRINGS es RMI.

Por defecto, el mecanismo de comunicación de RMI, llamado JRMP (Java Remote Method Protocol), no es seguro. En Java 5.0 se introdujeron dos nuevas clases, `javax.rmi.ssl.SslRMIClientSocketFactory` y `javax.rmi.ssl.SslRMIServerSocketFactory` que permiten proteger el canal de comunicaciones entre el cliente y el servidor de una aplicación basada en RMI usando los protocolos SSL/TLS [49]. Estas clases ofrecen una forma elegante de proteger las comunicaciones RMI mediante el uso de JSSE, lo cual asegura la integridad de los datos, la confidencialidad de los mismos (a través del cifrado) y la autenticación de clientes y servidores. Esto se consigue mediante el cifrado de clave simétrica para cifrar los datos entre el cliente y el servidor, y el cifrado de clave asimétrica (o cifrado mediante clave pública/privada) para autenticar las identidades de las partes que se quieren comunicar, así como para cifrar la clave secreta común que se utiliza durante el establecimiento de la sesión SSL [50].

Para proteger la capa externa de la plataforma se ha decidido utilizar estos mecanismos. Con ellos se evita el acceso por parte de cualquier tercero a los información en tránsito perteneciente a la plataforma.

En la Figura 3.3 se expone cómo todas la comunicaciones entre los elementos de la plataforma se pueden proteger mediante RMI-SSL. Tanto las interacciones entre los contextos y el RNS como las interacciones entre diferentes contextos para el movimiento y la comunicación de los agentes se protegen de esta forma.

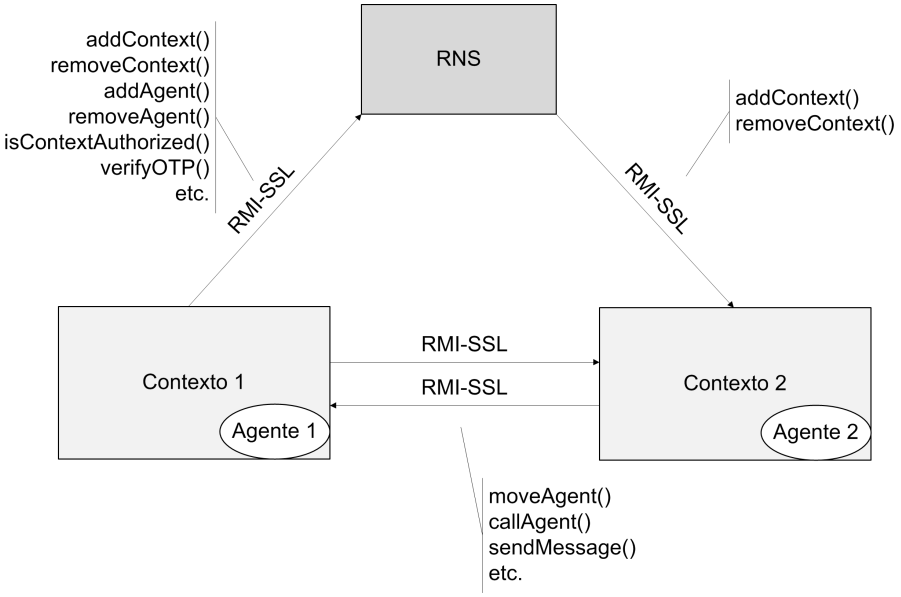



Figura 3.3: Comunicaciones RMI-SSL

Una vez establecidas las comunicaciones SSL correctamente, todos los elementos de la plataforma pueden comunicarse entre sí de una forma segura.

Siguiendo los requisitos previamente comentados, esta capa de seguridad se puede desactivar mediante cambios en la configuración. Sin embargo, no se recomienda su desactivación, ya que ésta implicaría que todas las comunicaciones de la plataforma se podrían interceptar, no solo exponiendo los datos de los agentes que viajan por la red sino el resto de credenciales de autenticación que se detallan en la Sección 3.3, pudiendo facilitar ataques de suplantación de identidad.

## 3.3 Autenticación y autorización a nivel de plataforma

Con el objetivo de poder controlar qué elementos acceden a la plataforma y qué acciones son capaces de realizar se han desarrollado unos mecanismos específicos de

autenticación y autorización a nivel de plataforma. Estos mecanismos ofrecen una manera sencilla de proteger la plataforma ante ataques por parte de contextos y agentes "malignos" mediante, por una parte, la limitación en el acceso a la plataforma y, por otra, la limitación en las acciones que estos contextos pueden realizar. Este mecanismo se podría extender en un futuro para ofrece un control más detallado sobre las acciones.

La autenticación dentro de la plataforma dispone de dos vertientes principales:

- Autenticación y autorización de los contextos y de los agentes ante el RNS.
- Autenticación del RNS frente a los contextos.

La primera trata de controlar qué contextos y qué agentes pueden formar parte de la plataforma y qué acciones pueden ejecutar en ella y la segunda trata de asegurar que un contexto solo es contactado por el RNS de su región, y no por ninguna otra entidad que se pueda hacer pasar por él, evitando ataques por parte de la plataforma a un contexto.

### 3.3.1 Autenticación y autorización de los contextos y de los agentes ante el RNS

Gracias a la autenticación y autorización a nivel de plataforma se puede controlar:

- Qué contextos se pueden añadir a la plataforma.
- Qué contextos pueden crear y destruir agentes.
- Qué agentes pueden realizar llamadas a otros agentes.
- Qué agentes pueden viajar a otros contextos.

Este mecanismo basa su funcionamiento en el uso del RNS como una autoridad de autenticación que centraliza la gestión de las identidades de los contextos y agentes, así como su autenticación y sus permisos o niveles de autorización.

Esta arquitectura centralizada ofrece grandes ventajas en la gestión de la seguridad de la plataforma como la necesidad de la intervención en un solo elemento de la misma para añadir, borrar o modificar las credenciales de nuevos contextos mediante la inclusión de las claves públicas de los contextos en el *keystore* del RNS, la confianza en un único elemento de la plataforma para realizar estas funciones, evitando la posibilidad de que otros elementos puedan modificar las credenciales o los permisos otorgados, etc. Si bien, desde el punto de vista del rendimiento y la escalabilidad de la plataforma sobre todo en entornos con dispositivos móviles [47], quizá habría que estudiar el uso de alguna

característica adicional para mejorar la eficiencia de una plataforma centralizada. Sin embargo, dado que el RNS es un elemento ya existente en la plataforma que se utiliza simplemente para la gestión de la misma y no para el propio trabajo de los agentes, es un elemento ideal para ofrecer esta funcionalidad.

El funcionamiento mostrado en la Figura 3.4, dónde se detallan los métodos invocados, es el siguiente:

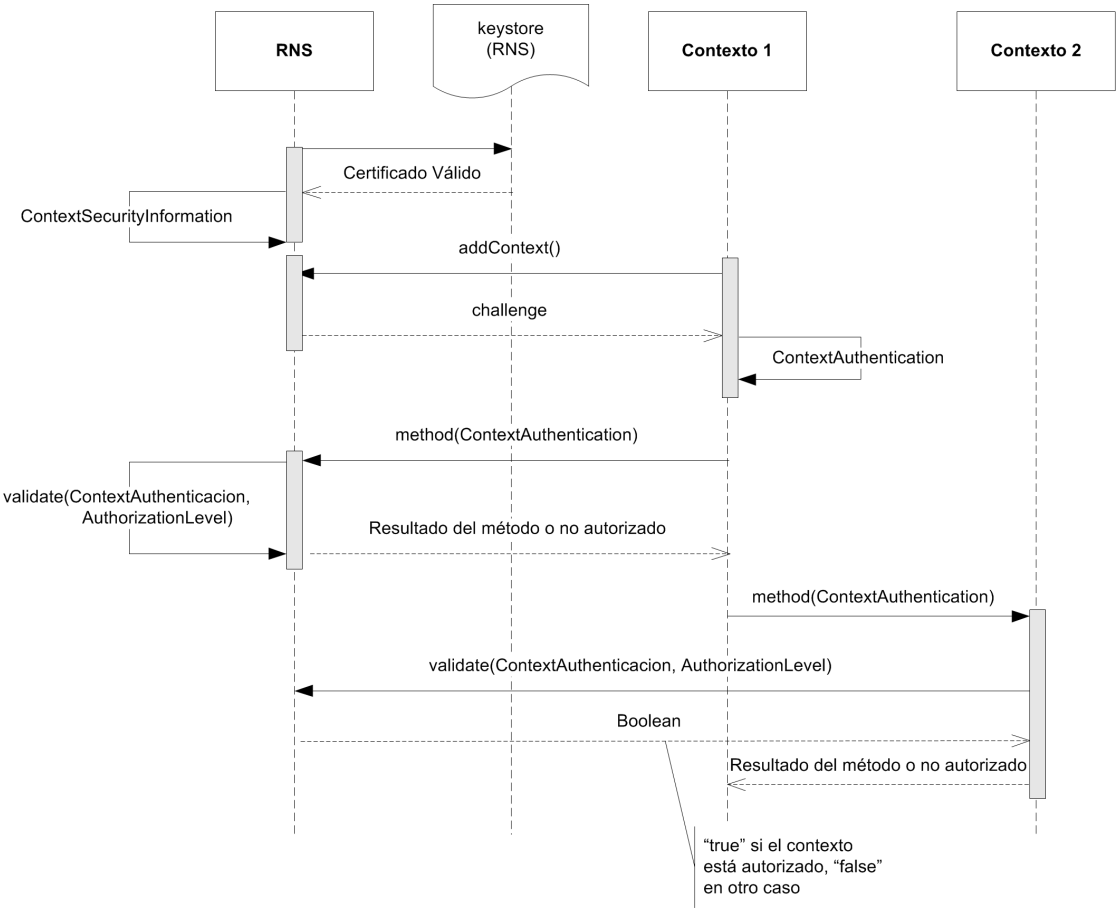



Figura 3.4: Autenticación y autorización de los contextos

- Al arrancar, el RNS analiza el contenido de su *keystore* y, para cada certificado válido, crea una estructura del tipo `ContextSecurityInformation` (Figura 3.5) en la que se se almacena: el nombre del contexto, su clave pública, un conjunto de permisos y un *challenge*. Este *challenge* se trata de una serie aleatoria de *bytes* (obtenidos a través de la clase `SecureRandom`). Por defecto, cuando se crea un nuevo objeto de este tipo, se le incluye el permiso `springs.security.authorizationLevel.PLATFORM_PERMISSION` (la función y el uso de los permisos se explicará más adelante). Todos los objetos del tipo

`ContextSecurityInformation` se almacenan en una estructura privada llamada `_authorizedCtxs` del tipo `hashtable` en el RNS, como se ve en el diagrama de clases de la Figura 3.6.

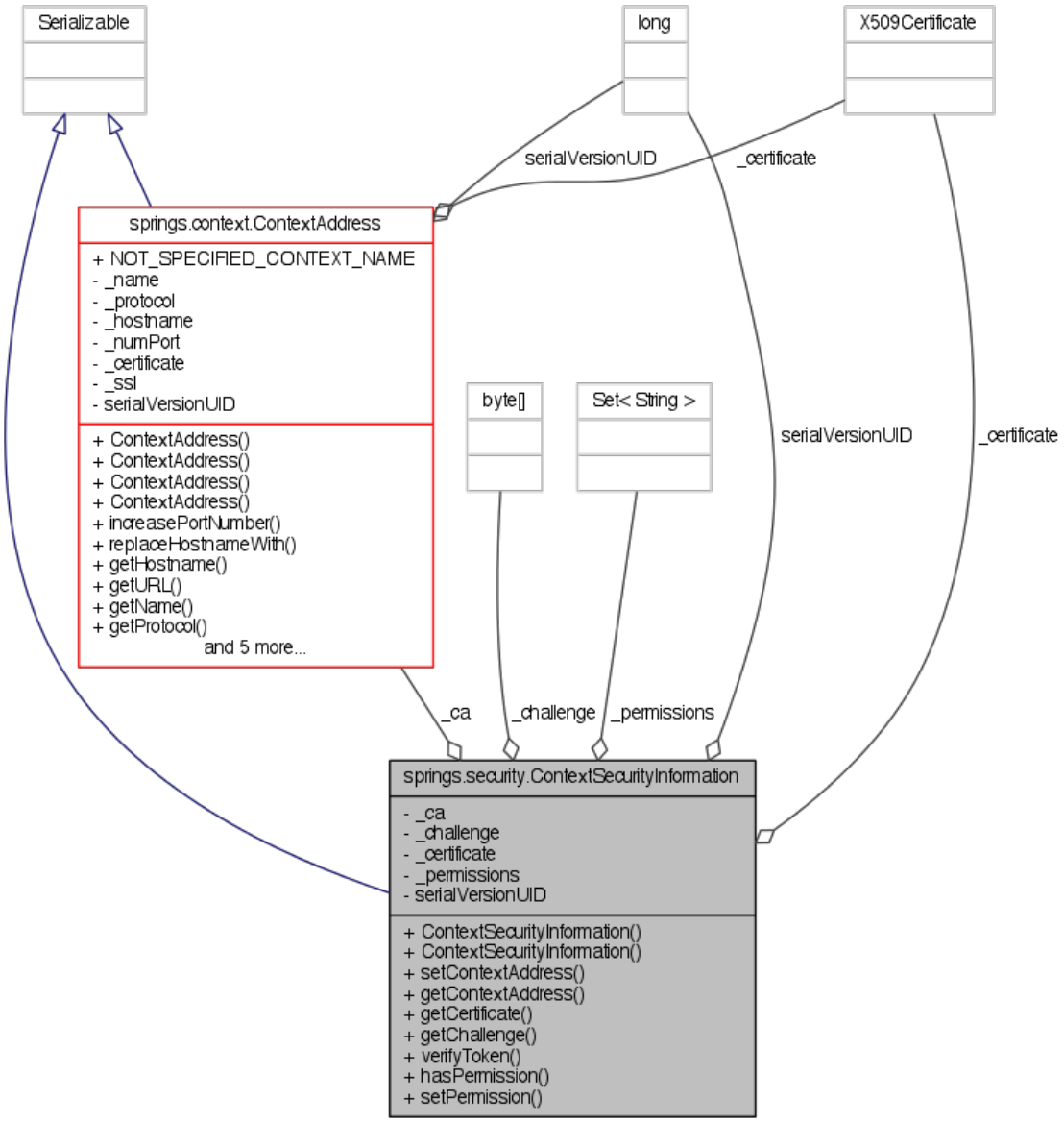



Figura 3.5: Clase ContextSecurityInformation

- Cuando un contexto se quiere unir a una región, realiza una invocación al método `addContext` del interfaz RMI del RNS. Éste método devuelve al contexto el *challenge* calculado por el RNS.

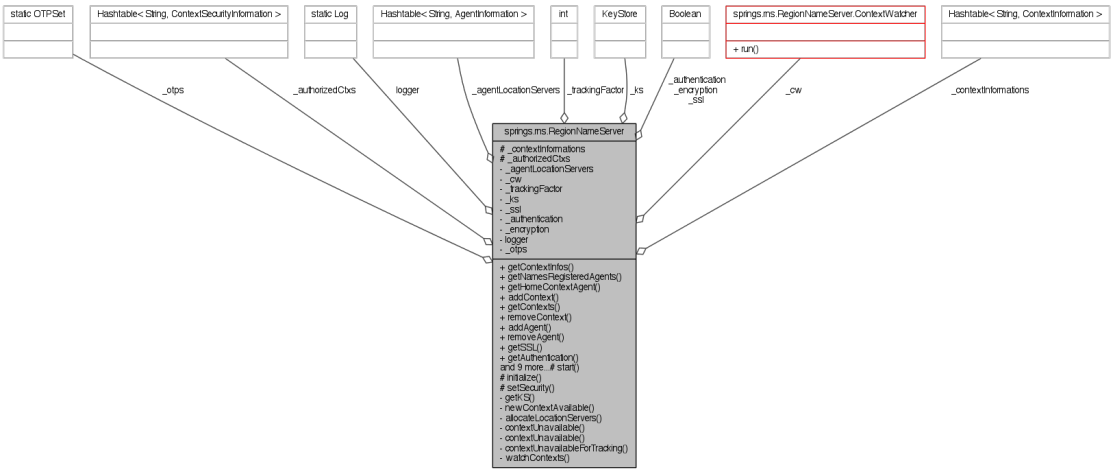



Figura 3.6: Clase RegionNameServer

- El contexto crea un objeto del tipo `ContextAuthentication` (mostrado en la Figura 3.7) utilizando el *challenge* recibido desde el RNS como uno de sus parámetros de entrada. El constructor del `ContextAuthentication` firma con una función criptográfica el *challenge* con la clave privada del contexto y lo almacena en una variable llamada *token*.

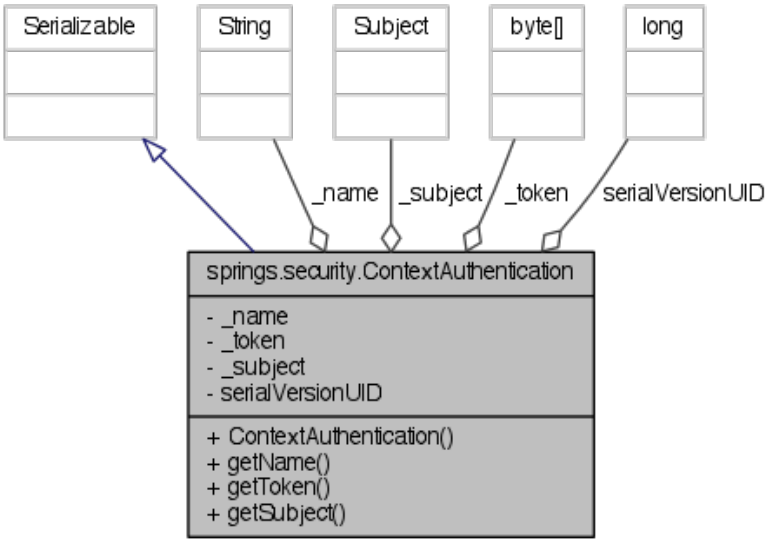



Figura 3.7: Clase ContextAuthentication

Este *token* es la información básica para autenticación del contexto frente al RNS.

- En cada subsiguiente operación que el contexto realiza contra el RNS o contra otro contexto, éste ha de enviar su objeto `ContextAuthentication` en la misma. El sistema que recibe la operación consulta al RNS si el objeto `ContextAuthentication` adjunto a la operación se encuentra autenticado en el sistema. Esto se realiza mediante el método `isContextAuthorized` del RNS.

  Dicho método recoge el objeto `ContextAuthentication` y compara que el *token* de dicho objeto ha sido calculado a partir del *challenge* ofrecido al contexto y su clave privada.

  Si esto es así, se permite devuelve un valor `true` y se permite la operación.

Este procedimiento evita la necesidad de utilizar claves secretas compartidas entre los sistemas, facilitando la gestión de la plataforma. De la misma forma mejora la seguridad de la misma, ya que el secreto compartido se crea nuevo cada vez que un contexto se añade a una región.

Como ya hemos indicado previamente, cuando un contexto crea un agente, éste hereda el objeto `ContextAuthentication` del contexto padre. Nadie más puede acceder a este objeto privado del agente. A partir de ese momento, el agente se autenticará y recibirá permisos según las credenciales de su contexto padre. De esta forma se evita que el agente pueda ser capaz de ejecutar diferentes acciones dependiendo del contexto en el que se está ejecutando. Se requiere que un agente se pueda autenticar de la misma forma independientemente de en qué contexto se encuentre.

Ya se ha introducido el concepto de permisos previamente al hablar del permiso `springs.security.authorizationLevel.PLATFORM_PERMISSION` que añade el RNS cuando incluye un nuevo objeto `ContextSecurityInformation`. Este permiso se refiere a la posibilidad de acceso por parte del elemento determinado a la plataforma. Es decir, es una especie de permiso especial utilizado para la autenticación en la plataforma. Cuando hemos hablado del uso del método `isContextAuthorized` para la autenticación, éste debe recibir el parámetro
`springs.security.authorizationLevel.PLATFORM_PERMISSION`.

De la misma forma, la clase `springs.security.authorizationLevel` ofrece otros permisos, como se puede ver en el diagrama de clases de la Figura 3.8

En dicha figura, observamos los siguientes permisos:

- **PLATFORM_PERMISSION**: Es el tipo especial de permiso que se necesita para la autenticación. Este permiso lo necesita todo elemento que quiera formar parte de la plataforma.
- **AGENT_PERMISSION**: Permite la creación y el borrado de agentes por parte del contexto. Un contexto solo puede borrar agentes que él mismo haya creado.

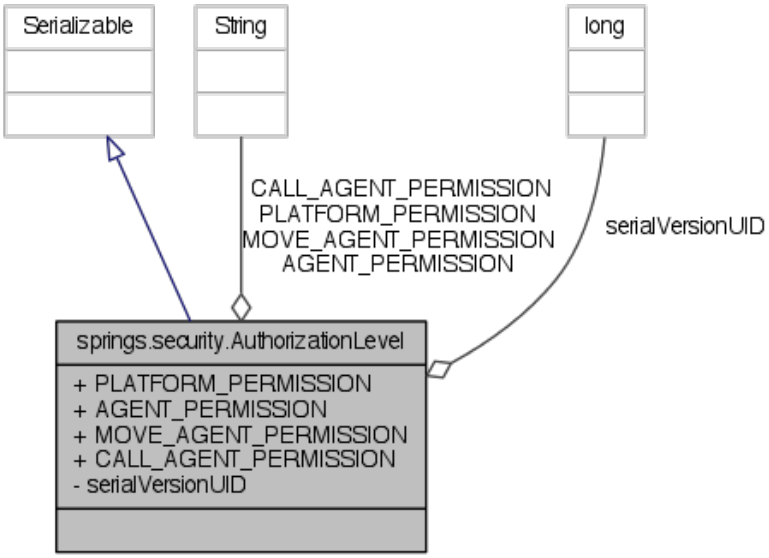



Figura 3.8: Clase AuthorizationLevel

- **MOVE_AGENT_PERMISSION**: Permite que los agentes creados por un contexto puedan viajar a otros contextos.
- **CALL_AGENT_PERMISSION**: Permite que los agentes creados por un contexto sean capaces de realizar llamadas a otros agentes.

En su arranque, el RNS lee su fichero de configuración. En éste se pueden especificar los permisos que se quieren otorgar a cada uno de los contextos. En una futura revisión de la plataforma, se podría permitir la modificación de la lista de permisos mientras el RNS esté en ejecución (cambios de configuración "en caliente").

Al igual que en el caso de la autenticación, cada vez que un elemento invoca una operación que requiere algún permiso especial (por ejemplo, la creación de un agente), el contexto realiza una llamada al RNS para validar si el elemento que realiza la petición tiene el nivel de permisos necesarios. Si no es así, no se podrá ejecutar la petición. En caso contrario, se procederá a realizar la operación.

### 3.3.2 Autenticación del RNS frente a los contextos

En la sección anterior hemos analizado cómo se autentican los contextos y los agentes ante el RNS y cómo se pueden autorizar cierto tipo de operaciones en la plataforma. También es necesario el caso contrario, poder autenticar de algún modo que el elemento gestor de la plataforma, es decir, el RNS, ante el resto de elementos de la plataforma.

Con esto se conseguirán evitar ataques por parte de otros elementos (por ejemplo, un agente "maligno") a un contexto invocándole métodos restringidos.

Para ello se ha ideado un protocolo sencillo basado en claves generadas y firmadas por el RNS que se pueden validar por parte de los contextos contactados y, de esta forma, permitir el acceso al contexto por parte del RNS. Estas claves son del tipo OTP (One Time Password), es decir, una vez que se han utilizado, se borran para que no puedan volver a ser usadas [51].

El funcionamiento de este mecanismo se describe en la Figura 3.9, dónde se detallan los métodos invocados, es el siguiente:

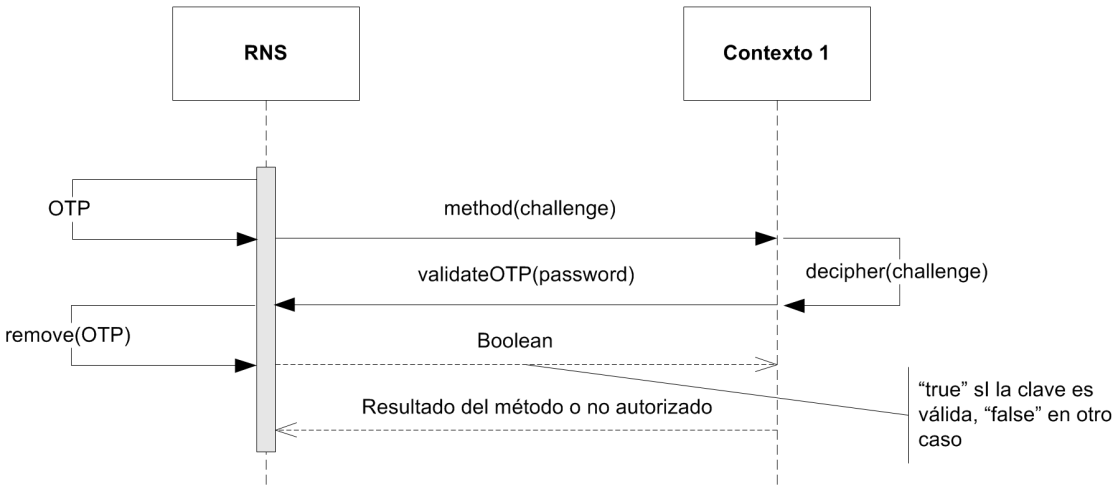



Figura 3.9: Autenticación y autorización del RNS

- Cuando el RNS necesita invocar un método de un contexto de su región crea un objeto de la clase `OTP` (mostrado en la Figura 3.10), que tiene como parámetros de entrada un identificador de operación (aleatorio), el nombre del contexto con el que se quiere contactar y su certificado. En la creación del `OTP` se genera una clave aleatoria (utilizando `SecureRandom`) y, ésta se cifra con la clave pública del contexto, creando un *challenge*. De esta forma nos aseguramos de que solo el contexto destino podrá tener acceso al *challenge*.

- El *challenge* junto con el identificador de operación se envía como parámetros al método del contexto invocado por el RNS.

- El contexto, recoge los parámetros y descifra, mediante su clave privada, el *challenge* enviado.

- El contexto invoca al método `verifyOTP` del RNS y, éste comprobará si la clave calculada es la correcta para el contexto y el identificador de operación. Si es así,

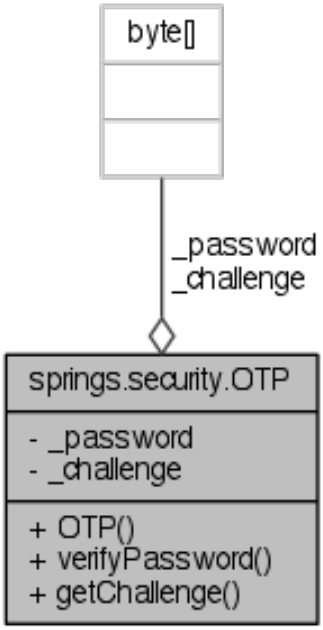



Figura 3.10: Clase OTP

se devolverá éxito, se borrará el OTP del RNS y el contexto seguirá ejecutando el código.

Si el OTP hubiera sido generado por otra entidad, al comprobar contra el RNS del sistema las credenciales éstas no existirían, con lo que el contexto no ejecutaría el código invocado.

Mediante este protocolo, podemos asegurar que una invocación remota por parte del RNS a un método de un contexto es realizada realmente por el RNS y no por cualquier otro elemento que estuviera intentando acceder a los métodos de un contexto sin autorización, protegiendo al contexto de posibles ataques por parte de terceros.

## 3.4 Autenticación a nivel de contexto

Para evitar ataques desde agentes a contextos, a parte de la autenticación y la autorización a nivel de plataforma que hemos explicado previamente, son necesarios otros mecanismos más a bajo nivel. El objetivo de estos mecanismos de seguridad es evitar el acceso no deseado a recursos de los contextos por parte de los agentes que viajan a los mismos.

Mediante este mecanismo, se puede controlar el acceso por parte de los agentes que viajan a un contexto de recursos tales como:

- Sistema de ficheros,
- conexiones de red,
- sistemas de bases de datos,
- propiedades del sistema.

Como hemos visto en las plataformas de agentes móviles estudiadas en el presente documento, existen métodos de seguridad inherentes a la JVM para defender una máquina virtual de código que se ejecute en la misma. El servicio Java que ofrece esta funcionalidad es el JAAS [52], cuyo funcionamiento está detallado en el Apéndice A.2.

En la implementación que se ha realizado para SPRINGS de JAAS, se ha optado por desarrollar dos mecanismos de login y dos contextos de seguridad diferentes, uno para gestionar los permisos que se quiere otorgar a un contexto en sí y otro para los permisos con los que permitiremos que se ejecuten los agentes dentro de un contexto:

- **Contexto de seguridad para contextos**. Se ha creado un contexto de seguridad específico para la seguridad dentro de la JVM de un contexto de SPRINGS. Para ello se han implementado las clases `ContextLogin, ContextLoginModule, ContextCallbackHandler y PasswordCallback`
  (Figuras 3.11, 3.12, 3.13 y 3.14).

  El *login* se realiza sencillamente dentro de un contexto utilizando una clave. Si el *login* tiene éxito, se crea un `Principal` de la clase `ContextPrincipal` (Figura 3.15).

  En la creación de un contexto, se realiza el *login* y el contexto comienza a ejecutarse con los permisos que hayan sido configurados. Lo habitual será que no exista ninguna limitación en los permisos con los que se permita ejecutar un contexto. Estos permisos se definen en el fichero de políticas de seguridad (normalmente `security.policy`) de la forma:

```
grant Principal springs.security.ContextPrincipal "context" {
    permission java.security.AllPermission "", "";
};
```

- **Contexto de seguridad para agentes**. El segundo contexto de seguridad que se ha creado es específico para agentes. Para ello se han desarrollado las clases `AgentLogin, AgentLoginModule y ContextAuthenticationCallback` (Figuras 3.16, 3.17 y 3.18).

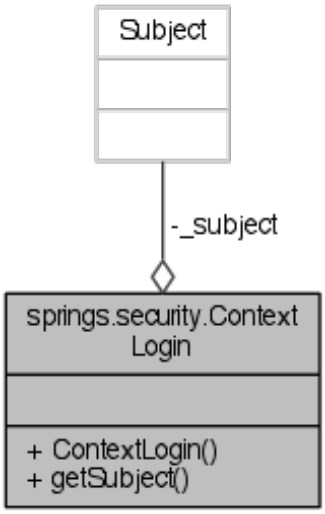



Figura 3.11: Clase ContextLogin

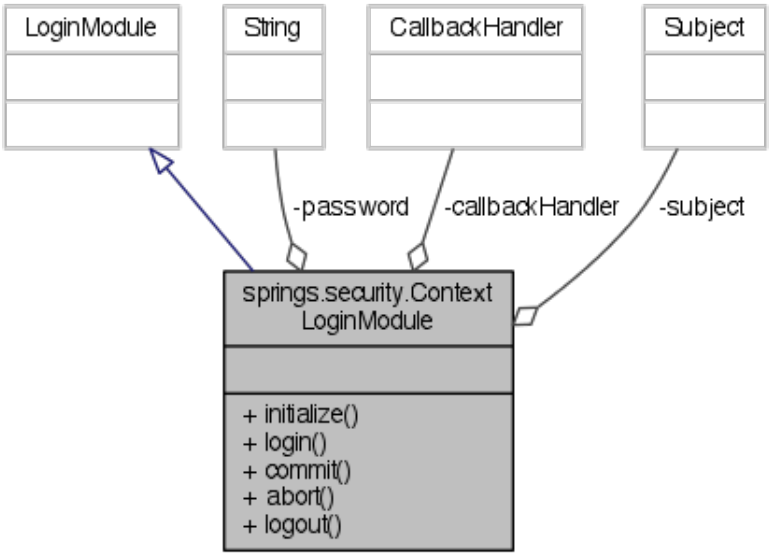



Figura 3.12: Clase ContextLoginModule

En el momento en el que un agente accede a un contexto y el contexto va a iniciar un nuevo *thread* con el código del agente, se intenta realizar un *login* con las credenciales del agente. Para ello se utiliza el objeto de la clase `ContextAuthentication` que, como hemos comentado previamente, identifica un agente utilizando las credenciales del contexto en el que fue creado. En el *login*, el

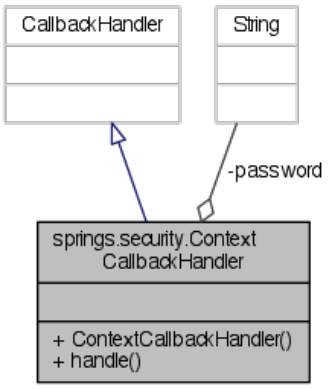



Figura 3.13: Clase ContextCallbackHandler

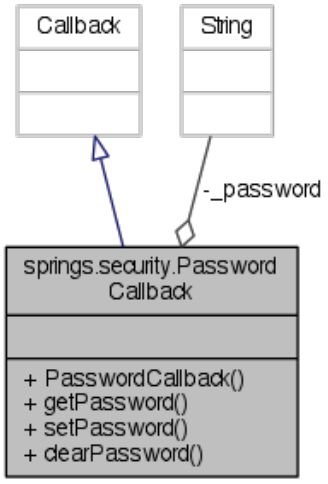



Figura 3.14: Clase PasswordCallback

contexto receptor accede al método de autenticación del RNS y, si tiene éxito, se permite el *login*, creando un `Principal` de la clase `AgentPrincipal` (Figura 3.19).

El contexto receptor ejecutará el código del agente bajo la identidad y con los permisos del contexto que creó el agente. Se permite configurar el nivel de autorización para cada identidad autorizada en la región, es decir, para cada contexto que se haya creado en la misma. El siguiente es un extracto de un fichero `security.policy` de ejemplo:

```
grant Principal springs.security.AgentPrincipal "Test" {
    permission java.util.PropertyPermission "user.home", "read";
};
```

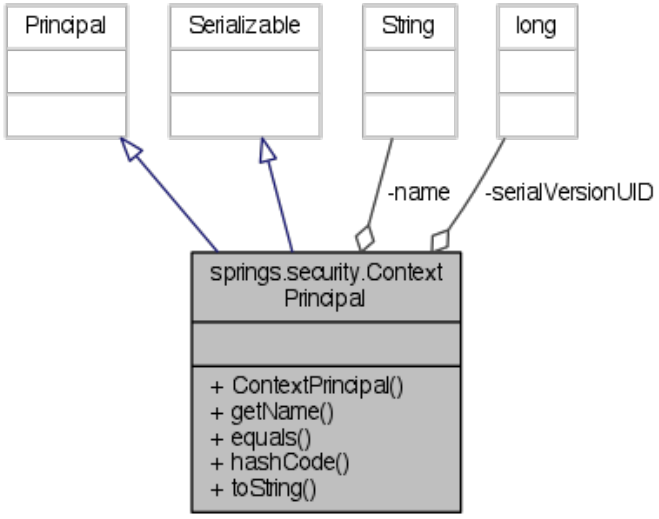



Figura 3.15: Clase ContextPrincipal

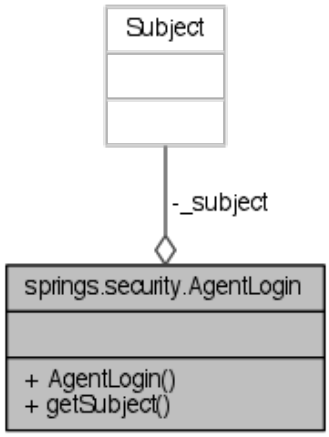



Figura 3.16: Clase AgentLogin

```
grant Principal springs.security.AgentPrincipal "Test2" {
    permission java.io.FilePermission "/etc/passwd", "read";
};
```

En el ejemplo anterior, vemos que a los agentes identificados como “Test” se les permite el acceso de lectura a la propiedad “user.home” y que los agentes identificados como “Test2” son capaces de leer el fichero `/etc/passwd` del contexto.

Cada *thread* de ejecución en el contexto para cada agente se ejecutará con los permisos definidos.

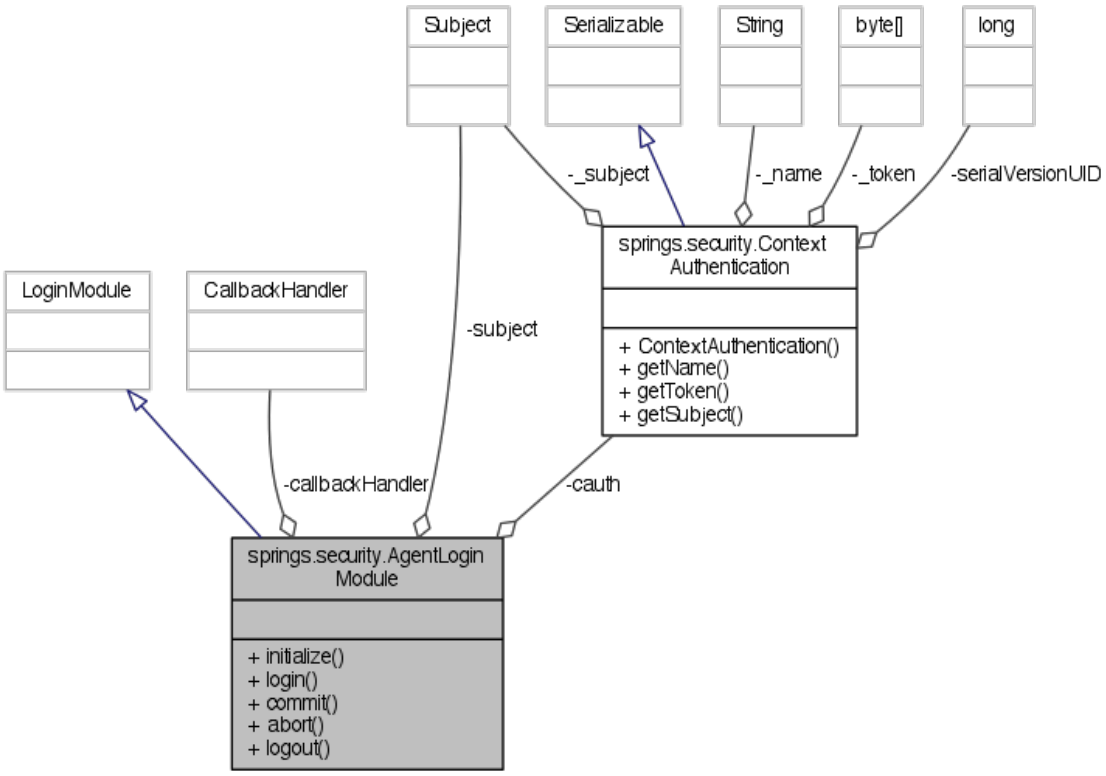



Figura 3.17: Clase AgentLoginModule

Gracias al uso de JAAS y a la protección ofrecida por el `SecurityManager` de Java, es posible disponer de un mecanismo de seguridad que permite limitar el impacto que pueden tener los agentes en un contexto, controlando el uso de recursos locales de un contexto que pueden realizar los agentes. Si bien, como ya se explicó en la Sección 2, es imposible defender un contexto ante ataques de denegación de servicio con una JVM estándar.

## 3.5 Cifrado y comprobación de integridad de datos

Para evitar que ciertos datos puedan ser leídos por otros elementos maliciosos y, además, para poder comprobar que un dato no ha sido alterado en el transcurso de un viaje de un agente, se ha dotado a la plataforma de un sistema de cifrado y comprobación de integridad de datos. Dicho sistema esta basado en el cifrado de clave asimétrica [53], haciendo uso del sistema de claves públicas y privadas que se ha desplegado en la plataforma.

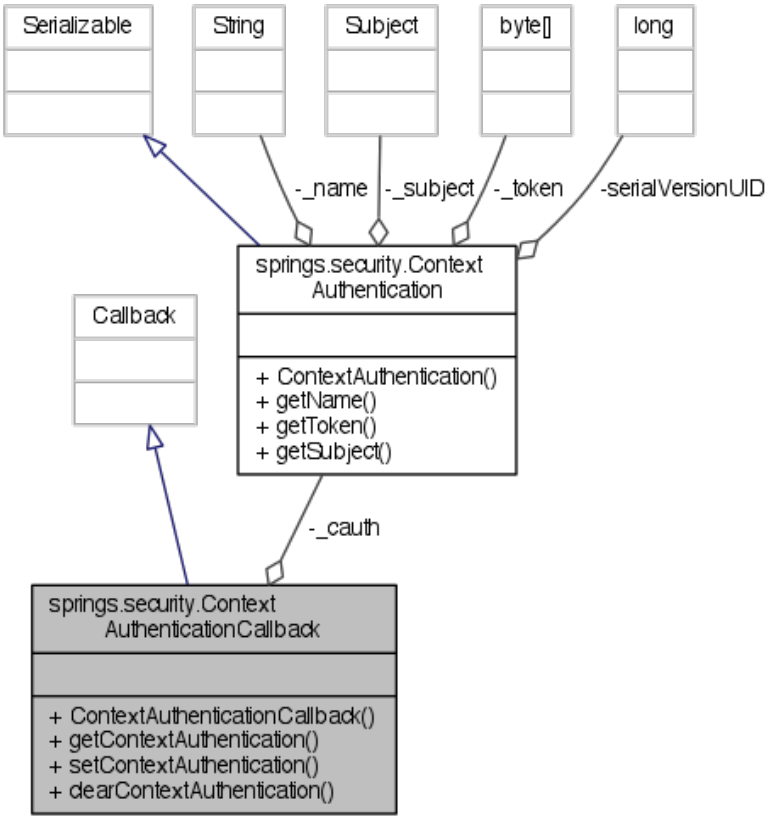



Figura 3.18: Clase ContextAuthenticationCallback

Un principio a tener en cuenta en este sistema, tal y como se ha comentado previamente, es que los agentes como tales no disponen de entidad criptográfica propia como un par de claves públicas/privadas. Con esto se consigue que, en caso de que de alguna forma algún elemento malicioso pudiera tener acceso a todos los datos de un agente, éste nunca poseería la clave privada para poderlos descifrar, con lo que se evitaría el descifrado directo de los datos. Siendo esto así, todo el sistema de cifrado se basa en las claves públicas y privadas de los contextos.

Los contextos ofrecen servicios de cifrado, descifrado y verificación de integridad de datos. Un contexto solo permite el servicio de descifrado a los agentes que hayan sido creados por él mismo. El RNS se encarga de funcionar como un directorio de claves públicas. Todos los elementos que requieran una clave pública de otro elemento de la plataforma han de consultarla mediante un método del RNS, como se aprecia en la Figura 3.20. Con este mecanismo evitamos que un elemento pudiera engañar al sistema ofreciendo una clave pública falsa con el fin de que el elemento original cifrara un dato que solo la entidad maliciosa pudiera descifrar. En el caso de que se quiera cifrar un

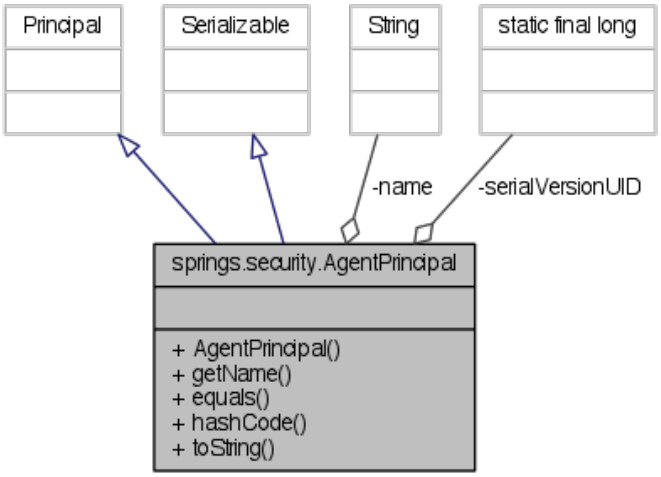



Figura 3.19: Clase AgentPrincipal

dato para un agente en especial, lo que se hace es obtener el contexto en el que se creó el agente y se cifra el dato con la clave pública de dicho contexto.

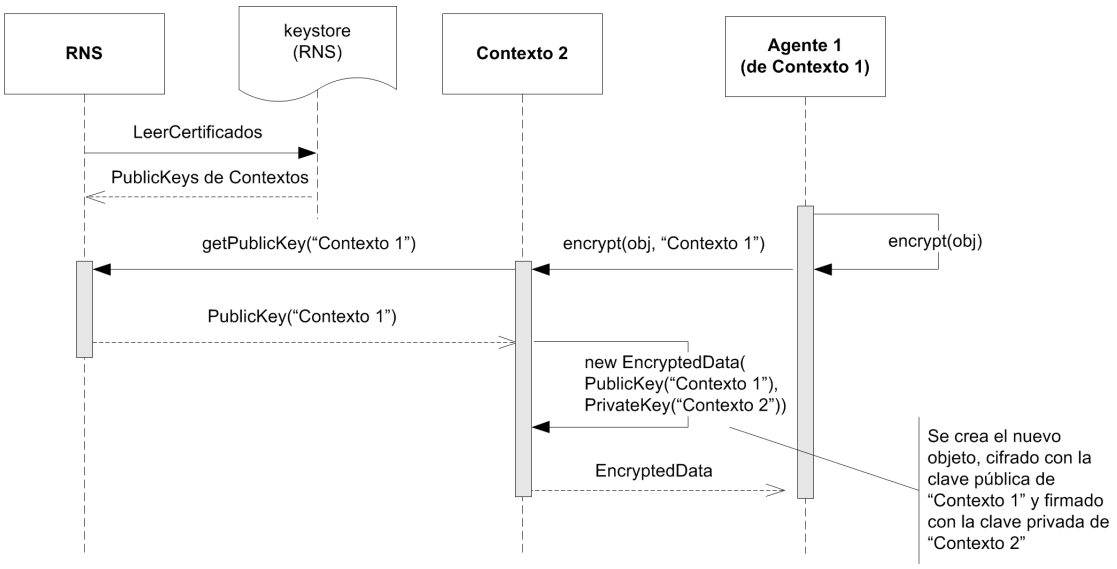



Figura 3.20: Cifrado y firma de datos

La contrapartida de que los agentes no dispongan de identidades criptográficas es que es necesario que un agente viaje a su contexto origen, donde exclusivamente podrá descifrar los datos. Es decir, ni el propio agente será capaz de descifrar un dato a no ser que viaje al contexto en el que fue creado. Este mecanismo asegura que un dato cifrado, aun siendo robado, solo pueda ser descifrado por un agente que haya sido creado

en el mismo contexto donde se creó el primer agente, es decir, que pertenezca al mismo contexto.

La firma y la validación de los datos se basan en la misma estructura expuesta de claves asimétricas. En el momento en que se cifra un dato, también se genera una estructura de firma (mediante el algoritmo "SHA1 con RSA" [54]). Para firmar un dato se utiliza la clave privada del contexto en el que se está cifrando el dato. Para verificar que los datos no han sido alterados es necesario descifrar los mismos y ejecutar un método de comprobación de integridad, que validará que los datos son los originales y que no han sido alterados.

En todo momento el agente puede obtener en qué entidad se ha firmado un dato. La estructura de datos cifrados se encuentra definida en la clase `EncryptedData` (detallada en la Figura 3.21).

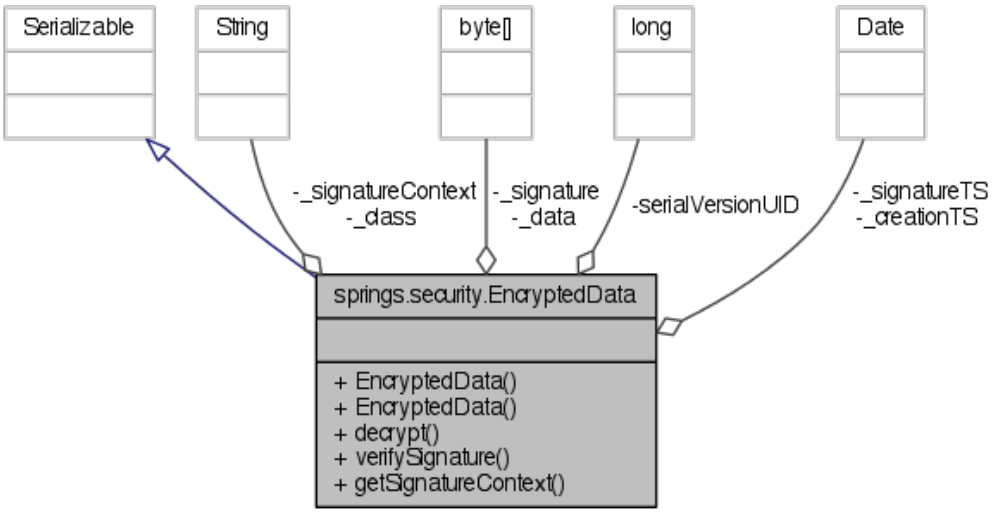



Figura 3.21: Clase EncryptedData

Desde el punto de vista del usuario o programador de agentes, el uso del sistema es muy sencillo. El procedimiento se halla descrito en la Figura 3.20. Para cifrar un dato, tan solo hace falta invocar el método `encrypt(Object obj)` para cifrar un dato que solo podrá ser descifrado en el contexto en el que se creó el agente o `encrypt(Object obj, String agentName)` si se quiere cifrar un dato para ser enviado a otro agente (siendo `obj` el objeto a cifrar y `agentName` el nombre del agente destino que lo va a poder descifrar). El resultado de dicha operación será un objeto de la clase `EncryptedData`.

Para descifrar el objeto, como se explica en la Figura 3.22, es necesario que el agente viaje al contexto en el que fue creado para que, una vez en él, ejecute el método `decrypt(EncryptedData data)` que devuelve el objeto descifrado.

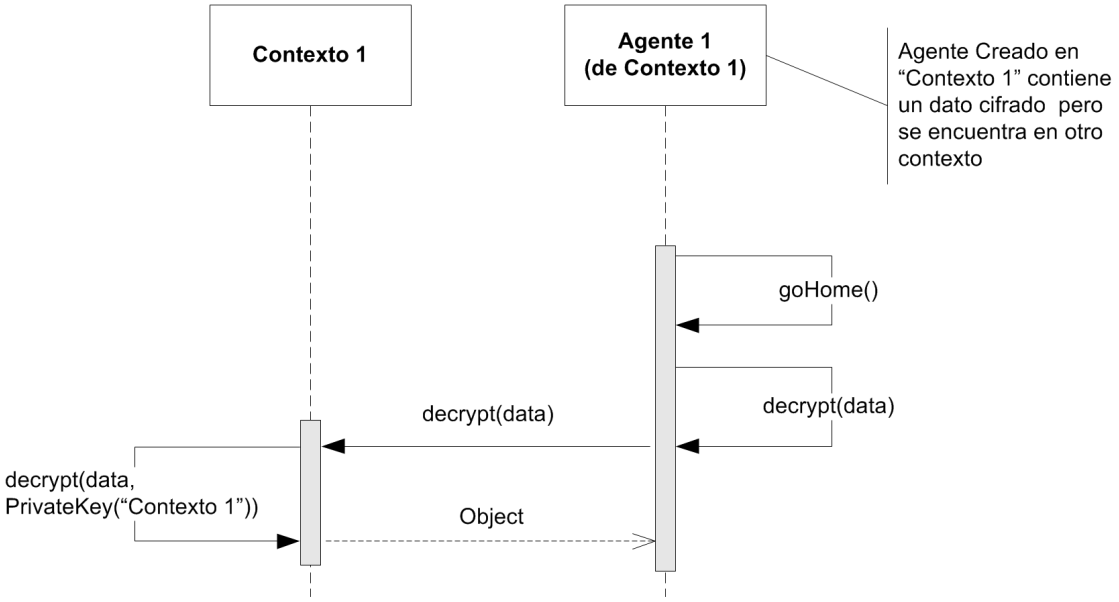



Figura 3.22: Descifrado de datos

Si, una vez descifrado el objeto, se quiere validar que no ha sido modificado en tránsito es necesario invocar el método `verifySignature(Object obj, EncryptedData encryptedObject)` que realizará dicha verificación, como se describe en la Figura 3.23.

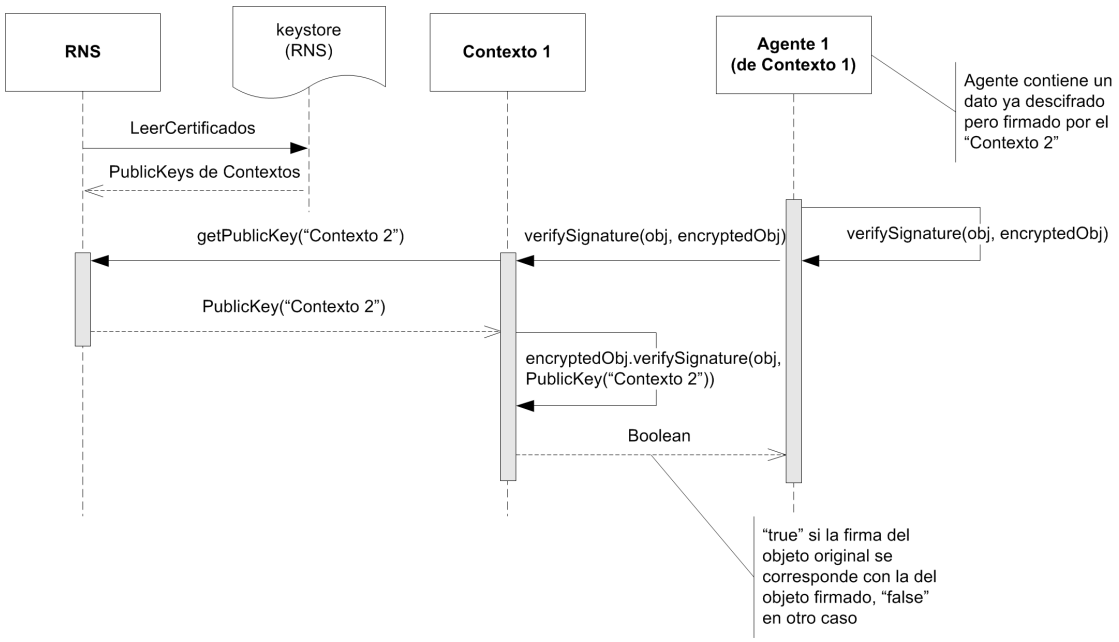



Figura 3.23: Comprobación firma de datos

Con estos mecanismos podemos proteger la confidencialidad y la integridad de los datos que lo precisen. Su uso es voluntario por parte del usuario, añadiendo complejidad a la plataforma solo si el usuario considera conveniente utilizarlos (posiblemente porque sepa que la plataforma sobre la que va a ejecutar sus agentes puede ser fácilmente expuesta a ataques). Estos mecanismos se integran perfectamente con el resto de mecanismos y prácticas habituales de la plataforma, con lo que la misma se puede seguir utilizando de la misma forma por parte de los usuarios.

Debido a que las claves públicas se deben obtener desde el RNS para utilizar de una forma fiable los mecanismos de cifrado y comprobación de integridad de datos, es necesario activar esta característica globalmente para toda la región, solo pudiendo utilizarse si ésta ha sido activada en el RNS.

## 3.6 Comparativa con otras plataformas de agentes móviles estudiadas

A modo de resumen, nos gustaría exponer en la Tabla 3.1 cómo se comporta SPRINGS en comparación con las plataformas estudiadas en la Sección 2.2 según las principales medidas de seguridad esgrimidas en el Capítulo 2, y más detalladamente, según la Tabla 2.1.

| | **SeMoA** | **Jade** | **Aglets** | **Tryllian** | **Voyager** | **SPRINGS** |
|---|---|---|---|---|---|---|
| **Autenticación** | Sí | Jade-S | Sí | A nivel de JAR | Sí, programable | Sí, bidireccional |
| **Confidencialidad** | | | | | | |
| **- Agentes** | Sí | No | No | No | No | No |
| **- Mensajes** | Sí | Jade-S | A nivel de dominio | No | No | Sí |
| **- Comunicaciones** | Sí | IMTPoverSSL | No | Sí | Sí | Sí |
| **Integridad** | | | | | | |
| **- Plataforma** | Sí | Sí | Sí | Sí | Sí | Sí |
| **- Agente** | Sí | Sí | Sí | Sí | Sí | Sí |
| **- Comunicaciones** | Sí | IMTPoverSSL | No | Sí | Sí | Sí |
| **Responsabilidad** | Sí | No | No | No | No | No |
| **Disponibilidad** | JVM | JVM | JVM | JVM | JVM | JVM |

Tabla 3.1: Comparativa de medidas de seguridad de SPRINGS y de las plataformas estudiadas

En la tabla podemos observar como SeMoA sigue siendo la plataforma de agentes móviles más segura. Aunque, como se ha explicado previamente, esto es debido a que es una plataforma diseñada desde un principio desde el punto de vista de la seguridad. Por ejemplo, un agente en SeMoA es un fichero JAR en el que están contenidas tanto las clases de las que se compone el agente como su estado, sus certificados y sus ficheros de comprobación de estado con las firmas digitales de cada uno de los elementos. El movimiento de un agente implica la copia de este fichero JAR, su paso por cada uno de los filtros de seguridad habilitados, donde se comprueban que tanto el código como los datos del agente no han sido modificados y que cumplen las medidas de seguridad

requeridas, y su posterior carga y ejecución en el servidor. Sin embargo en SPRINGS el movimiento de un agente se realiza mediante mecanismos propios de RMI, en los que se le pasa a un método de un contexto un objeto que contiene el estado del agente y se cargan remotamente las clases que sean necesarias siguiendo el mecanismo estándar de carga de clases de RMI.

Vemos cómo dentro del resto de plataformas de propósito general, SPRINGS destaca por ofrecer medidas de seguridad para solucionar gran parte de los ataques básicos a una plataforma de agentes móviles, además de alguna medida innovadora que la dotan de defensas ante ataques que el resto de plataformas no pueden evitar como, por ejemplo, el uso de la autenticación por parte del RNS para evitar que se pueda acceder a los métodos de un contexto.

Aun así, como ya se ha comentado en varias ocasiones en el presente documento, todas las plataformas se ejecutan sobre JVM no modificadas, con lo que les es imposible defenderse ante ataques de denegación de servicio.

# Capítulo 4

# Pruebas sobre la plataforma SPRINGS

En el presente capítulo vamos a pasar a describir las pruebas que hemos realizado a la plataforma SPRINGS tras la implementación de los mecanismos de seguridad descritos en el Capítulo 3.

En primer lugar describiremos las pruebas funcionales sobre cada uno de los mecanismos implementados, probando que realmente funcionan y simulando la protección que ofrecen ante posibles ataques a la plataforma.

Posteriormente pasaremos a describir las pruebas de carga realizadas, basadas en las pruebas que se realizaron para evaluar el rendimiento de la plataforma [12]. Primero se realizarán dichas pruebas sin las medidas de seguridad desarrolladas activas y después se irán activando cada una de las medidas para estudiar cómo afectan al rendimiento de la plataforma.

Por último ofrecemos un estudio de escalabilidad de la plataforma con el nivel máximo de seguridad activo, observando el comportamiento de la misma a medida que va gestionando más agentes.

Como complemento a esta sección se recomienda consultar tanto el Apéndice A, en el que se describe el entorno tecnológico sobre el que se han desarrollado las pruebas como el Apéndice C, que detalla cómo se pueden configurar los diferentes mecanismos de seguridad en la plataforma.

## 4.1 Pruebas funcionales

Como se ha presentado previamente, en esta sección vamos a describir las pruebas relacionadas con los mecanismos de seguridad implementados viendo cómo estos son capaces de ofrecer la protección necesaria ante diferentes tipos de ataques posibles que se puedan realizar.

Según la clasificación de ataques, las pruebas que se han realizado contienen cómo la plataforma se puede defender ante:

- Ataques de acceso a la plataforma por parte de elementos no autorizados, detallados en los Apéndices B.1, B.2 y B.4.
- Ataques de interceptación de comunicaciones, tanto a nivel de contextos como a nivel de agentes. Estos ataques están detallados en los Apéndices B.3, B.13 y B.14.
- Ataques de contextos contra la plataforma, descritos en los Apéndices B.5, B.6 y B.7.
- Ataques desde contextos o agentes a otros contextos, definidos en los Apéndices B.8 y B.10.
- Ataques de denegación de servicio. Por ejemplo, el realizado en el Apéndice B.15.

Más específicamente, se han realizado con éxito las siguientes pruebas funcionales en la plataforma, detalladas en el Apéndice B:

- Acceso a la plataforma de un contexto que no tiene activado SSL.
- Acceso a la plataforma de un contexto que usa un certificado no válido.
- Acceso y/o modificación de comunicaciones entre el RNS y los contextos o entre contextos.
- Acceso a la plataforma de un contexto cuyo certificado no está aceptado por el RNS.
- Creación de un agente por parte de un contexto que no tiene permisos para ello.
- Movimiento de un agente por parte de un contexto que no tiene permisos para ello.
- Agente realizando llamadas creado en un contexto que no tiene permisos para ello.
- Contexto atacando otro contexto simulando ser el RNS.
- Uso de autenticación de contexto con la autenticación de plataforma desactivada.

- Acceso a recursos no permitidos por la autorización a nivel de contexto.
- Uso de cifrado cuando está desactivado en la plataforma.
- Uso de cifrado y verificación de integridad de datos.
- Intento de descifrado de datos en un contexto incorrecto.
- Simulación de robo de dato cifrado e intento de descifrarlo.
- Ataque de denegación de servicio mediante el uso masivo de memoria de un contexto.

Todas estas pruebas han acabado ofreciendo los resultados esperados. Se puede consultar el detalle de las mismas en el Apéndice B.

## 4.2 Resultados Experimentales

En esta sección vamos a proceder al análisis de las pruebas de carga y de escalabilidad realizadas a la plataforma. Dichas pruebas están basadas en las pruebas que se realizaron para evaluar el rendimiento de la plataforma SPRINGS en su concepción [12].

### 4.2.1 Entorno de pruebas

El entorno utilizado en las pruebas de carga es el siguiente:

- Un contexto llamado "Test" en el que se crean agentes, estos esperan un periodo de "calentamiento" y una vez acabado dicho periodo los agentes entran en un bucle de llamadas y movimientos. En el caso estudiado, cada agente realiza 50 llamadas a un agente predesignado como su "pareja" y tras cada llamada se mueve aleatoriamente a un contexto de la región.
- Tres contextos que aceptan a los agentes que han sido creados en el contexto "Test" y les ofrecen sus servicios. Dichos contextos son "c1", "c2" y "c3".
- Un RNS que gestiona toda la región.

Se puede encontrar más detalle de las acciones que realizan los contextos y los agentes de las pruebas de carga en el Apéndice A.4.4.

### 4.2.2 Escenarios funcionales

Se han probado los siguientes escenarios para realizar la comparativa:

- Un escenario en el que no está activa ninguna medida de seguridad desarrollada.
- Un escenario en el que solo está activa la capa de seguridad externa en el RNS y, por lo tanto, en el resto de contextos. Todas las comunicaciones RMI de la región estarán cifradas mediante SSL.
- Un escenario en el que solo se encuentra activa la autorización y la autenticación a nivel de plataforma. Se le han otorgado permisos de creación de agentes, movimiento de los mismos y llamada a otros agentes al contexto "Test" en el RNS donde todos los contextos están autorizados.
- Un escenario en el que se encuentra activa la autorización y la autenticación a nivel de plataforma como en el caso anterior y, además, se encuentran activados los mecanismos de autenticación y autorización a nivel de contexto. La configuración de dichos mecanismos permite el acceso completo a todos los agentes creados en el contexto "Test".
- Un escenario donde se han habilitado todas las medidas de seguridad desarrolladas, según se ha ido explicando en los escenarios anteriores.

En las pruebas de carga se han creado 1500 agentes que deben realizar 50 llamadas a su agente "pareja" y 50 movimientos a otros contextos antes de terminar su ejecución en el contexto "Test".

Para las pruebas de escalabilidad se ha utilizado el experimento del escenario que incluye la seguridad máxima, pero incrementando el número de agentes de 100 en 100 hasta 1500.

Cabe destacar que estas pruebas ponen al límite a una plataforma de agentes móviles por dos razones principales. En primer lugar, por el número de agentes que forman parte de las pruebas y, en segundo lugar, por la dificultad que tienen los agentes para realizar llamadas a su agente "pareja" ya que éste se está moviendo continuamente, lo que hace que sea difícil invocar el método de un agente justo cuando está en un contexto. El éxito de dichas operaciones se basa en el reintento de las mismas gracias a los mecanismos que ofrece la plataforma SPRINGS.

En cada escenario se obtienen las siguientes métricas:

- Tiempo de ejecución de la prueba. Este tiempo, medido en segundos, nos indica la velocidad con la que la prueba total se ha realizado, desde que empieza a viajar el primer agente hasta que acaba su ejecución el último.

- Tiempo de estancia. Mide, en en segundos, la media aritmética del tiempo que cada uno de los agentes ha debido pasar en un determinado contexto. Este indicador nos da una medida de la carga que ha tenido que soportar un contexto, ya que cuanta más carga, más tiempo deben estar los agentes en un contexto para poderse ejecutar. Este tiempo también incluye el tiempo de llamada al agente "pareja".

- Tiempo de llamada. Este indicador mide en segundos cuánto tiempo tarda un agente en invocar con éxito a su agente "pareja", incluyendo todos los reintentos que sean necesarios.

Se ha intentado obtener la métrica relacionada con el tiempo que tarda un agente en realizar un movimiento. Para medir dicho tiempo, el código desarrollado implica:

- Antes de invocar un movimiento del agente, se obtiene el tiempo del sistema y se almacena en un objeto del agente.

- Se procede al movimiento del agente.

- En el método `postArrival()` del agente, se obtiene el tiempo del sistema (del nuevo sistema) y se compara con el tiempo almacenado.

En esta última comparación hemos obtenido tiempos negativos. La conclusión a la que hemos llegado es que, posiblemente, la gestión del tiempo de sistema que realiza el entorno virtualizado utilizado para las pruebas no funciona con la precisión que necesitamos para realizar nuestras mediciones. Hemos probado que, tras varias sincronizaciones sucesivas mediante un servidor NTP (Network Time Protocol), siempre obtenemos pequeñas correcciones del reloj interno de la máquina virtual del orden de milisegundos, que es lo que tardan en realizarse la mayoría de los viajes.

Por esta razón hemos decidido omitir del estudio actual las métricas de tiempo de movimiento, aplazándolas para cuando se pueda disponer de otro entorno de pruebas.

El entorno *hardware* sobre el que se han realizado las pruebas de carga y escalabilidad es el descrito en el Apéndice A.4.3.

### 4.2.3 Resultados y conclusiones de las pruebas de carga

A continuación se muestran varias figuras con los resultados obtenidos tras la ejecución de las pruebas de carga. Figura 4.1 expone los tiempos de ejecución de las pruebas, la Figura 4.2 muestra el tiempo de estancia medio en cada contexto por parte de cada agente y la Figura 4.3 indica el tiempo medio de llamada al agente "pareja" por parte de cada agente.

Analizando la Figura 4.1 observamos un detalle inesperado. Parece razonable que activando la capa de seguridad externa el tiempo de ejecución de la prueba se vea afectado, ya que todas las comunicaciones deben ser precedidas por el *handshake* de SSL y cifradas, con lo que se incrementa el tiempo al realizar todas las llamadas. Lo que no era esperado es que activando los mecanismos de autorización y autenticación a nivel de plataforma obtengamos un mejor rendimiento de la misma. Al activar dichos mecanismos, cada operación que requiere autorización (como puede ser la creación, el movimiento o una llamada a otro agente) necesita realizar una consulta previa al RNS para obtener la autorización, introduciendo un intervalo de tiempo de espera del agente dentro de un contexto, como se puede observar en el incremento del tiempo de estancia en los contextos de la Figura 4.2. Este tiempo de espera tiene el efecto lateral de incrementar la probabilidad de acceder a un agente en un contexto por parte de otro agente, como se vio en las pruebas expuestas en el artículo [12], con lo que el tiempo total de la prueba desciende al haber más éxito en la invocación al resto de agentes.

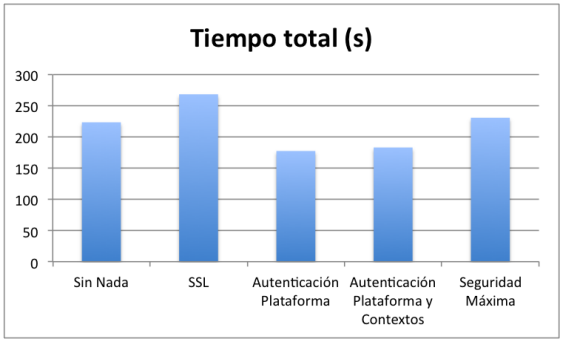



Figura 4.1: Pruebas de carga: Tiempo total de ejecución

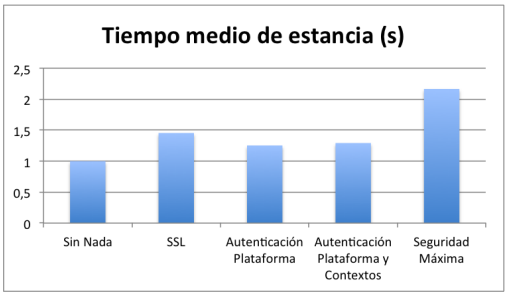



Figura 4.2: Pruebas de carga: Tiempo medio de estancia

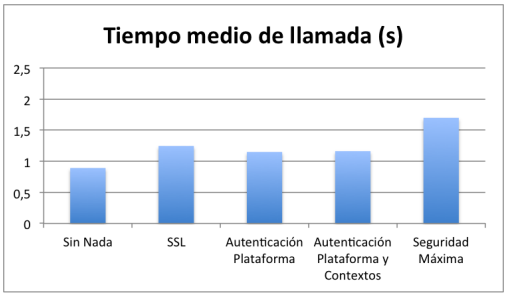



Figura 4.3: Pruebas de carga: Tiempo medio de invocación a agente

Por lo demás, los resultados son razonables. En los casos de uso de la capa externa de seguridad obtenemos un pequeño incremento de los tiempos de finalización de las pruebas, al igual que si utilizamos la autenticación y autorización a nivel de contextos, pero se considera que el impacto en el rendimiento de la plataforma es mínimo.

### 4.2.4 Resultados y conclusiones de las pruebas de escalabilidad

A continuación se muestran varias figuras con los resultados obtenidos tras la ejecución de las pruebas de carga. La Figura 4.4 expone los tiempos de ejecución de las pruebas, la Figura 4.5 muestra el tiempo de estancia medio en cada contexto por parte de cada agente y la Figura 4.6 indica el tiempo medio de llamada al agente "pareja" por parte de cada agente.

Viendo todas las figuras, se puede observar cómo la escalabilidad de la plataforma es prácticamente lineal, incrementándose todos los tiempos medidos linealmente en relación con el número de agentes existentes en la plataforma.

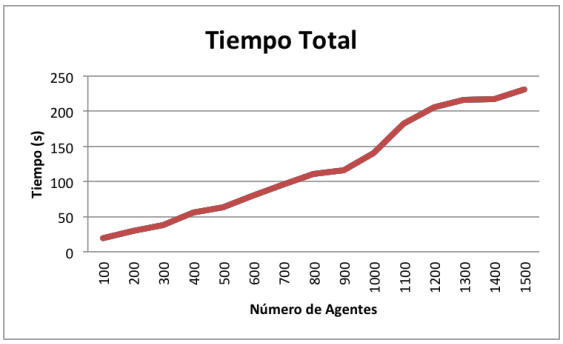



Figura 4.4: Pruebas de escalabilidad: Tiempo total de ejecución

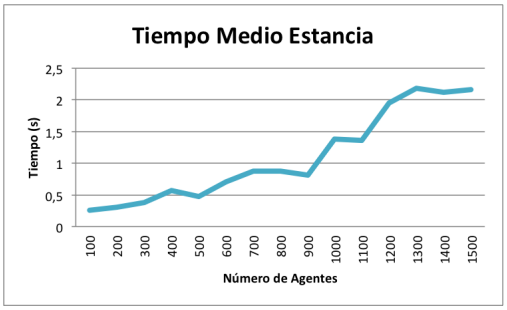



Figura 4.5: Pruebas de escalabilidad: Tiempo medio de estancia

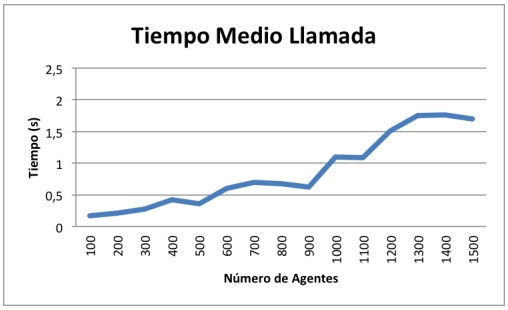



Figura 4.6: Pruebas de escalabilidad: Tiempo medio de invocación a agente

A este hecho ayuda el dimensionamiento del servidor que contenga el RNS, debido a que éste se puede convertir en un cuello de botella al tener que gestionar todas las peticiones de autorización y autenticación de la plataforma. Es importante que el RNS disponga de los recursos adecuados para permitir un buen funcionamiento de toda la plataforma.

# Capítulo 5

# Conclusiones y trabajo futuro

Este capítulo resume los logros y las conclusiones obtenidas tras la realización del presente Proyecto Fin de Carrera. Para ello, en primer lugar analizaremos los objetivos cumplidos y los resultados del proyecto, sugiriendo posibles líneas futuras de trabajo. También realizaremos una valoración personal sobre el desarrollo del proyecto.

## 5.1 Objetivos y resultados del proyecto

El presente proyecto ha ido completando cada uno de los objetivos establecidos en su propuesta. Más específicamente, hemos llevado a cabo con éxito las siguientes tareas:

- Se ha realizado un estudio de las medidas de seguridad propuestas para plataformas de agentes móviles, haciendo especial énfasis en los posibles ataques que se podrían llevar a cabo y diferentes formas de poderlos evitar.
- Hemos estudiado la forma de implementar diferentes mecanismos de seguridad por parte de las plataformas de agentes móviles más importantes de la actualidad. En el estudio, hemos observado cómo debe existir un equilibrio entre las medidas de seguridad implementadas por una plataforma y su usabilidad y rendimiento, ya que no siempre la plataforma con las mejores medidas de seguridad es la que más aceptación tiene.
- Se han definido e implementado en SPRINGS las medidas de seguridad más importantes para prevenir los ataques más comunes a una plataforma de agentes móviles. Hemos dotado a la plataforma de la capacidad de cifrar sus comunicaciones, de obligar a la autenticación de cada uno de sus elementos en la plataforma, permitiendo o denegando la creación de agentes y la capacidad de movimiento y de realización de llamadas de los mismos, de evitar el acceso

a recursos no autorizados de los contextos por parte de los agentes y de la capacidad de cifrar y comprobar la integridad de los datos a los agentes. En dicha implementación se ha buscado ofrecer un funcionamiento sencillo para facilitar el uso de dichas medidas de seguridad. También se ofrece la posibilidad de poder activar o desactivar cada una de las medidas a voluntad del administrador de la plataforma, con el objetivo de minimizar el impacto en el rendimiento de la plataforma, si éste fuera primordial.

- Hemos realizado pruebas funcionales simulando diferentes tipos de ataques demostrando la efectividad de las diferentes medidas de seguridad desarrolladas. De la misma forma hemos realizado pruebas de ataques que, con la tecnología estándar actual, no se pueden defender, pudiendo causar daños en la plataforma.

- Con el objetivo de estudiar el impacto en el rendimiento de la plataforma de cada una de las medidas implementadas, hemos realizado unas pruebas de carga activando cada una de las medidas de seguridad basadas en la metodología de anteriores pruebas de carga realizadas en la plataforma [12]. En dichas pruebas se ha visto que, efectivamente, la plataforma puede sufrir un impacto en rendimiento al utilizar dichas medidas pero que dicho impacto puede y debe ser asumido si se quiere utilizar la plataforma en un entorno real de producción donde los ataques sean posibles.

## 5.2 Trabajo futuro

Aun habiendo cumplido todos los requisitos de la propuesta del presente Proyecto Fin de Carrera, se han detectado una serie de mejoras que podrían incrementar la seguridad de la plataforma SPRINGS. Principalmente se han detectado tres principales áreas de mejora, que son:

- Integración con plataformas de gestión de identidades. En vez de utilizar el RNS como una entidad de autenticación, en entornos masivos abiertos a Internet se podría delegar dicha funcionalidad a otro tipo de servicios de gestión de identidades, como pueden ser los servicios basados en OpenID [48]

- Existen algunos mecanismos de seguridad que no se han podido implementar en el trascurso del presente proyecto. Algunos de estos mecanismos, como por ejemplo la gestión ante algún tipo de ataque de denegación de servicio, podrían intentar mitigarse. Si bien no todos los ataques de este tipo lo pueden ser, debido a las limitaciones en la actual máquina virtual de Java, como se explica en la Sección A.2.

- Mejoras en entornos de alta movilidad. Como hemos visto, la arquitectura de SPRINGS está centralizada a nivel del RNS. Sería recomendable ofrecer una

arquitectura alternativa que evitara la total dependencia (o la mitigara de alguna forma) de dicho elemento ofreciendo toda la funcionalidad desarrollada.

- El SID tiene previsto ampliar el estudio experimental inicial realizado en este trabajo.

Las extensiones de seguridad integradas en SPRINGS permitirán al grupo SID de la Universidad de Zaragoza continuar el desarrollo de la plataforma con soporte para proteger a los sistemas de agentes móviles de posibles ataques. El grupo SID tiene intención de continuar el estudio y extender la arquitectura diseñando e implementando otras medidas de seguridad que puedan resultar de interés. Posteriormente, se cree que se estará en condiciones de poder preparar un artículo de investigación conjunto que incluya parte del trabajo desarrollado en este proyecto.

Además, a través de la página web de SPRINGS [1] mantenida por el grupo SID, se permitirá el acceso a las extensiones de seguridad desarrolladas.

## 5.3 Valoración personal y problemas encontrados

El desarrollo del presente Proyecto Fin de Carrera se podría calificar como un reto. Se han producido una serie de problemas, principalmente de índole personal, que han hecho que tanto el tiempo estimado para el desarrollo del mismo, como el esfuerzo dedicado no haya sido el real. Dichos problemas se podrían resumir en:

- Compatibilización del desarrollo del proyecto con contrato laboral a tiempo parcial. En un principio, el autor pensó que sería posible compatibilizar dichas tareas pero, debido a una gran exigencia laboral, las tareas se fueron posponiendo y solo gracias a la paciencia del director del proyecto y a la insistencia y a un esfuerzo considerable, se ha podido ofrecer el presente resultado.
- Trabajo en remoto. El autor vive fuera de Zaragoza con la dificultad que eso añade a aspectos como la comunicación con el director del proyecto, el acceso a recursos de la Universidad de Zaragoza, etc.
- Escasa experiencia en desarrollo de software Java. Tras más de diez años de experiencia laboral, el autor nunca había tenido la necesidad de realizar desarrollos en el lenguaje de programación Java, cuyo uso y aprendizaje ha supuesto una dificultad añadida a la ejecución del proyecto.
- Desconocimiento de la tecnología de agentes móviles, ya que es un paradigma no implantado a nivel empresarial masivamente.

[1] http://osiris.cps.unizar.es/SPRINGS

- Trabajo sobre una plataforma ya existente. Este hecho tiene sus ventajas e inconvenientes. Como ventajas encontramos que el trabajo básico funcional de la plataforma ya se encuentra realizado. Sin embargo, esto puede haber requerido decisiones de diseño que no faciliten la implementación de otro tipo de características. Como desventajas hemos encontrado la complejidad tecnológica de la plataforma ya desarrollada y la falta de documentación sobre la misma.

La superación de todos estos problemas hace que la consecución del reto establecido sea incluso más satisfactoria que en condiciones normales.

La contrapartida se encuentra en que la planificación original del proyecto no se ha podido cumplir. Incluso, se ha tenido un efecto adverso al tener que parar completamente el desarrollo del proyecto y retomarlo pasado un tiempo varias veces, obligando a volver a ejecutar tareas que ya se habían completado previamente. Sin embargo, los últimos meses del proyecto sí que han tenido una dedicación total y esto ha ayudado definitivamente a la consecución de los objetivos del mismo.

De la misma forma, creemos que la experiencia laboral previa del autor en entornos de producción ha sido muy positiva en la implementación de ciertos "añadidos" a la plataforma, como puede ser el nuevo sistema de configuración o el mecanismo homogéneo de *logging*, así como el uso de herramientas de construcción de código y de gestión de versiones.

A nivel personal, el autor ha aprovechado para adquirir conocimientos en otros paradigmas "alternativos" de sistemas distribuidos como es el modelo de agentes móviles, quizá aplicables a trabajos futuros, así como en el desarrollo de *software*, especialmente en el lenguaje de programación Java.

De la misma forma, ha sido una buena oportunidad para volver al mundo universitario y analizar cómo funciona el mismo a día de hoy, pensando en futuras relaciones y cooperaciones.

# Apéndices

# Apéndice A

# Entorno Tecnológico

En este apéndice queremos reseñar las tecnologías utilizadas en la elaboración del presente Proyecto Fin de Carrera. Vamos a describir los conceptos generales de seguridad en informática y los detalles sobre el uso de Java como lenguaje y entorno de ejecución para muchas plataformas de agentes móviles y sus implicaciones en la seguridad de éstas. Después describiremos el entorno utilizado para el desarrollo del proyecto y el entorno en el que se han realizado las pruebas funcionales y en las pruebas de carga y escalabilidad del mismo.

## A.1 Conceptos generales de seguridad informática

A continuación se describen los conceptos básicos de seguridad en sistemas informáticos [55, 56]:

- Identificación. El receptor de un mensaje deber de ser capaz de identificar al emisor del mismo (Figura A.1).

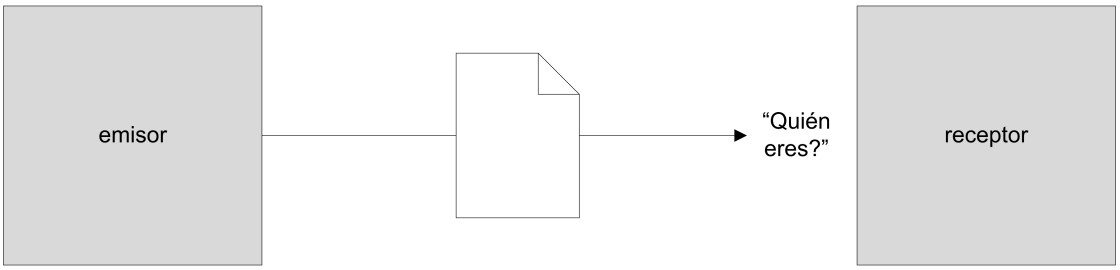



Figura A.1: Identificación del emisor por parte del receptor

- Autenticación. El receptor de un mensaje necesita verificar que la identidad que dice el emisor que tiene es válida (Figura A.2).

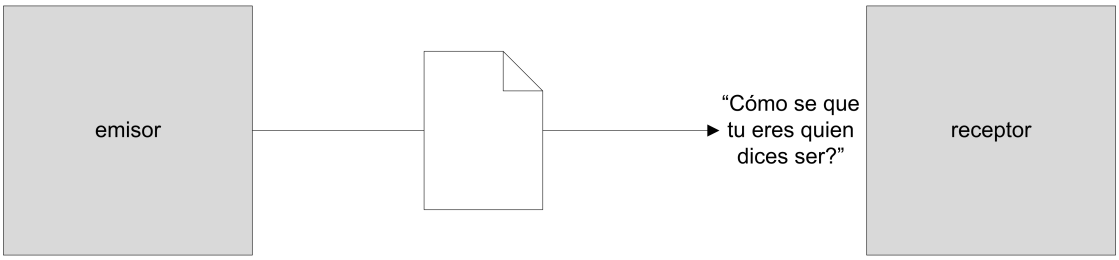



Figura A.2: Autenticación del emisor por parte del receptor

- Autorización. El receptor de un mensaje necesita determinar el nivel de seguridad al que el emisor tiene acceso (Figura A.2). Esto puede estar relacionado con a qué operaciones el emisor puede acceder o a qué información tiene acceso.

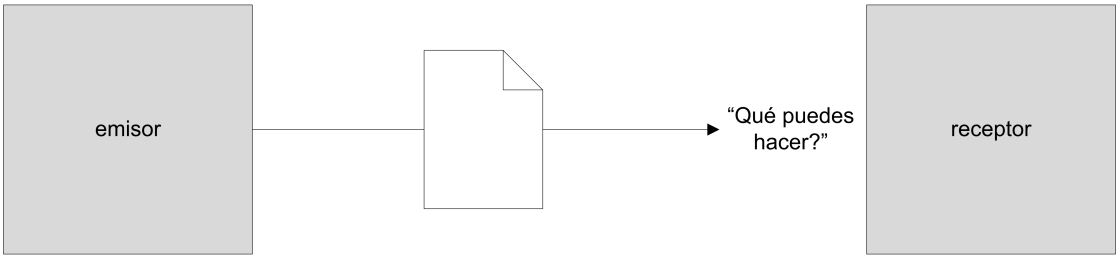



Figura A.3: El receptor identifica el nivel de autorización del emisor

- Integridad. La información intercambiada durante la transmisión debe permanecer inalterada hasta la recepción de la misma (Figura A.4).

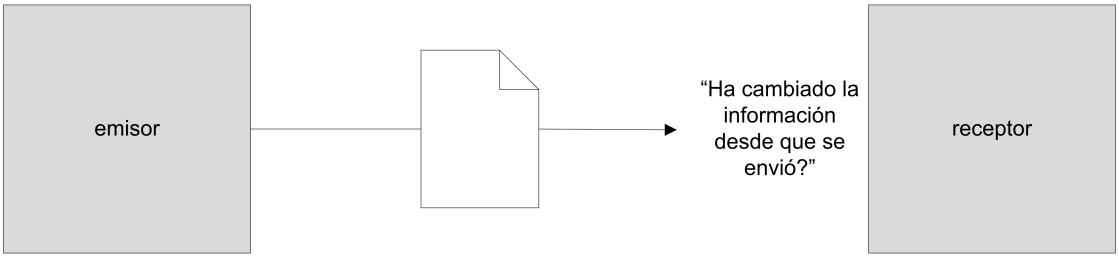



Figura A.4: La integridad de la información es cuestionada por el receptor

- Confidencialidad. La información transmitida no debe ser vista mientras esté en tránsito, excepto por los servicios autorizados (Figura A.5).

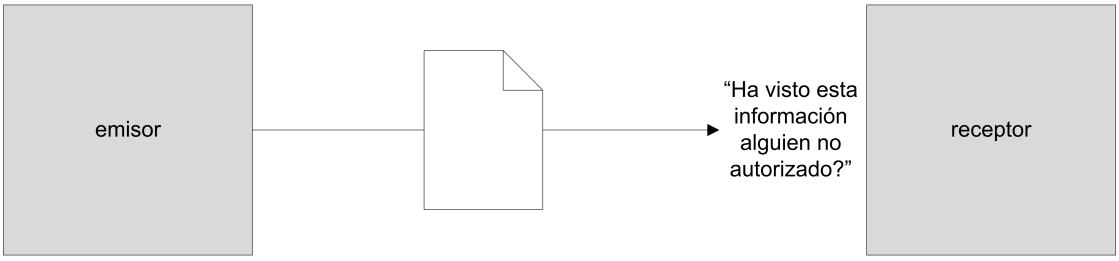



Figura A.5: La confidencialidad de la información es cuestionada por el receptor

Habitualmente se suele identificar el concepto de seguridad respecto a amenazas conocidas. En la mayoría de los casos existe una tendencia a identificar la seguridad con la autorización y la autenticación. Estos conceptos se limitan, junto con alguna característica adicional, a los *firewalls* y a la autenticación vía usuario y clave a nivel de aplicativo. Además, la criptografía ofrece métodos muy eficientes para ofrecer seguridad en mensajes y datos. Todos estos aspectos son, obviamente, parte de los mecanismos de seguridad de los agentes, pero estos agentes, son algunas veces móviles, lo que implica nuevas amenazas.

En el contexto de la tecnología de los agentes móviles consideramos tres clases principales de seguridad [28]:

1. Seguridad externa: Éste es el concepto típico de la seguridad en cualquier sistema software. Cubre los términos previamente mencionados, como *firewall*, autenticación, autorización y cifrado. En un enfoque clásico del diseño de sistemas, este nivel se define como la capa de seguridad de la arquitectura. Su objetivo es la protección contra ataques desde fuera del sistema.

2. Seguridad interna: Se refiere a la integridad de los componentes del software, una vez que la seguridad externa se ha aplicado pero se ha visto comprometida. Algunas facetas de dicha clase de seguridad pueden ser la detección de intrusos y/o su neutralización. En la arquitectura de un sistema de agentes móviles, este nivel se traduciría en módulos software y en una metodología, pero no en una capa de la arquitectura.

3. Integridad y privacidad: Esta clase contiene métodos para proteger la integridad del "estado" del sistema. Es una especie de segunda capa de la seguridad externa, que debería permitir al sistema degradarse "elegantemente" una vez que la seguridad externa se ha visto comprometida o que objetos "maliciosos" forman parte del sistema. Los métodos necesarios para la protección del sistema son bastante eficaces, teniendo en cuenta el estado del arte de los algoritmos utilizados en criptografía (AES (Advanced Encryption Standard), funciones *hash*, firmas digitales). Este nivel no suele aparecer en ninguna arquitectura de sistemas, excepto posiblemente como un módulo criptográfico.

En la práctica, los aspectos de seguridad se pueden categorizar de acuerdo con los requisitos generales expuestos a continuación [11, 13]:

- Autenticación: Un interlocutor tiene que ser capaz de probar que es quien dice ser. Basándose en la autenticación, un interlocutor puede decidir si confía o no en el otro interlocutor. La autenticación se puede basar en mecanismos de certificados y firmas digitales.
- Confidencialidad: La información tiene que estar protegida contra accesos no autorizados. La confidencialidad entre interlocutores se consigue normalmente usando criptografía.
- Integridad: La integridad quiere decir que la información no ha sido alterada. Las funciones de *hash*, como por ejemplo MD5 (Message-Digest algorithm 5), permiten que la integridad de un mensaje sea verificada. Se usan habitualmente en combinación con firmas digitales o con códigos de autenticación de mensajes (MAC).
- Responsabilidad: La responsabilidad se refiere a que cada parte de una comunicación "se hace cargo" de todas las acciones que pueda realizar un agente. Esto se consigue mediante la firma de un "contrato" entre las partes, usando firmas digitales.
- Disponibilidad: El acceso a un servicio no debe estar restringido de una forma no prevista. La disponibilidad garantiza un acceso fiable y rápido a los datos y recursos por parte de las entidades autorizadas.

## A.2 Seguridad en Java

El lenguaje Java ha tenido y sigue teniendo un papel muy importante en el desarrollo de plataformas de agentes móviles. El uso de un lenguaje interpretado facilita la construcción de un sistema de agentes móviles [57]. En ciertos casos puede ser necesario el acceso a variables globales y/o a punteros con la dirección de la instrucción que se está ejecutando en determinado momento o a la dirección de la pila de un proceso. La forma más sencilla de hacer esto es utilizando una máquina virtual que ejecute un lenguaje interpretado. Originalmente se crearon lenguajes interpretados con tal propósito como Telescript [58], si bien a lo largo de los años no han trascendido. También se utilizaron otros lenguajes interpretados como Agent TCL [59], siendo la plataforma D'Agents [60] la más representantiva escrita en dicho lenguaje.

Java, aun siendo un lenguaje de propósito general, dispone de una serie de características que lo hacen idóneo para la construcción de sistemas de agentes móviles, que son las siguientes:

- Independencia de plataforma. El lenguaje Java fue diseñado con el objetivo de "escribir una vez, ejecutar en cualquier sitio" [61]. El código Java se compila en un código máquina específico, llamado *Bytecode*, que se interpreta por la JVM. Existen JVMs para la mayoría de sistemas operativos que existen con lo que el código escrito debería poder ejecutarse en gran cantidad de máquinas sin necesidad de ser recompilado.

- Diseño orientado a objetos. Java es un lenguaje orientado a objetos que facilita la extensión de partes de una plataforma como puede ser un agente. De esta forma se facilita el mantenimiento de proyectos grandes con muchos subsistemas.

- Serialización. Java ofrece un mecanismo de serialización integrado. La serialización se utiliza para almacenar el estado actual de un programa para, por ejemplo, enviarlo a través de la red. Esto ofrece una manera estándar de enviar agentes móviles por la red. También se puede utilizar para hacer persistente el estado de ejecución de un agente para almacenarlos o para pausarlos.

- Redes. El lenguaje Java ofrece al programador una serie de métodos para realizar cualquier tipo de comunicación a través de la red. Estos métodos pertenecen a clases que forman parte del *core* de Java.

- Seguridad. Java ofrece una extensa serie de mecanismos de seguridad [62, 63] que pueden ser utilizados, adaptados o extendidos para su uso por plataformas de agentes móviles.

- *Reflection*. Java ofrece acceso a los métodos y variables de un objeto durante su ejecución. Las librerías estándar de Java contienen el paquete `java.lang.reflect` que cubre el acceso a métodos, así como a variables globales de un objeto ejecutándose en la JVM.

Aun así, existen ciertas desventajas en el uso de Java que suelen ser debidas a su independencia de plataforma:

- Puntero de ejecución. No se puede acceder a la pila de un programa en Java. Si un programa es serializado, solo sus variables globales serán serializadas, pero no su estado de ejecución. Este problema hace que las plataformas desarrolladas en Java no puedan implementar la conocida como "strong mobility" teniendo que utilizarse la "weak mobility" [44] ya que no es posible recuperar el estado de ejecución de la pila y restaurarlo.

- Control de recursos. Otra desventaja es la falta de control recursos (como memoria o uso de CPU por parte de *threads*) en la JVM. No existe, por ejemplo, un método `kill` o `destroy` para un *thread*. Solo se puede configurar en la creación de un *thread* el tipo de prioridad del mismo frente a la ejecución de otros *threads*, pero

no se puede evitar que determinado programa haga un uso no apropiado de los recursos de la JVM.

Como se ha mencionado previamente, Java ofrece una serie de mecanismos de seguridad incluidos por defecto en el plataforma. Dichos mecanismos incluyen [63]:

- El Verificador de *bytecode*. Como hemos comentado, todo el código fuente Java se compila en una especie de código máquina intermedio denominado *bytecode*. Dicho código es el que posteriormente se ejecuta en la JVM. Cuando se va a cargar el *bytecode* en la JVM se verifica la corrección del mismo mediante el verificador de *bytecode*. Dicho verificador se asegura de que los ficheros de clases siguen las reglas del lenguaje Java. El verificador también comprueba que todas las asignaciones de memoria son correctas, impidiendo accesos no permitidos [64].

- El *Classloader*. Es la clase responsable de buscar y cargar las clases Java en tiempo de ejecución. Se pueden crear tantos *Classloaders* como se considere oportunos, modificando el comportamiento de los mismos según sea necesario [65]. Por ejemplo, se puede necesitar que un *Classloader* solo cargue clases de ciertos servidores, no pudiendo cargarse clases de sitios no permitidos.

- El *Access Controller*. Permite o evita la mayoría de accesos del *core* de Java a sistema operativo, basándose en políticas de seguridad definidas por el usuario o por el administrador de sistemas.

- El *Security Manager*. Es el interfaz principal entre el *core* de java y el sistema operativo. Tiene la responsabilidad última de permitir o denegar el acceso a todos los recursos del sistema. Sin embargo existe principalmente por razones históricas, delegando toda su funcionalidad en el *Access Controller*.

- El paquete de seguridad. Las clases contenidas en el paquete `java.security` [66] al igual que las de las extensiones de seguridad permiten añadir características de seguridad a una aplicación. Estas clases ofrecen, entre otras, las siguientes funcionalidades:

  - Interfaz para proveedores de seguridad. Mediante este interfaz se pueden utilizar diferentes implementaciones de seguridad.
  - *Hashes* de mensajes.
  - Gestión de claves y certificados.
  - Firmas digitales.
  - Cifrado mediante el uso de JCE (Java Cryptography Extension) y de JSSE.
  - Autenticación mediante el uso de JAAS.

Como se puede comprobar, Java ofrece un extenso listado de medidas de seguridad siendo éstas, además, fácilmente extensibles y programables. Esto hace del lenguaje Java una plataforma ideal para el desarrollo de un sistema de agentes móviles seguro. Si bien, como se ha comentado previamente, existe un problema importante con la gestión de recursos que hace que las plataformas de agentes móviles basadas en la JVM estándar no puedan defenderse correctamente ante ciertos ataques de denegación de servicio y de abuso de recursos de la plataforma [67].

Debido al uso que tiene en las plataformas de agentes móviles, destacamos el paquete JAAS. El objetivo principal de JAAS es gestionar la emisión de permisos y, a su vez, realizar las comprobaciones de seguridad necesarias para dichos permisos [68].

JAAS forma parte del *framework* de seguridad de Java desde la versión 1.4 y estuvo disponible como un paquete opcional desde J2SE 1.3. La mayoría de clases se encuentran en el paquete `javax.security.auth`. Las tres clases principales son `LoginContext`, `Subject` y `PrivilegedAction` [69].

En JAAS la autenticación se realiza a través del *login*, que, si se ejecuta con éxito, provee un conjunto de permisos y, por lo tanto, de autorizaciones. Los *logins* se encuentran relacionados con un contexto determinado que representa un escenario en la aplicación (estamos hablando de contextos de seguridad a nivel de JAAS, no de los contextos como contenedores en SPRINGS). Mediante el uso de contextos se permite crear un sistema modular, donde en cada escenario se pueden utilizar métodos de *login* específicos. JAAS implementa la versión Java del *framework* PAM (Pluggable Authetication Module) muy utilizado en sistemas UNIX [52]. Mediante él, se permiten definir tantos métodos de autenticación como se consideren necesarios de una forma "apilada", permitiendo definir el flujo de autenticación en la invocación a cada uno de los módulos.

Los permisos son derechos a ejecutar una acción sobre un elemento. Por ejemplo, un permiso puede ser "todas las clases del paquete `hola.mundo` pueden abrir *sockets* a la dirección `www.hola.mundo`". Existen una serie de permisos ya definidos en las clases internas de Java, como pueden ser los pertenecientes a los paquetes [70]:

```
java.security.AllPermission
java.security.SecurityPermission
java.security.UnresolvedPermission
java.awt.AWTPermission
java.io.FilePermission
java.io.SerializablePermission
java.lang.reflect.ReflectPermission
java.lang.RuntimePermission
java.net.NetPermission
java.net.SocketPermission
java.sql.SQLPermission
java.util.PropertyPermission
java.util.logging.LoggingPermission
javax.net.ssl.SSLPermission
```

```
javax.security.auth.AuthPermission
javax.security.auth.PrivateCredentialPermission
javax.security.auth.kerberos.DelegationPermission
javax.security.auth.kerberos.ServicePermission
javax.sound.sampled.AudioPermission
```

No obstante, se pueden crear permisos nuevos mediante la extensión de la clase `java.security.Permission`.

Cuando un proceso ejecuta acciones protegidas, ese código acaba invocando al `SecurityManager` (y, por lo tanto, al `AccessControler`), que comprueba si se dispone de los permisos necesarios. Habitualmente se realizaría la comprobación contra el conjunto de permisos del sistema. Para invocar a JAAS es necesario que las acciones protegidas que se quiere que se invoquen bajo el conjunto de permisos de una identidad autenticada en el JAAS se ejecuten mediante los métodos `doAs` o `doAsPrivileged` [63].

## A.3 Entorno de desarrollo

En esta sección vamos a describir el entorno tecnológico sobre el que se ha desarrollado el presente proyecto.

### A.3.1 Lenguaje de programación Java

La plataforma de programación y ejecución sobre la que se ha desarrollado el proyecto ha sido Java de Oracle Corp. Más detalladamente, se ha desarrollado en su tramo final sobre un JDK con la siguiente versión:

```
java version "1.6.0_51"
Java(TM) SE Runtime Environment (build 1.6.0_51-b11-457-11M4509)
Java HotSpot(TM) 64-Bit Server VM (build 20.51-b01-457, mixed mode)
```

### A.3.2 *Hardware* de desarrollo

El proyecto se ha desarrollado en su parte principal sobre un sistema Apple Macbook Pro [71] con las características de la Tabla A.1.

### A.3.3 Herramientas adicionales

Como editores de texto se han utilizado Macvim [72], tanto para código como para documentación, y Eclipse [73], como editor de código exclusivamente.

| Modelo | Apple Macbook Pro 13" |
|---|---|
| Procesador | Intel i7 2.9GHz (Dual Core) |
| Memoria | 8GB DDR3 |
| Sistema Operativo | Mac OS X 10.8.4 |
| Virtualizador | Oracle VirtualBox 4.2.12 |

Tabla A.1: Características de máquina de desarrollo

Al ser entregada la plataforma, ésta se construía a través de una serie de *scripts* desarrollados en Bash. Sin embargo se ha preferido utilizar una manera más estándar y modular de realizar la compilación del código de la plataforma mediante el *software* Apache Ant [74]. Dicho *software* se utiliza para la compilación de código y para la generación de documentación a nivel de APIs a través de Javadoc [75]. Para generar documentación sobre el API más extensa que incluye diagramas UML (Unified Modeling Language), también se utiliza el software Doxygen [76].

Para controlar el versionado del código y de la documentación, se ha creado también un repositorio de código utilizando el software subversion [77] alojado en un servidor perteneciente al proyecto Zonazener [78].

El presente documento ha sido escrito utilizando LaTex [79].

## A.4 Entorno de pruebas

A continuación vamos a describir cada uno de los elementos tecnológicos utilizados durante el desarrollo de las pruebas realizadas en la plataforma.

### A.4.1 *Software* de gestión de certificados digitales

Para la gestión de certificados digitales hemos utilizado el software EJBCA [80]. Con él, hemos gestionado la CA utilizada en los entornos de pruebas para crear los certificados digitales utilizados tanto por el RNS como por los contextos.

### A.4.2 Máquinas virtuales para pruebas funcionales

Con el objeto de ofrecer un entorno lo más similar posible a un entorno final pero mucho más flexible y utilizable en un único sistema informático (el descrito en la Sección A.3) para realizar las pruebas funcionales de la plataforma, se ha implantado un sistema basado en máquinas virtuales. Concretamente, se utiliza el *software* de virtualización VirtualBox [81], en su versión 4.2.12 para Mac OS X.

Sobre dicho entorno se han creado varios servidores virtuales con las características de la Tabla A.2, cada uno conteniendo uno de los contextos de las pruebas funcionales.

| Procesador | 1 procesador virtual |
|---|---|
| Memoria | 1GB |
| Sistema Operativo | Linux Debian 5.0 |
| Versión de JAVA | 1.6.0-26-b03 |

Tabla A.2: Características de servidor virtual

De esta forma, hemos sido capaces de simular las características propias de ejecución de la plataforma simulando varios servidores diferentes siendo, además, capaces de realizar tareas como capturas de tráfico de red mientras las pruebas se ejecutan realmente en un único sistema físico.

### A.4.3 *Hardware* utilizado en las pruebas de carga y escalabilidad

Las pruebas de carga realizadas en la plataforma se han ejecutado utilizando *hardware* disponible en el Laboratorio 1.03a del Edificio Ada Byron del Campus Río Ebro de la Universidad de Zaragoza.

Se ha utilizado tres PCs de dicho laboratorio en conjunto con el ordenador utilizado para el desarrollo del presente proyecto.

Debido a la diversidad de Sistemas Operativos y al acceso limitado a la administración de los mismos, se ha decidido utilizar una capa de virtualización similar a la descrita para las pruebas funcionales en el entorno de las pruebas de carga, basada en el software Oracle VirtualBox.

El *hardware* utilizado ha sido:

- Apple MacBook Pro. Intel i7 2.9GHz (Dual Core), 8GB RAM, Mac OS X 10.8.4, Oracle VirtualBox 4.2.12.
- PC 1. Intel i5 3GHz (Dual Core). 16GB RAM, Microsoft Windows 7 Profesional SP1, Oracle VirtualBox 4.2.16.
- PC 2. Intel i5 3GHz (Dual Core). 16GB RAM, Microsoft Windows 7 Profesional SP1, Oracle VirtualBox 4.2.16.
- PC 3. Intel i5 3GHz (Dual Core). 16GB RAM, Microsoft Windows 7 Enterprise SP1, Oracle VirtualBox 4.2.16.

Todos los dispositivos tiene adaptadores de red Ethernet a 1 Gigabit/s; sin embargo,

el *switch* al que se encuentran conectados funciona a 100Mbit/s. El entorno de pruebas se muestra en el diagrama de la Figura A.6.

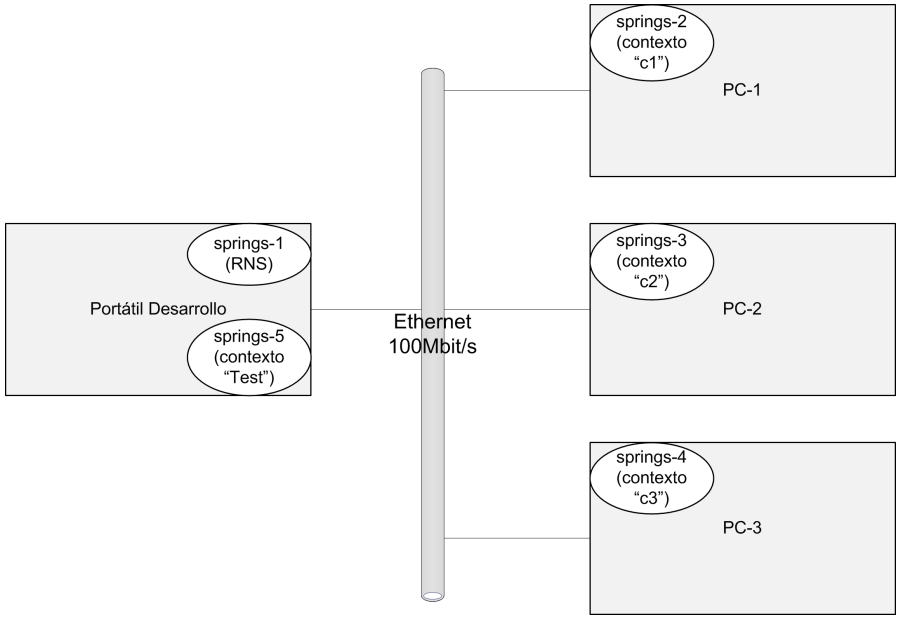



Figura A.6: Entorno de pruebas

En el Apple MacBook Pro se están ejecutando dos máquinas virtuales con las siguientes características:

- springs-1. 3 procesadores virtuales, 1GB RAM, Linux Debian 5.0, Java 1.6.0-26-b03. Ejecuta la instancia del RNS en las pruebas.
- springs-5. 1 procesador virtual, 1GB RAM, Linux Debian 5.0, Java 1.6.0-26-b03. Ejecuta la instancia del contexto Test en las pruebas.

En el PC 1 se ejecuta:

- springs-2. 4 procesadores virtuales, 1GB RAM, Linux Debian 5.0, Java 1.6.0-26-b03. Ejecuta la instancia del contexto "c1" en las pruebas.

En el PC 2 se ejecuta:

- springs-3. 4 procesadores virtuales, 1GB RAM, Linux Debian 5.0, Java 1.6.0-26-b03. Ejecuta la instancia del contexto "c2" en las pruebas.

En el PC 3 se ejecuta:

- springs-4. 4 procesadores virtuales, 1GB RAM, Linux Debian 5.0, Java 1.6.0-26-b03. Ejecuta la instancia del contexto "c3" en las pruebas.

En la Figura A.6 podemos encontrar un diagrama que contiene el entorno utilizado para las pruebas.

### A.4.4 Agente y contexto para pruebas de carga

El contexto "Test" es un contexto de la clase `LoadTestContext`. En dicha clase definimos el funcionamiento del contexto. Básicamente, el contexto acepta como parámetros de entrada el número de agentes a crear, el número de contextos de prueba existentes, el número de movimientos que deben realizar los agentes, si el contexto debe utilizar SSL o autenticación y el tiempo que debe esperar el contexto (*warm-up time*) para comenzar la prueba. También se incluye un *script* llamado `TestLauncher` que facilita el lanzamiento del contexto "Test". Su invocación es como sigue:

```
$ ./TestLauncher
Mandatory arguments missing:
 -aNN Number of agents
 -cNN Number of contexts
 -mNN Number of movements
 -s   Activate SSL
 -t   Activate Authentication
 -wNN Time to wait for the creation of the agents
```

En ella podemos ver cómo introducir los parámetros de entrada previamente definidos. Una vez introducidos y analizados dichos parámetros, la clase realiza las siguientes acciones:

- Intenta leer la información referente a su *keystore* y la almacena en el objeto `ks`.
- Crea un nuevo contexto con los parámetros de entrada definidos en el arranque y el *keystore* si fuera necesario.
- Crea una lista que contiene todos los agentes que formarán parte de la prueba para definir quién será el agente "pareja" de cada uno.
- Inicia un bucle en el que se van creando cada uno de los agentes, indicando en cada caso quién es el agente "pareja" del agente, el número de contextos que existen, el número de movimientos que tiene que realizar el agente y el tiempo que debe esperar para iniciar su funcionamiento.

Los agentes creados pertenecen a la clase `LoadTestAgent`. Dichos agentes realizan las siguientes funciones:

- Al ser creados, esperan el tiempo definido por el parámetro `warmUpTime`.
- Una vez terminado dicho intervalo, deciden aleatoriamente a qué contexto de los posibles va a viajar y se mueve a dicho contexto para ejecutar el método `travel()`.
- El método `travel()` en primer lugar, analiza si ya se han realizado todos los movimientos requeridos. Si es así hace que el agente se mueva al contexto en el que fue creado para ejecutar el método `end()`.
- Si el agente no debe volver todavía a su contexto, intenta realizar una llamada al método `hello()` de su agente "pareja". Si no lo consigue, se reintenta.
- Una vez realizada la llamada al agente "pareja", el agente se mueve a uno de los contextos disponibles, elegido aleatoriamente, para ejecutar de nuevo el método `travel()`, previo incremento de la variable que almacena el número de viajes realizados.
- Cuando el agente termina, vuelve al contexto en el que fue creado y ejecuta el método `end()`, se imprimen las estadísticas de tiempos de estancia en cada uno de los contextos y de tiempos de llamada para su posterior análisis.

### A.4.5 *Script* de análisis de resultados de pruebas de carga

Se ha desarrollado un script (llamado `get_stats.pl`) en el lenguaje de programación Perl [82] para el análisis de los resultados de las pruebas de carga.

Dicho script lee los *logs* generados en el contexto "Test" y calcula las estadísticas necesarias para realizar el análisis. Dichas estadísticas son las siguientes:

```
Test Conditions:
Number of agents: 1500
Number of contexts: 3
Number of Movements: 50
Test Results:
Number of agents created 1500
Number of finished agents: 1500
Test start time: 2013-09-01T11:34:27
Test end time: 2013-09-01T11:47:58
Time elapsed (s): 810
Number of stays: 75000
Average staying time (ms): 8008.08157333333
Median staying time (ms): 639
Mode staying time (ms): 16
Std deviation staying time (ms): 23745.205270202
Number of calls: 75000
Average call time (ms): 5470.29910666667
Median call time (ms): 66
Mode call time (ms): 10
Std deviation call time (ms): 21155.288896229
```

Estas estadísticas son las que se han utilizado para realizar el análisis y las figuras de las Sección 4.2.

# Apéndice B

# Pruebas funcionales

En este apéndice vamos a detallar todas las pruebas funcionales realizadas sobre la plataforma.

Para facilitar la lectura y la consulta de esta sección se ha decidido realizar una organización homogénea en la presentación de las pruebas. En primer lugar se expondrá un tipo de ataque a algún elemento de la plataforma. A continuación se explicará qué medida o medidas de seguridad desarrolladas evitarán el ataque descrito. Posteriormente se describirá el escenario de realización de las pruebas y su ejecución, pasando al análisis del resultado.

## B.1 Acceso a la plataforma de un contexto que no tiene activado SSL

En esta prueba se demostrará cómo, si la capa de seguridad externa está activada en una región, no es posible acceder a la misma sin utilizar SSL.

Para ello utilizaremos:

- Un servidor ejecutando un RNS con la capa de seguridad activa mediante la siguiente configuración:

```
[security]

ssl = true
keystore = ../etc/rns.keystore
keystorePass = springs
```

- Un servidor ejecutando un contexto “c1-maligno” con la siguiente configuración:

```
[security]

ssl = false
```

Una vez arrancado el RNS, al lanzar el contexto "c1-maligno" vemos cómo se aborta su ejecución obteniendo lo siguiente en su log:

```
2013-08-20 22:50:25,566 -- INFO -- Context SSL is false
2013-08-20 22:50:25,566 -- INFO -- Context Authentication is false
2013-08-20 22:50:30,777 -- ERROR -- Error starting context springs.
    context.ContextStartingException: c1-maligno: Error communicating with
     RNS springs.common.CommunicationException: c1-maligno: Security
    enabled in only one end: non-JRMP server at remote endpoint
```

Indicando que ha habido un error en la comunicación con el RNS debido a que la capa de seguridad (SSL) solo está activa en el lado del RNS.

## B.2 Acceso a la plataforma de un contexto que usa un certificado no válido

En esta prueba se demostrará cómo la capa de seguridad externa defiende a la plataforma de accesos de contextos que estén utilizando un certificado no aceptado (la JVM no confía en la entidad que firmó el certificado del cliente) por el RNS.

Para ello utilizaremos:

- Un servidor ejecutando el RNS con un certificado firmado por la CA "springsCA":

```
# keytool -keystore rns.keystore  -storetype JCEKS -list -v
Enter keystore password:

Keystore type: JCEKS
Keystore provider: SunJCE

...

Alias name: authkey
Creation date: Sep 10, 2012
Entry type: PrivateKeyEntry
Certificate chain length: 2
Certificate[1]:
Owner: C=es, L=zaragoza, O=unizar, OU=springs, CN=rns
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Serial number: 5a29306e2a453006
Valid from: Mon Sep 10 18:55:01 CEST 2012 until: Wed Sep 10 18:55:01
    CEST 2014
```

```
Certificate fingerprints:
         MD5:  DB:F9:DA:F3:BF:DB:BD:95:43:42:71:51:CD:AA:AB:24
         SHA1: A2:9B:DE:CB:2E:72:CD:93:F9:43:AD:6F:6F:D0:3A:60:1F:E6
    :4C:45
         Signature algorithm name: SHA1withRSA
         Version: 3

...

Certificate[2]:
Owner: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Serial number: 3e8f7eb6a8e5ac4b
Valid from: Wed Sep 05 20:33:40 CEST 2012 until: Mon Sep 05 20:30:25
     CEST 2022
Certificate fingerprints:
         MD5:  9D:80:A8:34:2B:B3:A1:ED:9A:2E:E9:7F:21:23:E0:15
         SHA1: F8:1C:0E:0B:49:9A:84:20:8A:88:7B:75:9A:BD:23:9D:81:9A
    :4A:4C
         Signature algorithm name: SHA1withRSA
         Version: 3

...
```

La clave pública de la CA está incluida en el fichero `cacerts` de la máquina virtual que ejecuta el RNS.

```
 keytool -keystore /System/Library/Java/Support/CoreDeploy.bundle/
    Contents/Home/lib/security/cacerts -list
...
springsca, Jun 24, 2013, trustedCertEntry,
Certificate fingerprint (MD5): 9D:80:A8:34:2B:B3:A1:ED:9A:2E:E9:7F
    :21:23:E0:15
...
```

La configuración relevante del RNS es:

```
[security]

ssl = true
keystore = ../etc/rns.keystore
keystorePass = springs
```

- Un servidor ejecutando un contexto “c1-maligno” tiene como certificado:

```
# keytool -keystore c1-maligno.keystore  -storetype JCEKS -list -v
Enter keystore password:

Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 1 entry

Alias name: c1-maligno
```

```
Creation date: Aug 20, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 2
Certificate[1]:
Owner: C=es, L=zaragoza, O=unizar, OU=springs, CN=c1-maligno
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=malignoCA
Serial number: 28adf6551ed69928
Valid from: Tue Aug 20 19:12:40 CEST 2013 until: Mon May 12 19:11:56
    CEST 2014
Certificate fingerprints:
         MD5:  66:0B:62:A6:7F:A7:17:5E:9D:F0:F3:4B:3C:80:A4:61
         SHA1: 36:4E:F3:BC:E7:0F:E8:A4:23:71:02:79:E5:5C:CF:DF:94:C6
   :9C:4F
         Signature algorithm name: SHA256withRSA
         Version: 3

...

Certificate[2]:
Owner: C=ES, L=zaragoza, O=unizar, OU=springs, CN=malignoCA
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=malignoCA
Serial number: 1b8d1f818ca5556c
Valid from: Tue Aug 20 19:11:56 CEST 2013 until: Mon May 12 19:11:56
    CEST 2014
Certificate fingerprints:
         MD5:  3D:CD:C7:C1:BB:7A:0F:BB:45:9A:78:BA:B5:BD:B2:9E
         SHA1: F3:27:5B:22:44:3F:B9:CF:E8:81:27:B4:13:67:97:58:D4:E4
   :C6:A7
         Signature algorithm name: SHA256withRSA
         Version: 3
...
```

La CA "malignoCA" no se encuentra en el fichero `cacerts` de la JVM del RNS.

La configuración relevante del contexto "c1-maligno" es:

```
[security]

ssl = true
keystore = ../etc/c1-maligno.keystore
keystorePass = springs
```

Una vez arrancado el RNS, al lanzar el contexto "c1-maligno" vemos cómo se aborta su ejecución obteniendo lo siguiente en el log:

```
2013-08-20 21:13:24,867 -- INFO -- Context SSL is true
2013-08-20 21:13:24,867 -- INFO -- Context Authentication is false
2013-08-20 21:13:31,280 -- ERROR -- Error starting context springs.
   context.ContextStartingException: c1-maligno: Error communicating with
    RNS springs.common.CommunicationException: c1-maligno: Security
   enabled in only one end: error during JRMP connection establishment;
   nested exception is:
       javax.net.ssl.SSLHandshakeException: Received fatal alert:
   bad_certificate
```

Indicando que ha habido un problema con la gestión de certificados SSL entre el contexto y el RNS y, por lo tanto, el contexto “c1-maligno” no ha podido ejecutarse.

## B.3 Acceso y/o modificación de comunicaciones entre el RNS y los contextos o entre contextos

En esta prueba se demostrará cómo cuando la capa de seguridad externa se encuentra activada en una región no es posible acceder a los datos intercambiados entre un contexto y el RNS o entre diferentes contextos.

En una comunicación RMI sin SSL se puede observar mediante una captura de red realizada con una herramienta como Tcpdump [83] cómo el tráfico no está cifrado. En la siguiente captura visualizada según su volcado ASCII se puede observar a simple vista cómo el tráfico no se encuentra cifrado:

```
JRMI..KN..192.168.56.2......192.168.56.2....P....w".......................D.M
    ...;.t..RegionNameServerQ....w....5....@.SUq..sr.)springs.rns.
    RegionNameServer_RMIImpl_Stub...........pxr..java.rmi.server.RemoteStub......e
    ....pxr..java.rmi.server.RemoteObject.a...a3....pxpw6.
UnicastRef.
192.168.2.101..'.Q.E.\..K..5....@.SUq...xRST..5....@.SUq..
```

Si bien es complicado decodificar tráfico RMI, si se obtienen los ficheros de clases específicos sí que se podría tener acceso a los datos serializados y poder leerlos o, incluso, modificarlos.

Con una herramienta como Wireshark [84] podemos analizar el tráfico de una forma sencilla tal y como se muestra en la Figura B.1.

| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 1 | 192.168.56.2 | 192.168.56.1 | TCP | 74 | 33251 > ndmp [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=4207654 TSecr=0 WS=32 |
| 2 | 192.168.56.1 | 192.168.56.2 | TCP | 78 | ndmp > 33251 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=16 TSval=813933398 TSecr=4207654 SACK_PERM=1 |
| 3 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33251 > ndmp [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=4207654 TSecr=813933398 |
| 4 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | [TCP Window Update] ndmp > 33251 [ACK] Seq=1 Ack=1 Win=131760 Len=0 TSval=813933398 TSecr=4207654 |
| 5 | 192.168.56.2 | 192.168.56.1 | RMI | 73 | JRMI, Version: 2, StreamProtocol |
| 6 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33251 [ACK] Seq=1 Ack=8 Win=131760 Len=0 TSval=813933399 TSecr=4207654 |
| 7 | 192.168.56.1 | 192.168.56.2 | RMI | 85 | JRMI, ProtocolAck |
| 8 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33251 > ndmp [ACK] Seq=8 Ack=20 Win=5856 Len=0 TSval=4207655 TSecr=813933401 |
| 9 | 192.168.56.2 | 192.168.56.1 | RMI | 84 | Continuation |
| 10 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33251 [ACK] Seq=20 Ack=26 Win=131728 Len=0 TSval=813933402 TSecr=4207655 |
| 11 | 192.168.56.2 | 192.168.56.1 | RMI | 126 | JRMI, Call |
| 12 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33251 [ACK] Seq=20 Ack=86 Win=131680 Len=0 TSval=813933405 TSecr=4207656 |
| 13 | 192.168.56.1 | 192.168.56.2 | RMI | 290 | JRMI, ReturnData |
| 14 | 192.168.56.2 | 192.168.56.1 | RMI | 67 | JRMI, Ping |
| 15 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33251 [ACK] Seq=244 Ack=87 Win=131680 Len=0 TSval=813933451 TSecr=4207668 |
| 16 | 192.168.56.1 | 192.168.56.2 | RMI | 67 | JRMI, PingAck |
| 17 | 192.168.56.2 | 192.168.56.1 | RMI | 81 | JRMI, DgcAck |
| 18 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33251 [ACK] Seq=245 Ack=102 Win=131664 Len=0 TSval=813933451 TSecr=4207668 |

Figura B.1: Captura de red tráfico RMI no cifrado

El escenario que vamos a utilizar en esta prueba es el siguiente:

- Un servidor ejecutando un RNS con la capa de seguridad activa mediante la siguiente configuración:

```
[security]

ssl = true
keystore = ../etc/rns.keystore
keystorePass = springs
```

El *keystore* del RNS está configurado como el de la prueba del Apéndice B.2.

- Un servidor ejecutando un contexto "c1" con la siguiente configuración:

```
[security]

ssl = true
keystore = ../etc/c1.keystore
keystorePass = springs
```

El *keystore* del contexto "c1" contiene los siguientes certificados:

```
# keytool -keystore c1.keystore -storetype JCEKS -list -v
Enter keystore password:

Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 1 entry

Alias name: c1
Creation date: Sep 10, 2012
Entry type: PrivateKeyEntry
Certificate chain length: 2
Certificate[1]:
Owner: C=es, L=zaragoza, O=unizar, OU=springs, CN=c1
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Serial number: 34948d415309b1e6
Valid from: Mon Sep 10 19:07:29 CEST 2012 until: Wed Sep 10 19:07:29
    CEST 2014
Certificate fingerprints:
         MD5:  17:36:31:0A:0C:80:51:84:3A:69:1C:A0:D6:A2:59:E6
         SHA1: 4A:F5:47:27:77:9B:97:DD:CD:F1:40:56:F8:B6:F0:F1:A4:CB
   :22:E2
         Signature algorithm name: SHA1withRSA
         Version: 3

...

Certificate[2]:
Owner: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Issuer: C=ES, L=zaragoza, O=unizar, OU=springs, CN=springsCA
Serial number: 3e8f7eb6a8e5ac4b
Valid from: Wed Sep 05 20:33:40 CEST 2012 until: Mon Sep 05 20:30:25
    CEST 2022
Certificate fingerprints:
         MD5:  9D:80:A8:34:2B:B3:A1:ED:9A:2E:E9:7F:21:23:E0:15
         SHA1: F8:1C:0E:0B:49:9A:84:20:8A:88:7B:75:9A:BD:23:9D:81:9A
   :4A:4C
```

```
          Signature algorithm name: SHA1withRSA
          Version: 3

...
```

Es decir, su certificado está firmado por la misma CA que el RNS. Además, tanto la JVM del RNS como la del contexto confían en la clave pública de dicha CA.

- Un administrador que, simulando un tercero "maligno", tenga acceso al tráfico de red.

En este escenario, una vez arrancado el RNS y viendo cómo está activa su configuración de SSL en el log:

```
2013-08-21 13:03:40,791 -- INFO -- Started RNS at port 10000!
2013-08-21 13:03:40,791 -- INFO -- SSL is set to true
```

Procederemos a realizar una captura de tráfico en uno de los servidores, capturando el tráfico entre el contexto y el RNS mediante un comando como el siguiente:

```
# tcpdump -i eth1 -s 0 port 10000 -w ctx-rns.cap
```

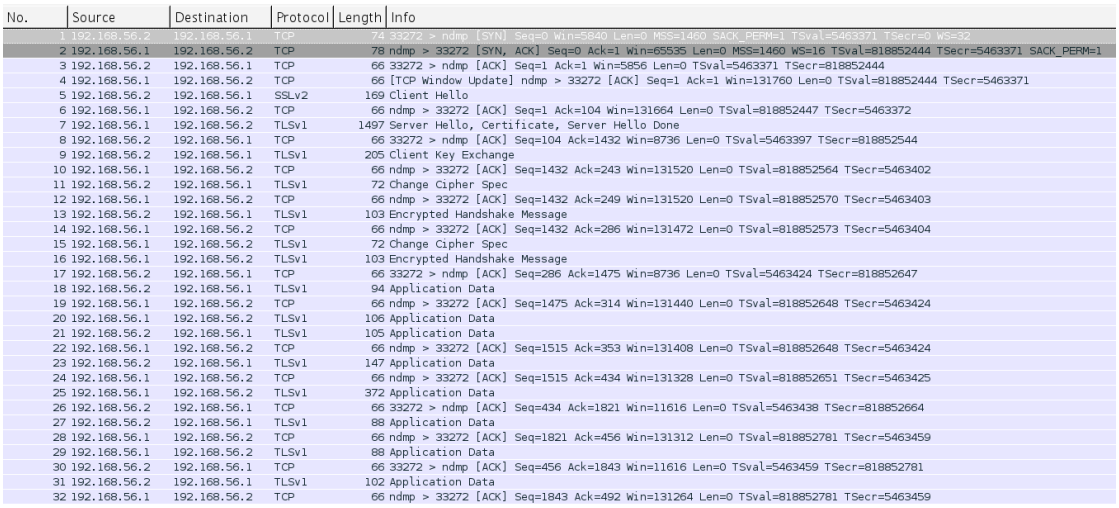
En este momento, arrancaremos el contexto "c1" viendo en su log que se ha activado el mecanismo de SSL y que arranca correctamente (es decir, es capaz de comunicarse con éxito con el RNS mediante RMI-SSL):

```
2013-08-21 13:03:40,538 -- INFO -- Context SSL is true
2013-08-21 13:03:40,538 -- INFO -- Context Authentication is false
2013-08-21 13:03:46,411 -- INFO -- Started context at port 9501!
```

En este momento vamos a analizar el tráfico capturado por nuestro "atacante". En primer lugar observamos cómo Wireshark lo detecta como tráfico SSL en la Figura B.2.

Si realizamos un volcado ASCII del tráfico, de la misma forma que lo hemos realizado con el tráfico no cifrado, observamos lo siguiente:

```
.e....<... ............/..3..2..
..............@.........................R...@..@..'.{M.2=..*.J..=.t.H.A.........M..
    R...i..+...H=QH,l..........
.v. R....9.....r..q.....q....xq...u..............9..6...0...0..........Z)0n*E0.0
..*.H..
.....0W1.0...U....springsCA1.0...U....springs1.0
..U.
..unizar1.0...U....zaragoza1.0...U....ES0..
120910165501Z.
140910165501Z0Q1.0
..U....rns1.0...U....springs1.0
..U.
..unizar1.0...U....zaragoza1.0...U....es0..0
..*.H..
.........0.......m..<v..r..P....w.%.R.Y...p<.."[f;N..z...^...?R.4./.Db....b.:
g.W...g.<v.....bAsS.q.=.t
....&p..|.}<....L...f...
```



| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 1 | 192.168.56.2 | 192.168.56.1 | TCP | 74 | 33272 > ndmp [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=5463371 TSecr=0 WS=32 |
| 2 | 192.168.56.1 | 192.168.56.2 | TCP | 78 | ndmp > 33272 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=16 TSval=818852444 TSecr=5463371 SACK_PERM=1 |
| 3 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33272 > ndmp [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=5463371 TSecr=818852444 |
| 4 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | [TCP Window Update] ndmp > 33272 [ACK] Seq=1 Ack=1 Win=131760 Len=0 TSval=818852444 TSecr=5463371 |
| 5 | 192.168.56.2 | 192.168.56.1 | SSLv2 | 169 | Client Hello |
| 6 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1 Ack=104 Win=131664 Len=0 TSval=818852447 TSecr=5463372 |
| 7 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 1497 | Server Hello, Certificate, Server Hello Done |
| 8 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33272 > ndmp [ACK] Seq=104 Ack=1432 Win=8736 Len=0 TSval=5463397 TSecr=818852544 |
| 9 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 205 | Client Key Exchange |
| 10 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1432 Ack=243 Win=131520 Len=0 TSval=818852564 TSecr=5463402 |
| 11 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 72 | Change Cipher Spec |
| 12 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1432 Ack=249 Win=131520 Len=0 TSval=818852570 TSecr=5463403 |
| 13 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 103 | Encrypted Handshake Message |
| 14 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1432 Ack=286 Win=131472 Len=0 TSval=818852573 TSecr=5463404 |
| 15 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 72 | Change Cipher Spec |
| 16 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 103 | Encrypted Handshake Message |
| 17 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33272 > ndmp [ACK] Seq=286 Ack=1475 Win=8736 Len=0 TSval=5463424 TSecr=818852647 |
| 18 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 94 | Application Data |
| 19 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1475 Ack=314 Win=131440 Len=0 TSval=818852648 TSecr=5463424 |
| 20 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 106 | Application Data |
| 21 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 105 | Application Data |
| 22 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1515 Ack=353 Win=131408 Len=0 TSval=818852648 TSecr=5463424 |
| 23 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 147 | Application Data |
| 24 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1515 Ack=434 Win=131328 Len=0 TSval=818852651 TSecr=5463425 |
| 25 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 372 | Application Data |
| 26 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33272 > ndmp [ACK] Seq=434 Ack=1821 Win=11616 Len=0 TSval=5463438 TSecr=818852664 |
| 27 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 88 | Application Data |
| 28 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1821 Ack=456 Win=131312 Len=0 TSval=818852781 TSecr=5463459 |
| 29 | 192.168.56.1 | 192.168.56.2 | TLSv1 | 88 | Application Data |
| 30 | 192.168.56.2 | 192.168.56.1 | TCP | 66 | 33272 > ndmp [ACK] Seq=456 Ack=1843 Win=11616 Len=0 TSval=5463459 TSecr=818852781 |
| 31 | 192.168.56.2 | 192.168.56.1 | TLSv1 | 102 | Application Data |
| 32 | 192.168.56.1 | 192.168.56.2 | TCP | 66 | ndmp > 33272 [ACK] Seq=1843 Ack=492 Win=131264 Len=0 TSval=818852781 TSecr=5463459 |

Figura B.2: Captura de red tráfico RMI cifrado

```
h.pMJ.C*....
........u0s0...U......6..wM..............0...U.......0.0...U.#..0...M......%00
    ......J...0...U...........0...U.%..0
..+.......0
..*.H..
.........j...J*a..#.....................C.@gW.?g.......1..f.....r...N!U.T.'....yP5.
    A.F....}5.........V...xA....k..NCN.K..L....A.N..t#..0
    .....0...0..........>.~....K0
..*.H..
.....0W1.0...U....springsCA1.0...U....springs1.0
..U.
..unizar1.0...U....zaragoza1.0...U....ES0..
120905183340Z.
220905183025Z0W1.0...U....springsCA1.0...U....springs1.0
..U.
..unizar1.0...U....zaragoza1.0...U....ES0..0
..*.H..
.........0..........P.S............+...
.X.E...5.T...................(..(.^u.."t.D..kE....&..@....n.)c.........../...‘.J4
    ...[....~R.....Y .M<..........c0a0...U......M......%00......J...0...U
    .......0....0...U.#..0...M......%00......J...0...U...........0
..*.H..
.........p...Xz
.U,....,...vu8.(...{a...u]..*t\*.
.Nd..6/...........
5....@}.da...c..}.V..>.
-...!TY..."...:.+..9..bc.n....y..i.Wq.N..C{.................Fv..q..a.S.]i.....
..e.6.?..Z‘.........,.b.4..$S.S..@..x..G1.....W..U+Ft..L\a.8. .‘vA@.@....‘..R...&.
    G..#......2.:F........m.............. .....x.&h.;1..4‘..4...B1a...,8l
    ........... ..1..d.8..}..Xe.......(.......a2‘......S...%x.[6L...J.!..Q1
    .......#.^].(>.....................=.Dq.+........"..K....Y.T.....?.............
    ........LH^...?... dAu..g\.....f.....S..L..M.‘U..........6.5..
-..7..Xq..X69.,.no..r.....->_=.c...gy..D....n...-..j....
O....Xb.x....$....H..T...u.z.{.[....
&............;..i‘P............f.....C...H..bA..RC.....?,..M‘.?.?w....8....IG
    .9%...fy...n.MI...2..C/’].S...YP.s9Iu6................^.E.h.3.g.B@{.iU..F
    .,..........D6\2>. 6...~...EI8.I....S,..]2\..1.
```

```
..a.....
?.V.......b.C.kh...t)C.....................7.Qw.O.......
....T.>..x..d.P........H..}.9D..N....‘y.....,hz..:..
```

En el que se puede observar a simple vista cómo la comunicación ha cambiado completamente comparada con la anterior comunicación sin cifrar, pasando a ser completamente binaria siguiendo el protocolo SSL.

De la misma forma que se ha demostrado que se protegen las comunicaciones entre un contexto y el RNS, se protegen también el resto de comunicaciones RMI de la plataforma. Especialmente importantes son las que se refieren a la comunicación entre agentes y al movimiento de agentes entre contextos.

## B.4 Acceso a la plataforma de un contexto cuyo certificado no está aceptado por el RNS

En esta prueba un contexto “maligno” con un certificado no aceptado por el RNS intentará acceder a la plataforma. Gracias al mecanismo de seguridad de autenticación a nivel de plataforma se evitará su acceso, protegiendo a la plataforma de ataques de contextos no permitidos.

El escenario de pruebas comprende:

- Un servidor ejecutando un RNS con el mecanismo de autenticación activo mediante la siguiente configuración:

  ```
  [security]

  authentication = true
  keystore = ../etc/rns.keystore
  keystorePass = springs
  ```

- Un servidor ejecutando un contexto llamado “c1-maligno” con un certificado que no se encuentra en el *keystore* de RNS.

Al arrancar el RNS observamos cómo se activan los mecanismos de autenticación a nivel de plataforma leyendo en el *log* lo siguiente:

```
2013-08-22 12:55:18,719 -- INFO -- Started RNS at port 10000!
2013-08-22 12:55:18,719 -- INFO -- SSL is set to false
2013-08-22 12:55:18,719 -- INFO -- Platform Authentication is set to true
2013-08-22 12:55:18,719 -- INFO -- Platform Encryption is set to false
```

Cuando arrancamos el contexto “c1-maligno” observamos el siguiente mensaje de error en su *log*:

```
2013-08-22 12:55:19,543 -- INFO -- Context SSL is false
2013-08-22 12:55:19,543 -- INFO -- Context Authentication is false
2013-08-22 12:55:24,646 -- ERROR -- Error starting context springs.
    context.ContextStartingException: c1-maligno: Error communicating with
     RNS java.rmi.ServerException: RemoteException occurred in server
    thread; nested exception is:
        java.rmi.RemoteException: The context certificate is not
    authorized
```

Indicando que el certificado del contexto no ha sido autorizado.

En el RNS vemos el siguiente mensaje de error:

```
2013-08-22 12:55:24,609 -- DEBUG -- Adding context c1-maligno with URL
    192.168.56.2:9501
2013-08-22 12:55:24,609 -- ERROR -- The context c1-maligno certificate is
     not authorized
```

Para que conste a nivel de RNS el intento de ataque a la plataforma.

## B.5 Creación de un agente por parte de un contexto que no tiene permisos para ello

En esta prueba se demostrará cómo se puede evitar que determinado contexto sea capaz de crear agentes mediante los mecanismos de seguridad de autenticación y autorización a nivel de plataforma.

Para realizar dicha prueba, contamos con:

- Un servidor ejecutando un RNS con el mecanismo de autenticación activo mediante la siguiente configuración:

  ```
  [security]

  authentication = true
  keystore = ../etc/rns.keystore
  keystorePass = springs
  ```

  Nótese que no existen líneas de configuración de permisos. Esto indica que no hay ningún permiso otorgado.

- Un servidor ejecutando un contexto llamado “Test” que intenta crear un agente mediante el siguiente código fuente:

  ```
  try {
      Context_RMIImpl.create(LOCAL_CONTEXT_NAME, PORT_TEST,
     PORT_CLASS_SERVER,
  ```

```
        RNS_ADDRESS, false, false, ks);
    Context context = Context_RMIImpl.instance();
    logger.info("Started context in port " + PORT_TEST + "!");

    logger.info("Creating MovingAgentExample...");
    MovingAgentExample ag = new MovingAgentExample();
    context.createAgent(ag, "MovingAgentExample");
    logger.info("Created MovingAgentExample!");
} catch (Exception e) {
    logger.error("Problem with context " + e.toString());
    System.exit(-1);
}
```

Una vez que el RNS está arrancado se lanza el contexto "Test" en cuyo *log* se puede ver el siguiente mensaje de error:

```
2013-08-22 13:28:48,710 -- INFO -- Started context in port 12000!
2013-08-22 13:28:48,710 -- INFO -- Creating MovingAgentExample...
2013-08-22 13:28:48,717 -- ERROR -- Problem with context springs.security
    .SecurityException: Test: Context not allowed to createAgent method
```

Terminando su ejecución. En el *log* del RNS se observa lo siguiente:

```
2013-08-22 13:28:48,578 -- INFO -- Added context Test with URL
    192.168.56.2:12000
2013-08-22 13:28:48,657 -- INFO -- The context Test has been authorized
    by the RNS
2013-08-22 13:28:48,657 -- INFO -- The context Test is allowed to
    PlatformPermission
2013-08-22 13:28:48,671 -- INFO -- The context Test has been authorized
    by the RNS
2013-08-22 13:28:48,671 -- ERROR -- The context Test is not allowed to
    AgentPermission
```

En el que se puede ver cómo el contexto "Test" se crea y se añade a la plataforma correctamente pero, cuando intenta crear un agente, el contexto no está autorizado y se le devuelve un error.

## B.6 Movimiento de un agente por parte de un contexto que no tiene permisos para ello

En esta prueba se demostrará cómo se puede evitar que determinados contextos sean capaces de crear agentes que se puedan mover, mediante los mecanismos de seguridad de autenticación y autorización a nivel de plataforma.

Para realizar dicha prueba, contamos con:

- Un servidor ejecutando un RNS con el mecanismo de autenticación activo mediante la siguiente configuración:

```
[security]

authentication = true
authentication.agentPermission = Test
keystore = ../etc/rns.keystore
keystorePass = springs
```

  Nótese que existe una linea de configuración para permitir la creación de agentes al contexto "Test", pero no se permite nada más.

- Un servidor ejecutando un contexto llamado "c1".

- Un servidor ejecutando un contexto llamado "Test" que crea un agente en cuyo código se realiza la petición de movimiento al contexto "c1" de la forma:

```
public void main()
{
    try {
        travel1();
    } catch (AgentmovementException e) {
        logger.error(e.toString());
        System.exit(-1);
    }
}

public void travel1() throws AgentMovementException
{
    logger.info(getNameWithContext() + ": I'm going to move to C1");
    logger.info(getNameWithContext() + ": my home is at " +
    getHomeAddress());
    moveTo("c1", "travel2");
    return;
}
```

En primer lugar lanzamos tanto el RNS como el contexto "c1". Después ejecutamos el contexto "Test" que crea el agente "MovingAgentExample" correctamente pero, posteriormente, da un error. Esta secuencia de eventos se ve en el *log* de la forma:

```
2013-08-22 14:28:02,194 -- INFO -- Started context in port 12000!
2013-08-22 14:28:02,194 -- INFO -- Creating MovingAgentExample...
2013-08-22 14:28:02,288 -- INFO -- Registered agent MovingAgentExample at
    the RNS! (2 locationServers assigned)
2013-08-22 14:28:02,302 -- INFO -- Registered agent MovingAgentExample!
2013-08-22 14:28:02,302 -- INFO -- Created agent MovingAgentExample in
   context manager!
2013-08-22 14:28:02,302 -- INFO -- Created MovingAgentExample!
2013-08-22 14:28:02,381 -- INFO -- MovingAgentExample@Test: I'm going to
   move to C1
```

```
2013-08-22 14:28:02,381 -- INFO -- MovingAgentExample@Test: my home is at
    rmi://192.168.56.3:12000
2013-08-22 14:28:02,387 -- ERROR -- springs.agent.AgentMovementException:
    Test: Context not allowed to access moveAgent method
```

Si observamos el *log* del RNS vemos cómo también se notifica la denegación de permisos de movimiento al agente creado por el contexto "Test".

```
2013-08-22 14:28:02,204 -- INFO -- Agent MovingAgentExample added to Test
2013-08-22 14:28:02,304 -- INFO -- The context Test has been authorized
   by the RNS
2013-08-22 14:28:02,304 -- DEBUG -- The context Test requests permission
   for MoveAgentPermission
2013-08-22 14:28:02,304 -- ERROR -- The context Test is not allowed to
   MoveAgentPermission
```

De esta forma se puede evitar que un contexto cree agentes que se puedan mover a otros contextos y, potencialmente, atacarlos.

## B.7 Agente realizando llamadas creado en un contexto que no tiene permisos para ello

En esta prueba se demostrará cómo se puede evitar, mediante el mecanismo de autenticación y autorización a nivel de plataforma, que un contexto determinado sea capaz de crear agentes que puedan realizar llamadas a otros agentes.

El escenario de la prueba es el siguiente:

- Un servidor ejecutando un RNS con el mecanismo de autenticación activo mediante la siguiente configuración:

  ```
  [security]

  authentication = true
  authentication.agentPermission = Test, Test2
  keystore = ../etc/rns.keystore
  keystorePass = springs
  ```

  Nótese que existe una linea de configuración para permitir la creación de agentes a los contextos "Test" y "Test", pero no se permite nada más.

- Un servidor ejecutando un contexto llamado "Test" que crea un agente llamado "ReceivingAgentExample". Este agente lo único que hace es esperar en un bucle infinito a que alguien invoque su método `hello()` que hace que el contexto imprima un mensaje de texto.

- Un servidor ejecutando un contexto llamado “Test” va a crear un agente en cuyo código se realiza una llamada al método `hello()` del agente “ReceivingAgentExample”, cuyo código relevante es como el siguiente:

```
callAgentMethod("ReceivingAgentExample", "hello");
```

En primer lugar lanzamos tanto el RNS como el contexto “Test”, esperando a que se cree el agente “ReceivingAgentExample”. Después ejecutamos el contexto “Test2” que crea el agente “CallingAgentExample” correctamente pero, posteriormente, da un error al invocar al método `callAgentMethod()`. Esta secuencia de eventos se ve en el *log* de la forma:

```
2013-08-22 18:16:20,792 -- INFO -- Started context in port 12001!
2013-08-22 18:16:20,792 -- INFO -- Creating CallingAgentExample...
2013-08-22 18:16:20,895 -- INFO -- Registered agent CallingAgentExample!
2013-08-22 18:16:20,897 -- INFO -- Created CallingAgentExample!
2013-08-22 18:16:30,909 -- INFO -- CallingAgentExample: calling hello...
2013-08-22 18:16:30,915 -- ERROR -- springs.security.SecurityException:
    Test2: Context not allowed to access callAgentMethod on
    ReceivingAgentExample
```

Si observamos el *log* del RNS vemos cómo también se notifica la denegación de permisos de llamada al agente creado por el contexto “Test2”.

```
2013-08-22 18:16:30,805 -- DEBUG -- The context Test2 requests permission
     for CallAgentPermission
2013-08-22 18:16:30,805 -- ERROR -- The context Test2 is not allowed to
    CallAgentPermission
```

De esta forma se puede evitar que un contexto cree agentes que se puedan realizar ataques mediante la invocación a métodos de otros agentes.

## B.8 Contexto atacando otro contexto simulando ser el RNS

En esta prueba se demostrará cómo se puede evitar, mediante el mecanismo de autenticación y autorización a nivel de plataforma, que un contexto determinado sea capaz invocar métodos permitidos solo al RNS de otro contexto.

El escenario de la prueba es el siguiente:

- Un servidor ejecutando un RNS con el mecanismo de autenticación activo mediante la siguiente configuración:

```
[security]

authentication = true
keystore = ../etc/rns.keystore
keystorePass = springs
```

- Un servidor ejecutando el contexto “c1”, que será la víctima del ataque.
- Un servidor ejecutando el contexto “Test” que ejecutará el siguiente código:

```
ContextAddress ca = new ContextAddress("c1", "rmi://springs-1:9501",
    false);

byte[] challenge = null;
String operationId = "BadGuy";

ReferenceServer.getContextManager(ca).addContext(ca, challenge,
   operationId);
```

  En dicho código se observa cómo se realiza una búsqueda en el sistema por el `ContextManager` del contexto “c1” y se invoca al método `addContext()` de dicho `ContextManager`.

Para realizar la prueba lanzamos tanto el RNS como el contexto “c1”. Acto seguido ejecutamos el contexto “Test’, en cuyo *log* vemos lo siguiente:

```
013-08-22 19:29:55,668 -- INFO -- Started context in port 12000!
2013-08-22 19:29:55,668 -- DEBUG -- Contacting remote agent
   ContextManager at rmi://springs-1:9501
2013-08-22 19:29:55,689 -- ERROR -- Problem with context java.rmi.
   ServerException: RemoteException occurred in server thread; nested
   exception is:
       java.rmi.RemoteException: springs.security.SecurityException: c1:
    OTP not allowed by the RNS
```

En él se nos dice que el OTP utilizado no ha sido autorizado por el RNS, es decir, que el RNS no ha generado el *challenge* referente al OTP enviado.

En el *log* del contexto “c1” vemos:

```
2013-08-22 19:29:55,664 -- ERROR -- OTP is not allowed by RNS
```

Y en el *log* del RNS también se nos avisa del mal uso del OTP:

```
2013-08-22 19:29:55,575 -- DEBUG -- Verifying OTP for c1-BadGuy
2013-08-22 19:29:55,575 -- ERROR -- The context c1 has not been granted
   the OTP
```

Como se ha demostrado, no es posible acceder a métodos reservados del `ContextManager` de un contexto a no ser que se hayan obtenido las credenciales necesarias previamente.

## B.9 Uso de autenticación de contexto con la autenticación de plataforma desactivada

En esta prueba veremos cómo la autenticación y autorización a nivel de contexto se desactivan si la autenticación y autorización a nivel de plataforma lo están. Esto se hace así ya que para autenticar a un agente por parte del contexto, es necesario tener el mecanismo de autenticación a nivel de plataforma y, sin éste, no podríamos definir si un agente es quien realmente dice que es.

El escenario de la prueba es el siguiente:

- Un servidor ejecutando un RNS con el mecanismo de autenticación desactivado.
- Un servidor ejecutando un contexto "c1", con el mecanismo de autenticación a nivel de contexto activado. El script de arranque del contexto es el siguiente:

```
export CLASSPATH=../../../lib/log4j-1.2.15.jar:../../../jar/springs.
    jar:../classes
export PATH=$JAVA_HOME:$PATH
java -Djava.security.policy=../etc/security.policy  -Djava.security.
    auth.login.config=../etc/jaas.config springs.context.
    ContextLauncher \$*
```

Es decir, utiliza los ficheros de configuración `jaas.config`

```
Agent
{
    springs.security.AgentLoginModule required;
};
contexts
{
    springs.security.ContextLoginModule required;
};
```

que define los *plugins* de autenticación que se utilizan en SPRINGS y el fichero `security.policy`:

```
/**
 * A minimal set of permissions needed for the stock server to run.
 */

grant {
    permission javax.security.auth.AuthPermission "
    createLoginContext";
    permission javax.security.auth.AuthPermission "doAs";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
    permission javax.security.auth.AuthPermission "modifyPrincipals
    ";
    permission javax.security.auth.AuthPermission "getSubject";
    permission javax.security.auth.AuthPermission "
    createLoginContext.Agent";
```

```
    permission java.net.SocketPermission "*", "accept, connect,
    resolve";
};

grant Principal springs.security.AgentPrincipal "Test" {
};


grant Principal springs.security.ContextPrincipal "context" {
    permission java.security.AllPermission "", "";
};
```

que define qué permisos se encuentran autorizados en el contexto. Por defecto se deben definir una serie de permisos para que el propio mecanismo de autenticación funcione y después, se pueden definir secciones para limitar el acceso a los recursos del servidor por parte del propio contexto y por parte de los agentes que viajen a él. En esta configuración hemos definido que los agentes que hayan sido creados en el contexto "Test" no tienen ningún permiso adicional.

El fichero de configuración del contexto contiene lo siguiente:

```
[security]

authentication = true
keystore = ../etc/c1.keystore
keystorePass = springs
```

- Un servidor con un contexto llamado "Test" que ejecuta un agente llamado "MovingAgentExample" que viaja al contexto "c1", intenta leer el fichero `/tmp/secreto` y, después, volverá al contexto "Test".

En la prueba lanzamos el RNS y el contexto "c1". En el *log* del contexto "c1" vemos el siguiente mensaje de error indicando que la autenticación a nivel de plataforma está desactivada, con lo que la autenticación y autorización a nivel de contexto también lo estará:

```
2013-08-23 11:01:42,927 -- ERROR -- Disabling ContextAuthentication as we
    are not able to authenticate with the RNS
```

En este momento lanzamos el contexto "Test", que crea el agente "MovingAgentExample" que, en el contexto "c1", deja lo siguiente en el *log*:

```
2013-08-23 11:01:48,146 -- INFO -- MovingAgentExample@c1: postArrival
   (11:01:48:146)
2013-08-23 11:01:48,146 -- INFO -- MovingAgentExample@c1: I've arrived in
    C1
2013-08-23 11:01:48,147 -- INFO -- MovingAgentExample@c1: my home is at
   rmi://192.168.56.3:12000
2013-08-23 11:01:48,147 -- INFO -- MovingAgentExample@c1: I am going to
   read /tmp/secreto
```

```
2013-08-23 11:01:48,147 -- INFO -- Going to read file
2013-08-23 11:01:48,147 -- INFO -- Esto es un secreto!!!
2013-08-23 11:01:48,147 -- INFO -- MovingAgentExample@c1: I'm going to
    move home
2013-08-23 11:01:48,149 -- INFO -- MovingAgentExample@c1: preDeparture
    from rmi://192.168.56.2:9501 to rmi://192.168.56.3:12000(11:01:48:149)
2013-08-23 11:01:48,158 -- INFO -- MovingAgentExample@c1: postDeparture
    (11:01:48:158)
```

Es decir, observamos cómo el agente sí que ha tenido acceso al fichero `/tmp/secreto` del contexto "c1" y ha sido capaz de leerlo.

## B.10 Acceso a recursos no permitidos por la autorización a nivel de contexto

En esta prueba veremos cómo el mecanismo de autenticación y autorización a nivel de contexto es capaz de evitar el acceso a los recursos de un contexto por parte de un agente. Con estas medidas logramos proteger a los contextos de ataques que puedan llevar a cabo agentes que viajen a ellos.

El escenario de la prueba es el mismo que el de la Sección B.9. La única diferencia es que, en este caso, sí que tendremos activado el mecanismo de autenticación y autorización a nivel de plataforma, con la siguiente configuración del RNS:

```
[security]

authentication = true
authentication.agentPermission = Test
authentication.moveAgentPermission = Test
keystore = ../etc/rns.keystore
keystorePass = springs
```

Ejecutamos tanto el RNS como el contexto "c1", no recibiendo en esta ocasión el mensaje de error diciendo que la autenticación a nivel de plataforma no está activa:

```
2013-08-23 12:06:01,073 -- INFO -- Context SSL is false
2013-08-23 12:06:01,073 -- INFO -- Context Authentication is true
2013-08-23 12:06:06,281 -- INFO -- Started context at port 9501!
```

A continuación ejecutamos el contexto "Test" que crea el agente "MovingAgentExample". Cuando éste llega al contexto "c1" procede a intentar leer el fichero `/tmp/secreto` protegido, obteniéndose el siguiente *log*:

```
2013-08-23 12:06:15,924 -- INFO -- Agent Subject: Principal: (
    AgentPrincipal: name=Test) logged into the system
```

```
2013-08-23 12:06:15,925 -- INFO -- MovingAgentExample@c1: postArrival
    (12:06:15:925)
2013-08-23 12:06:15,925 -- INFO -- MovingAgentExample@c1: I've arrived in
     C1
2013-08-23 12:06:15,925 -- INFO -- MovingAgentExample@c1: my home is at
    rmi://192.168.56.3:12000
2013-08-23 12:06:15,925 -- INFO -- MovingAgentExample@c1: I am going to
    read /tmp/secreto
2013-08-23 12:06:15,925 -- INFO -- Going to read file
2013-08-23 12:06:15,926 -- ERROR -- Problem reading file! java.security.
    AccessControlException: access denied (java.io.FilePermission /tmp/
    secreto read)
2013-08-23 12:06:15,926 -- INFO -- MovingAgentExample@c1: I'm going to
    move home
2013-08-23 12:06:15,931 -- INFO -- MovingAgentExample@c1: preDeparture
    from rmi://192.168.56.2:9501 to rmi://192.168.56.3:12000(12:06:15:930)
2013-08-23 12:06:15,955 -- INFO -- MovingAgentExample@c1: postDeparture
    (12:06:15:955)
```

En dicho *log* observamos como, en primer lugar, el agente realiza el *login* en el contexto, tras ser aceptado procede a su ejecución. Durante la ejecución, éste intenta acceder al fichero `/tmp/secreto` pero su acceso es impedido por el contexto.

## B.11 Uso de cifrado cuando está desactivado en la plataforma

En esta prueba veremos cómo no es posible utilizar las funciones de cifrado de la plataforma si éste está desactivado a nivel de RNS.

Para ello, el escenario requerido es el siguiente:

- Un servidor ejecutando el RNS con la opción de cifrado global desactivada de la forma:

  ```
  [security]

  encryption = false
  keystore = ../etc/rns.keystore
  keystorePass = springs
  ```

- Un servidor ejecutando un contexto llamado "c1".
- Un servidor ejecutando un contexto llamado "Test" en el que se crea un agente llamado "MovingAgentExample" cuyo código contiene lo siguiente:

  ```
  private String fraseSecreta = "Esta es una frase secreta";
  private EncryptedData secreto;
  ```

```
secreto = encrypt(fraseSecreta);
```

Para realizar la prueba ejecutaremos el RNS, donde en su *log* podemos observar la siguiente línea indicando que el cifrado a nivel de plataforma está desactivado:

```
2013-08-23 15:56:30,387 -- INFO -- Platform Encryption is set to false
```

Acto seguido, lanzamos los contextos "c1" y "Test'. En este último, vemos como al crearse el agente "MovingAgentExample" y utilizar la función de cifrado previamente descrita, se devuelve una excepción:

```
2013-08-23 15:59:27,734 -- ERROR -- springs.security.SecurityException:
    Test: Encryption disabled on region
```

De esta forma se puede evitar el uso de las funciones de cifrado y validación de contenidos en la plataforma.

## B.12 Uso de cifrado y verificación de integridad de datos

En esta prueba comprobaremos que los métodos de cifrado y verificación de integridad de datos funcionan correctamente.

Para ello, el escenario requerido es el siguiente:

- Un servidor ejecutando el RNS con la opción de cifrado global activada de la forma:

  ```
  [security]

  encryption = true
  keystore = ../etc/rns.keystore
  keystorePass = springs
  ```

- Un servidor ejecutando un contexto llamado "c1".

- Un servidor ejecutando un contexto llamado "Test" en el que se crea un agente llamado "MovingAgentExample", cuyo código hace que el agente viaje al contexto "c1", donde cifra un dato:

  ```
  secreto = encrypt(fraseSecreta);
  ```

  Después realiza un viaje al contexto donde fue creado para poder descifrar el secreto de la forma:

  ```
  String datosEnClaro = (String) decrypt(secreto);
  ```

A su vez comprueba que los datos no han sido alterados en el tránsito con el siguiente código:

```
Boolean verify = verifySignature(datosEnClaro, secreto);
```

Para realizar la prueba ejecutaremos el RNS, donde en su *log* podemos observar la siguiente línea indicando que el cifrado a nivel de plataforma está activado:

```
2013-08-23 16:38:13,717 -- INFO -- Platform Encryption is set to true
```

Acto seguido, lanzamos los contextos "c1" y "Test'. En los logs de ambos vamos viendo el comportamiento programado en el código del agente "MovingAgentExample". En primer lugar el agente se mueve al contexto "c1" donde realiza el cifrado de los datos:

```
2013-08-23 18:34:04,063 -- INFO -- MovingAgentExample@c1 Secreto cifrado,
    firmado por c1
```

Después el agente viaja a su contexto original donde descifra e imprime los datos:

```
2013-08-23 18:34:04,083 -- INFO -- MovingAgentExample@Test Vamos a
   intentar leer los datos cifrados
2013-08-23 18:34:04,245 -- INFO -- MovingAgentExample@Test los datos en
   claro son: Esta es una frase secreta
```

Una vez realizadas dichas acciones verifica la integridad de los datos cifrados:

```
2013-08-23 18:34:04,253 -- INFO -- MovingAgentExample@Test los datos son
   los originales true
```

De esta forma hemos comprobado cómo se realiza el cifrado, el descifrado y la verificación de integridad de datos en la plataforma. En caso de que una estructura cifrada fuera accesible por parte de un agente o de un contexto "maligno" estos solo podrían ser descifrados por un agente que hubiera sido creado en el mismo contexto en el que se creó el agente que sufrió el robo de datos.

## B.13 Intento de descifrado de datos en un contexto incorrecto

En esta prueba veremos como no es posible descifrar los datos de un agente en cualquier contexto. Esto solo se puede realizar en el contexto que creó al agente.

Para ello utilizaremos el escenario de pruebas de la Sección B.12. La única modificación es el código del agente `MovingAgentExample` en el que, en vez de realizar el descifrado de los datos en el contexto "Test", éste se va a intentar realizar en el contexto "c1".

Ejecutando la prueba de la misma forma que lo hicimos en la Sección B.12 con el nuevo agente, ahora obtenemos el siguiente mensaje de error en el *log* del contexto "c1":

```
2013-08-23 18:53:32,853 -- ERROR -- Exception: springs.security.
    SecurityException: c1: Agent is not allowed to decrypt in this context
```

Con esta prueba hemos podido verificar que un dato cifrado para un agente no es posible descifrarlo en cualquier contexto, sino que es necesario que se haga en el contexto que creó el agente.

## B.14 Simulación de robo de dato cifrado e intento de descifrarlo

En esta prueba vamos a simular el robo de un dato cifrado de un agente y vamos a intentar descifrarlo de alguna forma.

Para ello vamos a basarnos en el escenario de pruebas de la Sección B.12, pero vamos a modificar el código del agente "MovingAgentExample" haciendo que el cifrado de los datos los realice para un agente de otro contexto que no es el suyo de la forma:

```
secreto = encrypt(fraseSecreta, "C1");
```

También pondremos código para descifrar el dato cifrado tanto en el viaje al contexto "c1" como en su contexto original.

Una vez ejecutados tanto el RNS como los contextos "c1" y "Test" vamos a analizar los mensajes del *log* de los contextos. En primer lugar vemos que el cifrado del dato se realiza correctamente:

```
2013-08-23 19:13:54,854 -- INFO -- MovingAgentExample@c1 Secreto cifrado,
     firmado por c1
```

A continuación vemos el intento de descifrado del dato en el contexto "c1":

```
2013-08-23 19:13:54,855 -- ERROR -- Exception: springs.security.
    SecurityException: c1: Agent is not allowed to decrypt in this context
```

El cual, como hemos visto previamente, devuelve un mensaje de error indicando que no se puede utilizar el método de descifrado de un contexto que no sea el que ha creado el agente.

Para finalizar, tras el movimiento del agente a su contexto original, éste es el resultado de descifrado del dato en dicho contexto:

```
2013-08-23 19:13:55,066 -- ERROR -- Exception: springs.security.
    SecurityException: Test: Not possible to decrypt! javax.crypto.
    BadPaddingException: Data must start with zero
```

Indicando que, como estaba previsto, no es posible descifrar un dato cifrado con la clave pública de un contexto en otro contexto, haciendo imposible descifrar un dato cifrado tras robárselo a un agente.

## B.15 Ataque de uso masivo de memoria de un contexto

A continuación vamos a detallar un ataque para el que la plataforma actual, debido a ejecutarse sobre la JVM estándar, no tiene forma de protegerse a día de hoy. El ataque consiste en el uso masivo de toda la memoria disponible de un contexto haciendo que dicho contexto aborte su ejecución.

El escenario de la prueba es el siguiente:

- Un servidor ejecutando el RNS.
- Un servidor ejecutando un contexto llamado "c1".
- Un servidor ejecutando un contexto llamado "Test" que crea un agente con el siguiente código:

```
public class MovingAgentExample extends SpringsAgent_RMIImpl
{
   public static Log logger = new Log();
   public static List<byte[]> list = new ArrayList<byte[]>();

   public MovingAgentExample() throws RemoteException
   {
   }

   public void main()
   {
       try {
        travel1();
       } catch (Exception e) {
           logger.error(e.toString());
           System.exit(-1);
       }
   }

   public void travel1() throws Exception
   {
      logger.info(getNameWithContext() + ": I'm going to move to C1
    ");
      logger.info(getNameWithContext() + ": my home is at " +
    getHomeAddress());
      moveTo("c1", "travel2");
      return;
   }
```

```
   public void travel2() throws AgentMovementException
   {
      logger.info(getNameWithContext() + ": I've arrived in C1");

      while (true) {
          BigInteger bi1 = new BigInteger("10000000000000000");
          byte[] b1 = bi1.toByteArray();
          list.add(b1);
      }
   }
}
```

En dicho código se ve cómo el agente, en primer lugar, viaja al contexto "c1" en el que entra en un bucle infinito que va añadiendo elementos a una lista de *arrays* de *bytes*.

Para realizar la prueba, en primer lugar, ejecutamos tanto el RNS como el contexto "c1". A continuación lanzamos el contexto "Test" que crea el agente "MovingAgentExample". Tras pocos segundos de ejecución de dicho agente en el contexto "c1" obtenemos el siguiente mensaje:

```
java.lang.OutOfMemoryError: Java heap space
        at java.lang.AbstractStringBuilder.<init>(AbstractStringBuilder.
    java:45)
        at java.lang.StringBuilder.<init>(StringBuilder.java:68)
        at springs.util.SpringsException.getMessage(SpringsException.java
    :117)
        at java.lang.Throwable.getLocalizedMessage(Throwable.java:267)
        at java.lang.Throwable.toString(Throwable.java:343)
        at springs.context.threads.AgentExecutor.run(AgentExecutor.java
    :149)
```

Indicando que el contexto "c1" ha dejado de funcionar debido a la falta de memoria. Este ataque junto con ataques de uso de CPU del contexto o invasión masiva de agentes en un contexto no se pueden defender con la plataforma actual, siendo recomendable estudiar qué métodos o qué JVM se podría utilizar en la plataforma para defenderse de ellos.

# Apéndice C

# Manual de Usuario

En el presente capítulo vamos a detallar los procedimientos necesarios para la correcta utilización de los mecanismos de seguridad de la plataforma SPRINGS.

En primer lugar describiremos los procedimientos de administración de la plataforma pasando posteriormente a comentar las modificaciones más importantes en el uso del API de SPRINGS para hacer uso de los mecanismos de seguridad de una manera programática.

## C.1 Administración de la plataforma

Se ha dotado a la plataforma de varios mecanismos para facilitar su administración. Principalmente se ha desarrollado una gestión de la configuración del RNS y de los contextos a través de ficheros de configuración. Además se ha mejorado el *logging* de la plataforma utilizando un mecanismo estándar como es el ofrecido por Log4j [85] para todos los elementos.

Vamos a detallar los mecanismos de gestión de certificados, los procedimientos de administración del RNS para, después, describir los de los contextos.

### C.1.1 Gestión de certificados digitales y *keystores*

Como ya describimos en la Sección 3.1, todos los elementos estructurales de la plataforma SPRINGS están identificados unívocamente por un certificado digital. En Java, la forma estándar de almacenar los certificados digitales es utilizar los llamados *keystores* [63]. A continuación vamos a detallar los procedimientos más utilizados para la gestión de certificados digitales y *keystores* en la plataforma.

### Importación de clave pública de una CA en el sistema

Es necesario que la JVM confíe en la CA que ha expedido los certificados utilizados en la plataforma. Para ello hay que importar la clave pública de la CA en el fichero `cacerts` de la máquina, cuya clave por defecto es "changeit", mediante la herramienta `keytool` de la forma:

```
# keytool -import -alias nombreCA -keystore ruta_completa/cacerts -v -
    file clavePublica.pem
```

Por ejemplo, para realizar dicha operación bajo el sistema operativo Mac OS X importando la clave pública de la CA que hemos creado para las pruebas de SPRINGS, el comando que debemos ejecutar es el siguiente:

```
# keytool -import -alias springsCA -keystore /System/Library/Java/Support
    /CoreDeploy.bundle/Contents/Home/lib/security/cacerts -v -file
    Downloads/springsCA.cacert.pem
```

Ante la pregunta de si queremos confiar en la clave pública, deberemos decir que sí y, de esta forma, nuestra JVM ya confiará en la CA.

### Creación de *keystore* para RNS

El procedimiento para crear un *keystore* para el RNS, dado un certificado digital en formato PKCS#12 [46] es el siguiente:

```
$ keytool -importkeystore -srckeystore rns.p12 -destkeystore rns.keystore
      -srcstoretype PKCS12 -deststoretype JCEKS -srcalias rns -destalias
    authkey
```

Es importante que el alias dentro del *keystore* del certificado sea `auth` ya que es el que el RNS leerá para cargarlo como certificado propio.

### Creación de *keystore* para contexto

El procedimiento para crear un *keystore* para un contexto, dado un certificado digital en formato p12 [46] es el siguiente:

```
$ keytool -importkeystore -srckeystore nombre.p12 -destkeystore nombre.
    keystore  -srcstoretype PKCS12 -deststoretype JCEKS -srcalias nombre -
    destalias nombre
```

Es importante que el alias dentro del *keystore* del certificado sea el mismo que el nombre del contexto ya que es el que el contexto leerá para cargarlo como certificado propio.

#### Importación de la clave pública de un contexto en el RNS

Para importar las claves públicas de los contextos en el RNS para su uso en la autenticación a nivel de plataforma y/o en el cifrado, es necesario seguir el siguiente procedimiento.

1. En primer lugar, dado un certificado en formato PKCS#12, debemos extraer su clave pública en formato DER (Distinguished Encoding Rules) utilizando la herramienta `openssl` [86] de la forma:

```
$ openssl pkcs12 -in nombre.p12 -out nombre.pem -nodes
$ openssl x509 -outform der -in nombre.pem -out nombre.der
```

2. A continuación procedemos a la importación de la clave pública en el *keystore* del RNS de la forma:

```
$ keytool -keystore rns.keystore -storetype JCEKS -import -file
    nombre.der -alias nombre
```

   Como hemos explicado previamente, es importante que el alias de la clave pública en el *keystore* se corresponda con el nombre del contexto.

### C.1.2 Procedimientos de administración del RNS

Para realizar una correcta gestión de los mecanismos de seguridad del RNS vamos a pasar a detallar sus principales procedimientos de administración.

#### *Script* de arranque

Con el objetivo de facilitar el arranque del RNS se ofrece un *script* que se encarga de configurar todas las variables necesarias para la correcta ejecución del proceso. Dicho *script* se llama `RNSLauncher`. El contenido del mismo es el siguiente:

```
export CLASSPATH=lib/log4j-1.2.15.jar:jar/springs.jar:classes
export PATH=$JAVA_HOME:$PATH
java springs.rns.RegionNameServerLauncher $*
```

En él podemos ver cómo podemos ajustar las variables `CLASSPATH` y `PATH` para apuntarlas a las rutas correctas donde tengamos tanto el fichero JAR con el código de la plataforma como otras clases de agentes necesarias.

El *script* requiere un parámetro indicando la ruta del fichero de configuración del RNS:

```
# bin/RNSLauncher
Mandatory arguments missing:
 -c configuration file
```

Una vez introducido el parámetro indicando la ruta del fichero de configuración, el RNS lo leerá y se ejecutará.

Si arranca el RNS con éxito podremos ver una línea en el *log* (si está activado) como la siguiente:

```
2013-08-27 09:24:24,169 -- INFO -- Started RNS at port 10000!
```

### Fichero de configuración

Se ha incluido en la plataforma un sistema de ficheros de configuración basado en propiedades de Java. El fichero se encuentra dividido en secciones en las que se pueden configurar diferentes propiedades de la forma:

```
propiedad = valor
```

A continuación vamos a pasar a describir las secciones del fichero de configuración del RNS con todas las propiedades que podemos configurar.

### Sección "general"

En esta sección se pueden configurar los aspectos generales del RNS. Más detalladamente podemos configurar la siguiente propiedad que indica el puerto en el que el RNS escuchará peticiones:

```
port = integer
```

### Sección "log"

En esta sección se configuran los aspectos referentes al *log* de la plataforma. Como hemos comentado, el log de la plataforma se basa en el paquete Log4j para el que se pueden encontrar todas las opciones de configuración en [87]. El siguiente es un ejemplo de configuración del *log*:

```
[log]

log4j.rootLogger = INFO, A1
log4j.appender.A1.layout = org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern = %d -- %p -- %m%n
```

```
log4j.appender.A1 = org.apache.log4j.RollingFileAppender
log4j.appender.A1.File = log/rns.log
log4j.appender.A1.MaxBackupIndex = 1
```

En dicho ejemplo estamos configurando el *logging* del RNS para que se envíe todo lo que tenga prioridad "INFO" al fichero "log/rns.log" en modo *append*, es decir, se van añadiendo contenidos.

### Sección "security"

En esta sección vamos a poder configurar todas las medidas de seguridad de la plataforma dependientes del RNS. Más específicamente podemos configurar las siguientes funcionalidades:

- `ssl`

  La propiedad `ssl` indica si la plataforma va a utilizar el mecanismo de seguridad de la capa externa (RMI sobre SSL) definido en la Sección 3.2.

  ```
  ssl = boolean
  ```

  En el arranque se podrá ver una línea en el *log* indicando que la plataforma está utilizando SSL:

  ```
  2013-08-27 09:25:54,048 -- INFO -- Started RNS at port 10000!
  2013-08-27 09:25:54,048 -- INFO -- SSL is set to true
  ```

- `authentication`

  La propiedad `authentication` indica si la plataforma va a utilizar el mecanismo de autenticación y autorización a nivel de plataforma definido en la Sección 3.3.

  ```
  authentication = boolean
  ```

  Si dicha propiedad es *true*, se habilitan las tres siguientes propiedades:

  ```
  authentication.agentPermission = string
  authentication.moveAgentPermission = string
  authentication.callAgentPermission = string
  ```

  Dichas propiedades indican el conjunto de permisos a otorgar, según lo explicado en la Sección 3.3. En cada propiedad se puede incluir el listado de contextos a los que se les otorga un permiso. Los contextos en el listado deben ir separados por comas.

  Por ejemplo, si queremos darle permisos de creación de agentes a los contextos "c1" y "c2", permisos de movimiento de agentes a los agentes creados en el contexto "c1" y permisos de llamada a otros agentes a los agentes creados en el contexto "c2", las líneas del fichero de configuración serían las siguientes:

```
authentication = true
authentication.agentPermission = c1, c2
authentication.moveAgentPermission = c1
authentication.callAgentPermission = c2
```

Una configuración así la veríamos reflejada en el *log* de la siguiente forma:

```
2013-08-27 09:26:44,650 -- DEBUG -- Starting RNS at port 10000...
2013-08-27 09:26:44,658 -- DEBUG -- Setting up permission
    AGENT_PERMISSION to c1
2013-08-27 09:26:44,658 -- DEBUG -- Setting up permission
    AGENT_PERMISSION to c2
2013-08-27 09:26:44,658 -- DEBUG -- Setting up permission
    MOVE_AGENT_PERMISSION to c1
2013-08-27 09:26:44,658 -- DEBUG -- Setting up permission
    CALL_AGENT_PERMISSION to c2
2013-08-27 09:26:49,927 -- INFO -- Started RNS at port 10000!
2013-08-27 09:26:49,927 -- INFO -- SSL is set to true
2013-08-27 09:26:49,927 -- INFO -- Platform Authentication is set to
     true
```

- `encryption`

  Para habilitar o deshabilitar el cifrado a nivel de plataforma se utiliza la siguiente propiedad:

  ```
  encryption = boolean
  ```

  Con lo que, si dicha propiedad tiene el valor *true*, se permitirá el uso de las funciones de cifrado y comprobación de integridad de datos definidas en la Sección 3.5.

  Podremos comprobar la activación de la capacidad de cifrado de la plataforma en el arranque del RNS buscando la siguiente línea en el log:

  ```
  2013-08-27 09:30:17,171 -- INFO -- Platform Encryption is set to
      true
  ```

- `keystore`

  Todas las funcionalidades que requieren certificados digitales necesitan la configuración de un *keystore* para el RNS. Por lo tanto, si alguna de las propiedades de seguridad (`ssl`, `authentication` o `encryption`) tiene valor *true* es necesario configurar un *keystore* válido. Dicho *keystore* se configura de la siguiente manera:

  ```
  keystore = string
  keystorePass = string
  ```

  Donde para la propiedad `keystore` debemos poner el nombre con la ruta completa del *keystore* a utilizar y en la propiedad `keystorePass` pondremos la clave que protege dicho *keystore*.

### C.1.3 Procedimientos de administración de los contextos

A continuación se detallan los principales procedimientos de administración de los contextos.

#### *Script* de arranque

Para facilitar el arranque de un contexto genérico se puede invocar un *script* de arranque que configura todas las variables necesarias para la correcta ejecución del proceso. Dicho *script* se llama `ContextLauncher`. El contenido del mismo es el siguiente:

```
export CLASSPATH=lib/log4j-1.2.15.jar:jar/springs.jar:classes
export PATH=$JAVA_HOME:$PATH
java -Djava.security.policy=etc/security.policy  -Djava.security.auth.
    login.config=etc/jaas.conf springs.context.ContextLauncher $*
```

En dicho *script* podemos configurar las variables `CLASSPATH` y `PATH` para adecuarlas a las rutas donde tengamos tanto el fichero JAR con el código de la plataforma como otras clases necesarias.

El *script* requiere un parámetro indicando la ruta del fichero de configuración del contexto:

```
# bin/ContextLauncher
Mandatory arguments missing:
 -c configuration file
```

Una vez introducido dicho parámetro obligatorio, el contexto leerá dicho fichero y se ejecutará.

Si el contexto arranca con éxito podremos ver una línea como la siguiente en el *log* (si éste se encuentra activado):

```
2013-08-26 16:35:55,871 -- INFO -- Started context at port 9501!
```

#### Ficheros de configuración

Al igual que en el RNS, se pueden configurar los aspectos básicos de un contexto mediante un fichero de configuración principal. Como se puede ver en el *script* de arranque, también existen otros dos ficheros de configuración que se utilizan en el caso de que esté activada la autenticación y autorización a nivel de contexto como veremos más adelante.

A continuación vamos a describir las secciones en las que se divide el fichero de configuración principal del contexto.

### Sección “general”

En esta sección se pueden configurar los aspectos generales del contexto. Más detalladamente podemos configurar las siguientes propiedades:

```
port = integer
```

Indica el puerto en el que el contexto escuchará peticiones.

```
name = string
```

Indica el nombre único en la región que adoptará el contexto.

```
addressRNS = string
```

Configura la dirección donde el RNS se encuentra escuchando peticiones. Por ejemplo, puede ser del tipo `addressRNS = rmi://host:10000`.

```
portClassServer = integer
```

Indica el puerto en el que estará escuchando el servidor de clases.

En un arranque correcto del contexto, se podrán ver las siguientes líneas en el *log*:

```
2013-08-26 16:36:41,690 -- DEBUG -- Launched RMI registry at port 9501!
2013-08-26 16:36:41,700 -- DEBUG -- Communicating new context to RNS at
    rmi://host:10000...
2013-08-26 16:36:41,702 -- DEBUG -- Contacting remote agent
    RegionNameServer at rmi://host:10000
2013-08-26 16:36:42,068 -- DEBUG -- Communicated new context to RNS at
    rmi://host:10000!
2013-08-26 16:36:42,069 -- DEBUG -- Creating ContextManager at port 9501
    with ssl = true...
2013-08-26 16:36:42,073 -- DEBUG -- Created ContextManager at port 9501
    with ssl = true!
2013-08-26 16:36:42,073 -- INFO -- Started context at port 9501!
```

### Sección “log”

En esta sección se configuran los aspectos referentes al *log* de la plataforma. Como hemos comentado, el log de la plataforma se basa en el paquete Log4j para el que se pueden encontrar todas la opciones de configuración en [87]. El siguiente es un ejemplo de configuración del *log*:

```
[log]

log4j.rootLogger=DEBUG, A1
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.A1.layout.ConversionPattern=%d -- %p -- %m%n
log4j.appender.A1=org.apache.log4j.RollingFileAppender
log4j.appender.A1.File=log/c1.log
```

En dicho ejemplo estamos configurando el *logging* del contexto para que se envíe todo lo que tenga prioridad "DEBUG" al fichero "log/c1.log" en modo *append*, es decir, se van añadiendo contenidos.

### Sección "security"

En esta sección vamos a poder configurar todas las medidas de seguridad del contexto. Depende de qué medida se trate, es necesario que el resto de elementos de la plataforma (principalmente el RNS) también la tengan activa. Más específicamente podemos configurar las siguientes funcionalidades:

- `ssl`

  La propiedad `ssl` indica si el contexto debe utilizar el mecanismo de seguridad de la capa externa (RMI sobre SSL) definido en la Sección 3.2. Hay que tener en cuenta que si el RNS tiene activo este mecanismo, el resto de contextos de la plataforma deben tenerlo también para comunicarse con él.

  ```
  ssl = boolean
  ```

  En el arranque se podrá ver un línea en el *log* indicando que el contexto está utilizando SSL:

  ```
  2013-08-26 16:36:42,073 -- DEBUG -- Created ContextManager at port
      9501 with ssl = true!
  ```

- `authentication`

  La propiedad `authentication` indica si el contexto va a utilizar el mecanismo de autenticación y autorización a nivel de contexto definido en la Sección 3.4. Hay que tener en cuenta que si la plataforma tiene desactivada la autenticación y la autorización a nivel de plataforma no es posible autenticar a los agentes que viajan a un contexto, con lo que la autenticación a nivel de contexto se desactiva automáticamente, no importando el valor que contenga la propiedad:

  ```
  authentication = boolean
  ```

  Si dicha propiedad es *true*, se habilita el uso de dos ficheros adicionales de configuración en los que se definen las medidas de seguridad a aplicar. Estos ficheros de configuración son los definidos en el *script* de arranque del contexto según las opciones `-Djava.security.auth.login.config` y

`-Djava.security.policy` de la invocación a `springs.context.ContextLauncher`.

El fichero de configuración definido en la opción `-Djava.security.auth.login.config` es un fichero de configuración cuyo contenido, en principio, es constante ya que, en la versión actual de la plataforma, no existen más que dos métodos de *login*, uno para los contextos y otro para los agentes. El contenido de dicho fichero de configuración debe de ser el siguiente:

```
Agent
{
    springs.security.AgentLoginModule required;
};
contexts
{
    springs.security.ContextLoginModule required;
};
```

El segundo fichero de configuración, correspondiente a la opción de configuración `-Djava.security.policy` es el fichero estándar `security.policy` [68] de la JVM en el que podemos configurar a qué recursos tiene acceso la máquina virtual sobre la que corre el contexto. En SPRINGS dicho fichero de configuración dispone de varias secciones. En primer lugar una común necesaria para que los mecanismos de autenticación funcionen:

```
grant {
    permission javax.security.auth.AuthPermission "
   createLoginContext";
    permission javax.security.auth.AuthPermission "doAs";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
    permission javax.security.auth.AuthPermission "modifyPrincipals
   ";
    permission javax.security.auth.AuthPermission "getSubject";
    permission javax.security.auth.AuthPermission "
   createLoginContext.Agent";
    permission java.net.SocketPermission "host", "accept, connect,
   resolve";
};
```

Es importante permitir las conexiones de red hacia y desde el servidor en el que reside el RNS

La siguiente sección del fichero se corresponde a los permisos con los que queremos que se ejecute el código del contexto SPRINGS en sí. Habitualmente le daremos todos los permisos posibles de la forma:

```
grant Principal springs.security.ContextPrincipal "context" {
    permission java.security.AllPermission "", "";
};
```

Por último podremos definir tantas secciones referentes a agentes como queramos controlar. Un ejemplo como el siguiente muestra que para cada agente creado en el contexto “Test”, permitimos el acceso en modo lectura al fichero `/etc/passwd`:

```
grant Principal springs.security.AgentPrincipal "Test" {
    permission java.io.FilePermission "/etc/passwd", "read";
};
```

En el fichero podremos poner tantas secciones `grant Principal` como queramos y, en ellas, podremos configurar tantos permisos como se necesiten.

Conviene hacer notar que los permisos son "en positivo", es decir, por defecto, un agente perteneciente a un `Principal` configurado no tiene acceso a ningún recurso (salvo a los recursos accesibles mediante los permisos de la sección común) excepto a los recursos que se le configuren dentro de su `Principal`.

Se pueden encontrar más detalles sobre la gestión particular de permisos en [70].

- `keystore`

  Todas las funcionalidades que requieren certificados digitales necesitan la configuración de un *keystore* para el contexto. Por lo tanto, si alguna de las propiedades de seguridad (`ssl` o `authentication`) tiene valor *true* es necesario configurar un *keystore* válido. Dicho *keystore* se configura de la siguiente manera:

  ```
  keystore = string
  keystorePass = string
  ```

  Donde para la propiedad `keystore` debemos poner el nombre con la ruta completa del *keystore* a utilizar y en la propiedad `keystorePass` pondremos la clave que protege dicho *keystore*.

## C.2 Guía del programador

La mayoría de los mecanismos de seguridad desarrollados están implementados a nivel interno de la plataforma, gestionados por el administrador de la misma. Sin embargo existen varias operaciones que un programador de agentes sobre la plataforma SPRINGS debería conocer para poder utilizar algunas funciones de seguridad.

Principalmente vamos a indicar cómo un programador puede crear un contexto que utilice los mecanismos de seguridad desarrollados y cómo puede utilizar los mecanismos de cifrado, descifrado y validación de integridad de datos durante la programación de agentes móviles. De la misma forma también describimos como utilizar el mecanismo de *logging* implementado en toda la plataforma.

### C.2.1 Uso de mecanismo de *logging*

Para ofrecer un mecanismo de *logging* unificado en toda la plataforma, utilizable tanto a nivel de contexto como a nivel de agente, se ha desarrollado la clase `springs.util.Log`,

que ofrece los siguientes métodos para escribir en el *log* según la severidad del mensaje que queremos imprimir:

```
/**
 * Logs a line with trace severity
 * @param line the line.
 * @see java.lang.String
 */
public void trace(String line);

/**
 * Logs a line with debug severity
 * @param line the line.
 * @see java.lang.String
 */
public void debug(String line);

/**
 * Logs a line with info severity
 * @param line the line.
 * @see java.lang.String
 */
public void info(String line);

/**
 * Logs a line with warn severity
 * @param line the line.
 * @see java.lang.String
 */
public void warn(String line);

/**
 * Logs a line with error severity
 * @param line the line.
 * @see java.lang.String
 */
public void error(String line);

 /**
 * Logs a line with fatal severity
 * @param line the line.
 * @see java.lang.String
 */
public void fatal(String line);
```

Para poder utilizar dichos métodos, tan solo hay que instanciar un objeto de la clase `Log` de la forma:

```
Log logger = new Log(configFile);
```

Donde `configFile` es el nombre del fichero de configuración del contexto o del RNS.

### C.2.2 Interfaz para la creación de contextos

Para poder crear y lanzar un contexto seguro a nivel programático, se ha creado un método específico que permite crear el contexto e introducir todas las opciones necesarias. Dicho método estático de la clase `springs.context.Context_RMIImpl` es:

```
public static void create(final String name, final int portNumber, final
    int portClassServer, final String addressRNS, final Boolean ssl,
    Boolean authentication, final KeyStore ks) throws
    ContextStartingException;
```

El cual acepta los siguientes parámetros:

- `name`. Es el nombre con el que queremos crear el contexto.
- `portNumber`. Es el número de puerto en el que estará escuchando el contexto.
- `portClassServer`. El puerto en el que se debe lanzar el `ClassServer`.
- `addressRNS`. La dirección en la que el RNS acepta peticiones.
- `ssl`. Si se debe utilizar el mecanismo de seguridad de la capa externa. En caso de que esta opción no sea congruente con la misma opción configurada en el RNS de la región, se devolverá una excepción del tipo `ContextStartingException`.
- `authentication`. Si se deben utilizar los mecanismos de autenticación y autorización a nivel de contexto.
- `ks`. El *keystore* donde se almacena el certificado del contexto.

Si se utiliza este método seremos capaces de crear un contexto que utilice las medidas de seguridad implementadas.

### C.2.3 Uso de funciones de cifrado en un agente

Otro mecanismo al que se tiene acceso desde el punto de vista del programador de agentes de la plataforma, es al cifrado, descifrado y validación de integridad de datos. Para cifrar un objeto se pueden utilizar los siguientes métodos de un agente:

```
public EncryptedData encrypt(Object obj) throws SecurityException;
public EncryptedData encrypt(Object obj, String agentName) throws
    SecurityException;
```

El primer método se debe utilizar para cifrar datos para ser leídos por el mismo agente en el contexto en el que se creó y el segundo para cifrar datos para ser enviados a otros agentes.

A continuación presentamos un ejemplo del cifrado de un dato para ser utilizado por el propio agente:

```
public static Log logger = new Log();
private String fraseSecreta = "Esta es una frase secreta";
private EncryptedData secreto;


public void main()
{
  try {
      secreto = encrypt(fraseSecreta);
      logger.info(getNameWithContext() + " Secreto cifrado, firmado por "
     +
           secreto.getSignatureContext());
  } catch (Exception e) {
       logger.error(e.toString());
  }
}
```

El siguiente ejemplo muestra cómo podemos cifrar un dato para que pueda ser leído por el agente "Agente-1":

```
public static Log logger = new Log();
private String fraseSecreta = "Esta es una frase secreta";
private EncryptedData secreto;


public void main()
{
  try {
      secreto = encrypt(fraseSecreta, "Agente-1");
      logger.info(getNameWithContext() + " Secreto cifrado, firmado por "
     +
           secreto.getSignatureContext());
  } catch (Exception e) {
       logger.error(e.toString());
  }
}
```

Los datos cifrados se encuentran en un objeto de la clase `springs.security.EncryptedData` pero no son accesibles directamente desde un agente.

Para descifrar un dato es necesario que el agente viaje al contexto en el que fue creado y, una vez allí, ejecute el siguiente método:

```
public Object decrypt(final EncryptedData data) throws SecurityException;
```

Que devolverá el objeto original.

Siguiendo con el ejemplo anterior, si se quiere descifrar el dato cifrado previamente, lo siguiente es lo que habría que hacer:

```
private EncryptedData secreto;

...
   goHome("end");
   return;
}

public void end()
{
    // Esta rutina se ha de ejecutar en el contexto en el que fue creado
    el agente.

    String datosEnClaro = null;
    try {
        logger.info(getNameWithContext() + " Vamos a intentar leer los
    datos cifrados");
         datosEnClaro = (String) decrypt(secreto);
         logger.info(getNameWithContext() + " los datos en claro son: " +
     datosEnClaro);
     } catch (Exception e) {
         logger.error("Exception: " + e.toString());
     }
}
```

Si además se quiere validar que los datos no han sido modificados desde el momento en el que se cifraron, conviene utilizar el método de verificación de la firma digital del dato cifrado:

```
public Boolean verifySignature(Object obj, EncryptedData encryptedObject)
     throws SecurityException;
```

Cuyos parámetros son el objeto descifrado y la estructura de datos cifrada, devolviéndose al usuario *true* si los datos cifrados no se han modificado en tránsito o *false* en caso contrario.

A continuación se muestra un ejemplo de la comprobación de integridad de un dato:

```
private EncryptedData secreto;

public void end()
{
    String datosEnClaro = null;
    try {
     logger.info(getNameWithContext() + " Vamos a intentar leer los datos
     cifrados");
          datosEnClaro = (String) decrypt(secreto);
          logger.info(getNameWithContext() + " los datos en claro son: "
    + datosEnClaro);
    } catch (Exception e) {
         logger.error("Exception: " + e.toString());
```

```
    }

    logger.info(getNameWithContext() + " Vamos a verificar los datos
    cifrados");
    try {
        Boolean verify = verifySignature(datosEnClaro, secreto);
        logger.info(getNameWithContext() + " los datos son los originales
     " + verify);
    } catch (Exception e) {
        logger.error("Exception: " + e.toString());
    }
}
```

# Bibliografía


[1] Danny B. Lange. "Mobile Objects and Mobile Agents: The Future of Distributed Computing?" In: *Proceedings of The European Conference on Object-Oriented Programming*. 1998.

[2] Tommy Thorn. "Programming languages for mobile code". In: *ACM Computing Surveys (CSUR)* 29.3 (1997), pp. 213–239.

[3] Danny B. Lange and Mitsuru Oshima. "Seven Good Reasons for Mobile Agents". In: *Communications of ACM* 42.3 (1999).

[4] S. Papastavrou, G. Samaras, and E. Pitoura. "Mobile Agents for WWW Distributed Database Access". In: *Proceedings 15th International Data Engineering Conference*. Sydney, Australia, 1999, pp. 228–237.

[5] Serge Fenet and Salima Hassas. "A Distributed Intrusion Detection and Response System Based on Mobile Autonomous Agents Using Social Insects Communication Paradigm". In: *First International Workshop on Security of Mobile Multiagent Systems, Autonomous Agents Conference*. May 2001.

[6] Timon C. Du, Eldon Y. Li, and An-Pin Chang. "Mobile agents in distributed network management". In: *Communications of ACM* 46.7 (July 2003), pp. 127–132.

[7] Giovanni Vigna, ed. *Mobile Agents and Security*. London, UK, UK: Springer-Verlag, 1998. ISBN: 3-540-64792-9.

[8] N. Borselius. "Mobile agent security". In: *Electronics and Communication Engineering Journal* 14.5 (Oct. 2002), pp. 211–218.

[9] David M. Chess, Colin G. Harrison, and Aaron Kershenbaum. *Mobile agents: Are they a good idea?* Tech. rep. RC 19887. IBM Research Report, Oct. 1994.

[10] G. Vigna. "Mobile Agents: Ten Reasons For Failure". In: *Proceedings of Mobile Data Management 2004*. Berkeley, CA, Jan. 2004, pp. 298–299.

[11] Wayne Jansen and Tom Karygiannis. *Mobile Agent Security*. Tech. rep. NIST special publication 800-19. Gaithersburg, US: National Institute of Standards and Technology, Computer Security Division, 1994. URL: `http://citeseer.ist.psu.edu/jansen00nist.html`.

[12] S. Ilarri, R. Trillo, and E. Mena. “SPRINGS: a scalable platform for highly mobile agents in distributed computing environments”. In: *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*. Los Alamitos, CA, June 2006, pp. 633–637.

[13] Peter Braun and Wilhelm Rossak. *Mobile agents. Basic concepts, mobility models and the tracy toolkit*. depunkt.verlag, 2005.

[14] W. A. Jansen. “Countermeasures for mobile agent security”. In: *Computer Communications* 23.5 (2000), pp. 1667–1676.

[15] Danny B. Lange et al. “Aglets: Programming Mobile Agents in Java.” In: *WWCA*. Ed. by Takashi Masuda, Yoshifumi Masunaga, and Michiharu Tsukamoto. Vol. 1274. Lecture Notes in Computer Science. Springer, Jan. 3, 2002, pp. 253–266. ISBN: 3-540-63343-X. URL: `http://dblp.uni-trier.de/db/conf/wwca/wwca97.html#LangeOKK97`.

[16] G. Glass. “Mobility”. In: ed. by Dejan Milojiči&cacute, Frederick Douglis, and Richard Wheeler. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. Chap. ObjectSpace Voyager core package technical overview, pp. 611–627. ISBN: 0-201-37928-7. URL: `http://dl.acm.org/citation.cfm?id=303461.342806`.

[17] Haiping Xu and Sol M. Shatz. “ADK: An Agent Development Kit Based on a Formal Design Model for Multi-Agent Systems”. In: *Autom. Softw. Eng.* 10.4 (2003), pp. 337–365.

[18] Volker Roth and Mehrdad Jalali-Sohi. “Concepts and Architecture of a Security-Centric Mobile Agent Server”. In: *Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems*. ISADS ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 435–. ISBN: 0-7695-1065-5.

[19] ACL Fipa. “FIPA ACL Message Structure Specification”. In: *Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00061/SC00061G.html (30.6. 2004)* (2002).

[20] Tim Finin et al. “KQML as an agent communication language”. In: *Proceedings of the third international conference on Information and knowledge management*. ACM. 1994, pp. 456–463.

[21] Carl E Landwehr et al. “A taxonomy of computer program security flaws”. In: *ACM Computing Surveys (CSUR)* 26.3 (1994), pp. 211–254.

[22] Burt Kaliski. “PKCS# 7: Cryptographic Message Syntax Version 1.5”. In: (1998).

[23] John Larmouth. *ASN. 1 complete*. Morgan Kaufmann, 2000.

[24] International Telecommunication Union. *The Directory — Models*. ITU-T Recommendation X.501. Nov. 1993.

[25] Dave Cooper. “Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile”. In: (2008).

[26] U. Pinsdorf and V. Roth. "Mobile agent interoperability patterns and practice". In: *Engineering of Computer-Based Systems, 2002. Proceedings. Ninth Annual IEEE International Conference and Workshop on the*. IEEE. 2002, pp. 238–244. ISBN: 0769515495.

[27] F. Bellifemine et al. "JADE: A White Paper". In: *EXP in search of innovation* 3.3 (2003), pp. 6–19. URL: http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf.

[28] Regine Endsuleit and Jacques Calmet. "A security analysis on JADE(-S) V. 3.2". In: *Proceedings of NordSec*. 2005, pp. 20–28.

[29] Troy Bryan Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc., 1998.

[30] Giosue Vitaglione. *Mutual-authenticated SSL IMTP connections*. Telecom Italia LAB, 2004. URL: http://jade.tilab.com/doc/tutorials/SSL-IMTP/SSL-IMTP.doc.

[31] Xosé A. Vila Sobrino, A. Schuster, and Adolfo Riera. "Security for a Multi-Agent System based on JADE." In: *Computers and Security* 26.5 (2007), pp. 391–400. URL: http://dblp.uni-trier.de/db/journals/compsec/compsec26.html#SobrinoSR07.

[32] *JADE security guide*. JADE Board. 2005. URL: http://jade.tilab.com.

[33] D.S. Milojičić, F. Douglis, and R. Wheeler. *Mobility: processes, computers, and agents*. ACM Press Series. Addison-Wesley, 1999. ISBN: 9780201379280. URL: http://books.google.es/books?id=JbxQAAAAMAAJ.

[34] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. "Aglets specification 1.1 draft". In: *Whitepaper Draft 0.65, Sept* 8 (1998).

[35] Günter Karjoth, Danny B Lange, and Mitsuru Oshima. "A security model for aglets". In: *Internet Computing, IEEE* 1.4 (1997), pp. 68–77.

[36] Haiping Xu and Sol M Shatz. "Adk: An agent development kit based on a formal design model for multi-agent systems". In: *Automated Software Engineering* 10.4 (2003), pp. 337–365.

[37] BV Tryllian. "Agent Development Kit". In: (2001).

[38] Diana Dong. "Java Applet Security". In: (2004).

[39] Graham Glass. "Overview of voyager: Objectspace's product family for state-of-the-art distributed computing". In: *CTO ObjectSpace* (1999).

[40] Steve Vinoski. "CORBA: Integrating diverse applications within distributed heterogeneous environments". In: *Communications Magazine, IEEE* 35.2 (1997), pp. 46–55.

[41] Raquel Trillo, Sergio Ilarri, and Eduardo Mena. "Comparison and performance evaluation of mobile agent platforms". In: *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*. IEEE. 2007, pp. 41–41.

[42] Ritu Gupta and Gaurav Kansal. “A Survey on Comparative Study of Mobile Agent Platforms”. In: *International Journal of Engineering Science and Technology* 3.3 (2011), pp. 1943–1948.

[43] *Voyager Security Developer’s Guide version 1.2 for Voyager 8.0*. Recursion Software Inc. 2011. URL: `http://www.recursionsw.com/products/voyager-tech-docs/`.

[44] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. “Weak and strong mobility in mobile agent applications”. In: *Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000), Manchester (UK)*. 2000.

[45] S.A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. MIT Press, 2000. ISBN: 9780262024914. URL: `http://books.google.com/books?id=U8VUaUiYohIC`.

[46] D. Cooper et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). Internet Engineering Task Force, May 2008. URL: `http://www.ietf.org/rfc/rfc5280.txt`.

[47] Oscar Urra et al. “Mobile Agents and Mobile Devices: Friendship or Difficult Relationship?” In: *Journal of Physical Agents* 3.2 (2009), pp. 27–37.

[48] David Recordon and Drummond Reed. “OpenID 2.0: a platform for user-centric identity management”. In: *Proceedings of the second ACM workshop on Digital identity management*. DIM ’06. Alexandria, Virginia, USA: ACM, 2006, pp. 11–16. ISBN: 1-59593-547-9. DOI: `http://doi.acm.org/10.1145/1179529.1179532`. URL: `http://doi.acm.org/10.1145/1179529.1179532`.

[49] Luis Miguel Alventosa. *Using the SSL/TLS-based RMI Socket Factories in J2SE 5.0*. May 2006. URL: `https://blogs.oracle.com/lmalventosa/entry/using_the_ssl_tls_based`.

[50] Krishnan Viswanath. *The New RMI*. Oct. 2005. URL: `https://today.java.net/pub/a/today/2005/10/06/the-new-rmi.html`.

[51] Neil Haller et al. *A one-time password system*. Tech. rep. RFC 1938, May, 1996.

[52] Java Authentication. *Authorization Service (JAAS). Reference Guide for the J2SE Development Kit 5.0*. 2001.

[53] Gustavus J Simmons. “Symmetric and asymmetric encryption”. In: *ACM Computing Surveys (CSUR)* 11.4 (1979), pp. 305–330.

[54] J Jonsson and B Kaliski. “RFC 3447: Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1”. In: *Request for Comments (RFC)* 3447 (2003).

[55] Gary Stoneburner. *SP 800-33. Underlying Technical Models for Information Technology Security*. Tech. rep. Gaithersburg, MD, United States, 2001.

[56] Kai Rannenberg. “Recent Development in Information Technology Security Evaluation-The Need for Evaluation Criteria for Multilateral Security.” In: *Security and Control of Information Technology in Society*. 1993, pp. 113–128.

[57] Gianpaolo Cugola et al. “Analyzing mobile code languages”. In: *Mobile Object Systems Towards the Programmable Internet*. Springer, 1997, pp. 91–109.

[58] James E White. “Telescript technology: The foundation for the electronic marketplace”. In: *General Magic white paper* 282 (1994).

[59] Robert S Gray. “Agent Tcl: A flexible and secure mobile-agent system”. In: (1997).

[60] Robert S Gray et al. “D’Agents: Security in a multiple-language, mobile-agent system”. In: *Mobile agents and security*. Springer, 1998, pp. 154–187.

[61] Douglas Kramer. “The Java Platform”. In: *White Paper, Sun Microsystems, Mountain View, CA* (1996).

[62] Li Gong. “Java 2 platform security architecture”. In: *Sun Microsystems (http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html)* (2002).

[63] S. Oaks. *Java security*. Java series. O’Reilly, 2001. ISBN: 9780596001575. URL: `http://books.google.com.au/books?id=EhX9BjHj9M4C`.

[64] Xavier Leroy. “Java bytecode verification: an overview”. In: *Computer aided verification*. Springer. 2001, pp. 265–285.

[65] Sheng Liang and Gilad Bracha. “Dynamic class loading in the Java virtual machine”. In: *ACM SIGPLAN Notices* 33.10 (1998), pp. 36–44.

[66] Oracle. *Java Security Package*. URL: `http://docs.oracle.com/javase/6/docs/api/java/security/package-summary.html`.

[67] Walter Binder and Volker Roth. “Security Risks in Java-based Mobile Code Systems”. In: *Scalable Computing: Practice and Experience* 7.4 (2001).

[68] Michael Coté. *Jaas in action*. 2010.

[69] Giacomo Cabri, Luca Ferrari, and Letizia Leonardi. “Applying security policies through agent roles: A JAAS based approach”. In: *Science of Computer Programming* 59.1 (2006), pp. 127–146.

[70] Oracle. *Permissions in the Java™ SE 6 Development Kit (JDK)*. URL: `http://docs.oracle.com/javase/6/docs/technotes/guides/security/permissions.html`.

[71] Apple. *Macbook Pro*. URL: `https://www.apple.com/macbook-pro/`.

[72] Bjorn Winckler. *Macvim*. URL: `https://code.google.com/p/macvim/`.

[73] IDE Eclipse. *The Eclipse Foundation*. 2007.

[74] Apache Ant. *The Apache Ant Project*. 2010.

[75] Douglas Kramer. “API documentation from source code comments: a case study of Javadoc”. In: *Proceedings of the 17th annual international conference on Computer documentation*. ACM. 1999, pp. 147–153.

[76] Dimitri van Heesch. *Doxygen: Source code documentation generator tool*. 2008.

[77] Ben Collins-Sussman. “The subversion project: buiding a better CVS”. In: *Linux Journal* 2002.94 (2002), p. 3.

[78] Jorge Sainz Vela. *Zonazener*. 2003. URL: `http://www.zonazener.com`.

[79] Leslie Lamport and A LaTEX. *Document Preparation System*. 1986.

[80] Juan Manuel Alor Osorio. “Evaluación de la herramienta EJBCA para un Prestador de Servicios de Certificación”. In: (2011).

[81] VM Oracle. *VirtualBox*. 2011. URL: `https://www.virtualbox.org`.

[82] Larry Wall et al. *The Perl programming language*. 1994.

[83] Van Jacobson, Craig Leres, Steven McCanne, et al. *Tcpdump*. 1989.

[84] Gerald Combs et al. “Wireshark”. In: *Web page: http://www.wireshark.org/last modified* (2007), pp. 12–02.

[85] Ceki Gulcu. “Short introduction to log4j”. In: (2002).

[86] Eric A Young, Tim J Hudson, and Ralf S Engelschall. “OpenSSL”. In: *World Wide Web, http://www.openssl.org/, Last visited* 9 (2001).

[87] M Chauhuan. “Logging with Log4j-An Efficient Way to Log Java Applications”. In: *Developer.com, http://www.developer.com/java/ent/article.php/10933_3097221_3* ().